"Języki Atari XL/XE" zostały pomyślane jako vademecum dla wszystkich, którzy piszą programy w różnych językach dostępnych na ośmiobitowe komputery Atari. Pożyteczne informacje znajda tu zarówno początkujący programiści, stawiający pierwsze kroki, jak również autorzy profesjonalnych programów w kilku najpopularniejszych językach dostępnych dla komputerów Atari XL/XE. Druga część książki zawiera opisy języków kompilowanych: Kyan Pascal, Deep Blue C i Action!.

Książka ta nie Jest podręcznikiem do nauki języków programowania, lecz stanowi podręczne kompendium wiedzy o językach. Zadaniem tej książki jest pomoc w rozwiązywaniu problemów pojawiających się podczas pisania programów. Zawarte tu informacje mogą być wykorzystywane w rożnym stopniu, zależnie od poziomu posiadanej wiedzy. Dla użyt-kowników nie znających zasad programowania komputerów książka ta będzie przydatna w przyszłości, gdy poznają już podstawy tej sztuki.

"Języki Atari XL/XE" sa przeznaczone dla szerokiego kręgu odbiorców i stanowią dalszy ciąg wydanego przez SOETO "Poradnika programisty Atari".

WPROWADZENIE

"Języki Atari XL/XE" są przewidziane jako źródło informacji dla wszystkich, którzy pisząc programy chcą maksymalnie wykorzystać wbudowane możliwości komputera. Zawarte tu informacje pomogą przy tworzeniu programów w kilku najpopularniejszych językach dostępnych dla komputerów Atari XL/XE. Mam nadzieję, że dzięki temu książka ta stanie się niezbędną pozycją w bibliotece każdego programisty, zarówno początkującego, jak i profesjonalisty. Każdy znajdzie tu informacje pozwalające wybrać język programowania odpowiedni dla rozwiązywanego zagadnienia oraz rozwinąć pisany w nim program.

"Języki Atari XL/XE" nie są podręcznikiem do nauki języków programowania. Jednakże stanowią one podręczne kompendium wiedzy o językach. Zadaniem tej książki jest pomoc w rozwiązywaniu problemów pojawiających się podczas pisania programów w różnych językach. Dla użytkowników nie znających zasad programowania komputerów książka ta będzie przydatna w przyszłości, gdy poznają już podstawy tej sztuki.

"Języki Atari XL/XE" zawierają informacje, które mogą być wykorzystane w zależności od poziomu posiadanej wiedzy. Oznacza to, że początkujący programista nie będzie mógł skorzystać ze wszystkich zawartych tu wiadomości. Wytrawny znawca zagadnień programowania uzna natomiast, że część informacji jest zupełnie zbędna, gdyż dawno już znana. Książka ta jest przeznaczona dla szerokiego kręgu odbiorców i stanowi jakby dalszy ciąg wydanego przez SOETO "Poradnika programisty Atari". Znajduje się tu wiele odwołań do treści zawartych w "Poradniku" i każdy programista powinien posiadać obie te książki.

Opisy języków programowania mają znaczną objętość. Zrezygnowałem z wprowadzania ograniczeń i znacznych skrótów, aby nie utrudniać korzystania z książki. Spowodowało to jednak konieczność podzielenia jej na dwie części. Część pierwsza zawiera opisy języków interpretowanych: Turbo—Basic XL, Basic XE, Microsoft Basic i Logo, a druga — opisy języków kompilowanych: Kyan Pascal, Deep Blue C i Action!.

Zasady opisu

Interpretery i kompilatory języków używają pewnych słów do oznaczenia wykonywanych operacji. Są to nazwy instrukcji i funkcji języka, czyli tzw. słowa kluczowe. Ich użycie w sposób inny niż określony zasadami składni języka może spowodować błąd. Należy więc przestrzegać podanych tu zasad, a odstępstwa od nich są dozwolone jedynie w uzasadnionych przypadkach.

Ponadto opisywane języki posiadają wiele dodatkowych procedur (instrukcji i funkcji), które nie są standardowymi operacjami, lecz zostały dołączone w celu rozszerzenia możliwości języka. Zebrane są one w bibliotece języka lub w dodatkowych plikach dołączanych do tworzonego programu. Procedury te zostały również opisane w słownikach.

W całej książce jest zastosowana jednolita konwencja opisu składni (struktury) instrukcji. Ponadto każdy wzór składni jest poparty przykładami. Reguły interpretacji poszczególnych symboli składniowych są następujące:

1. Sposób pisania słów kluczowych jest różny w różnych językach. Szczegółowe zasady są podane na początku każdego rozdziału.
2. Wyrażenia umieszczone w nawiasach trójkątnych (< i >) określają, co powinno znaleźć się w tym miejscu w poprawnej instrukcji.
3. Wyrażania umieszczona w nawiasach kwadratowych ([i]) oznaczają dowolny parametr instrukcji. Parametr taki nie jest konieczny, lecz może być użyty w razie potrzeby.
4. Wielokropek (...) oznacza dowolną liczbę powtórzeń podanego przed nim elementu instrukcji. Element taki może być umieszczony w instrukcji tyle razy, ile to jest konieczna.
5. Znak podobny do dwukropka, lecz złożony z dwóch pionowych linii (|) symbolizuje wybór. Oznacza to konieczność użycia jednego i tylko jednego z elementów, między którymi ten znak się znajduje.

W słownikach przyjęta została następująca kolejność opisu: nazwa, typ, składnia, przykłady i działanie. W nazwie uwzględnione jest słowo kluczowe oraz skrót, jeśli występuje. Typ określa rodzaj słowa kluczowego: instrukcja, funkcja liczbowa, funkcja tekstowa lub operator. Składnia opisuje przy użyciu symboli podanych we wprowadzeniu dozwolona sposoby zapisu słowa kluczowego. Pozostałe punkty nie wymagają objaśniania.

Rozdział 1

KYAN PASCAL

Kyan Pascal został napisany w roku 1985 w amerykańskiej firmie Kyan Software. Jest to implementacja języka Pascal dla ośmiobitowych komputerów Atari- Zamieszczony dalej opis dotyczy wersji 2.0. Kyan Pascal wczytywany jest z dyskietki, która. zawiera edytor, kompilator i bibliotekę. Ponadto na dyskietce znajdują się dodatkowe procedury, które można dołączyć do własnego programu. Wersje kasetowe tych programów zastały opracowane w Polsce.

Kyan Pascal jest pełną implementacją języka wzorcowego Pascal i zawiera wszystkie przewidziane w nim struktury programowe. Do nauki programowania najlepszym podręcznikiem będzie więc opis języka wzorcowego pod tytułem "Pascal" autorstwa M. Iglewskiego, J. Madeya i S. Matwina, a wydany przez WNT. Ponadto kursy programowania w Pascalu publikowane były w wielu czasopismach (m. in. "Bajtek", "IKS", "Młody Technik").

Kyan Pascal odczytywany jest automatycznie z dyskietki znajdującej się w stacji dysków numer 1 w chwili włączenia komputera. Sposób uruchomienia edytora i kompilatora jest podany w rozdziałach opisujących te części systemu. Programy napisane w Kyan Pascalu mogą być uruchamiane samodzielnie pod warunkiem, że biblioteka zawarta w pliku LIB znajduje się na tej samej dyskietce lub została dołączona do programu. Dotyczy to również programów zapisanych pod nazwą AUTORUN.SYS, które samoczynnie wczytują się i uruchamiają po włączeniu komputera.

Sposób wykorzystania pamięci komputera przez edytor i kompilator Kyan Pascala jest dla użytkownika zupełnie nieistotny. Ważne jest jedynie położenie w pamięci gotowego, skompilowanego programu. Obszar od adresu 37888 ($9400) do 48127 ($BBFF) zajmuje biblioteka (LIB). Powyżej znajduje się pamięć obrazu w trybie 0. Dla programu oraz jego danych i stosu pozostaje miejsce od szczytu DOS—u do początku biblioteki. Obszar ten rozciąga się od adresu 8192 ($2000) do 37B87 ($93FF). Skompilowany program wczytywany jest normalnie od adresu 8192, co można zmienić poprzez dyrektywę ORG. Dostępny obszar pamięci zmniejsza się znacznie przy stosowaniu trybów graficznych, które wymagają więcej miejsca niż tryb 0, ze względu na konieczność przeniesienia pamięci obrazu poniżej biblioteki. Do dyspozycji użytkownika pozostaje ponadto szósta strona pamięci RAM (1536— 1791 = $0600-$06FF).

Słowa kluczowe, operatory i nazwy zmiennych mogą być wprowadzane zarówno małymi, jak i dużymi literami — kompilator Kyan Pascal nie rozróżnia ich między sobą.

Po wczytaniu Kyan Pascala na ekranie pojawia się jego nazwa oraz znak ">". W celu przejścia do edytora należy wpisać "D:ED" i nacisnąć klawisz <RETURN>. Powoduje to wczytanie oraz uruchomienie edytora (wersja 1.1), który zgłasza się podając swoją nazwę oraz pytanie "FILENAME ?". Na pytanie to należy odpowiedzieć wpisując specyfikację pliku (koniecznie razem z nazwą urządzenia, np. "D:PROGRAM.I").

Wskazany plik, zawierający program źródłowy w Pascalu, jest odczytywany z dyskietki i można przystąpić do jego redagowania. Jeżeli na dyskietce nie ma pliku o podanej nazwie, to edytor informuje o tym komunikatem "FILE NOT FOUND". W takim przypadku można rozpocząć pisanie nowego programu, a podana nazwa zostaje zapamiętana i posłuży do jego późniejszego zapisania.

Podczas pracy edytora można wykorzystać wiele pomocniczych funkcji. Część z nich uzyskuje się przez naciskanie klawiszy literowych wraz z <CONTROL>, a niektóre dostępne są po uprzednim naciśnięciu <ESC>. Klawisz <ESC> powoduje ponadto wyświetlenie na ekranie znaczenia tych funkcji oraz trzech dodatkowych informacji. Kolejno na ekranie ukazują się: aktualna nazwa pliku oraz ciągi "A" i "B". Ciągi te są wykorzystywane do wymieniania pewnych fragmentów tekstu pisanego programu.

Naciśnięcie <ESC> udostępnia użytkownikowi następujące funkcje edytora:

A - ustalenie ciągu "A";
B - ustalenie ciągu "B";
C — zmiana ciągu "A" na "B";
G — przejście do wskazanego wiersza programu;
H - wyświetlenie funkcji osiąganych z <CONTROL>;
I - dołączenie innego pliku źródłowego;
P - ustalenie nazwy aktualnie redagowanego pliku;
Q - przerwanie pracy edytora bez zapisu zawartości;
S - zapisanie zawartości edytora i kontynuowanie pracy;
X — zapisanie zawartości edytora i zakończenie jego pracy;
ESC - powrót do edytora.

Klawisze <A> i <B> służą do ustalenia ciągów, przy czym ciąg "A" jest wykorzystywany przez wszystkie funkcje przeszukiwania i wymiany, a ciąg "B" tylko przez funkcję wymiany. Aktualnie ustalony ciąg jest wyświetlany na ekranie. Maksymalna długość ciągu jest równa 40 znaków.

Klawisz <C> pozwala na zastąpienie ciągu "A" przez ciąg "B". Po jego naciśnięciu należy wybrać odpowiednim klawiszem wymianę wszystkich (All) lub niektórych (Some) ciągów "A" na "B" albo przerwanie funkcji (Quit). Po wskazaniu drugiego z wymienionych wariantów dla każdego pokazanego ciągu "A" trzeba wybrać wymianę (Yes) lub nie (No) albo przerwanie funkcji (.Quit). Funkcja ta pozwala na przyspieszenie wpisywania programu, dzięki możliwości zastąpienia często używanych słów kluczowych skrótami i zamianie ich ria pełne słowa po zakończeniu pisania programu.

Po naciśnięciu klawisza <G> należy podać numer wiersza programu, a edytor ustawia na nim kursor. Wiersze programu w

Pascalu nie są numerowane, więc podawany numer jest kolejnym numerem logicznego wiersza programu źródłowego, licząc cd jego początku. Funkcja ta jest szczególnie przydatna podczas poprawiania błędów wykrytych w czasie kompilacji, gdyż kompilator podaje numer wiersza, w którym został napotkany błąd.

Wyświetlenie funkcji osiąganych przez naciśnięcie klawiszy literowych razem z <CONTROL> uzyskuje się poprzez <H>. Wykaz i opis tych funkcji znajduje się w dalszej części rozdziału.

Naciśnięcie klawisza <I> umożliwia dołączenie innego pliku do tworzonego programu źródłowego. Plik ten jest dołączany na. aktualnej pozycji kursora. Można dzięki temu korzystać z gotowych podprogramów i procedur, co znacznie przyspiesza pracą. Trzeba pamiętać, że wymagane jest zawsze podanie pełnej specyfikacji dołączanego pliku (z nazwą i ewentualnie numerem urządzenia).

Jeżeli konieczna jest zmiana nazwy redagowanego pliku, ta trzeba nacisnąć klawisz <P> i wpisać nową specyfikacją. Samo naciśnięcie klawisza <RETURN> pozostawia bez zmiany poprzednio ustaloną specyfikacje.

Przerwanie pracy edytora bez zapisu zawartości powoduje klawisz <Q>. Ponieważ można w ten sposób pomyłkowo skasować rezultat wielogodzinnej pracy, to edytor upewnia się pytaniem "THE CHANGES MADA HAVE NOT BEEN SAVED, ARE YOU SURE (Y/N)?" (dokonane zmiany nie zostały zapisane, czy jesteś pewny"?). Teraz klawisz <Y> kończy prace edytora, zaś <N> przerywa funkcję.

klawisz <S> pozwala na zapisanie bieżącej zawartości edytora i kontynuowanie redagowania programu. Zapisywany plik otrzymuje nazwę ustaloną przy uruchamianiu edytora lub przy użyciu klawisza <P>. Służy to głównie do zapisywania kolejnych etapów pracy, a więc zabezpiecza przed niespodziewanymi wyłączeniami prądu i umożliwia tworzenie kilku wersji programu.

Po zakończeniu redagowania programu klawisz <X> powoduje jego zapisanie i przerywa pracą edytora. Nazwa pliku do zapisu jest określana identycznie jak w funkcji "S". Ponowne uruchomienie edytora jest możliwe po wpisaniu "!", o ile nie był tymczasem wczytywany kompilator lub inny program. W takim przypadku konieczne jest powtórne odczytanie edytora z dyskietki.

Poniżej wymienione są klawisze, które naciśnięte razem z <CONTROL> wywołują dodatkowe funkcje edytora. Wykaz tych funkcji jest wyświetlany po naciśnięciu kolejno <ESC> i <H>.

A — przesuniecie kursora o jedno słowo w lewo;
C — przesunięcie kursora o 20 wierszy w dół;
D — przesunięcie kursora o jeden znak w prawo;
E — przesunięcie kursora o jeden wiersz w górą;
F — przesunięcie kursora o jedno słowo w prawo;
G - skasowanie znaku spod kursora;
O - usunięcie fragmentu tekstu;
P — wstawienie fragmentu tekstu;
Q — skasowanie znaku z lewej strony kursora;
R - przesunięcie kursora o 20 wierszy w górę;
S - przesunięcie kursora o jeden znak w lewo;
T — przesunięcie kursora na początek pliku;

V - przesunięcie kursora na koniec pliku;
W - przeszukiwanie pliku od kursora do początku;
X - przesunięcie kursora o jeden wiersz w dół;
Y - usunięcie wiersza spod kursora;
Z - przeszukiwanie pliku od kursora do końca;

Większość tych kombinacji klawiszy służy do przemieszczania kursora wewnątrz redagowanego tekstu programu. Ich znaczenie jest oczywiste i nie wymaga specjalnego opisu.

Trzy dalsze kombinacje mają odpowiedniki w funkcjach klawiszy edytora ekranowego Atari. Kombinacji <CONTROL> i <G> odpowiadają klawisze <CONTROL> i <DELETE>, kombinacji <CONTROL> i <Q> odpowiada <DELETE>, zaś <CONTROL> i <Y> ma identyczne znaczenie jak <SHIFT> i <DELETE>.

Naciśnięcie kombinacji klawiszy <CONTROL> i <O> zaznacza początek fragmentu tekstu do wycięcia. Po przesunięciu kursora na koniec wycinanej części i powownym naciśnięciu <CONTROL> i <O> następuje usunięcie tego fragmentu (zaznaczonego na ekranie przez wyświetlenie w negatywie). Usunięty w ten sposób tekst jest umieszczany w specjalnym buforze, przy czym uprzednia zawartość tego bufora jest kasowana.

Klawisze <CONTROL> i <P> powodują wstawienie na aktualnej pozycji kursora fragmentu tekstu, który znajduje się w buforze (umieszczony tam przez <CONTROL> i <O>). Bufor nie jest przy tym kasowany, więc wybrany fragment można powielić dowolną ilość razy.

Dwie ostatnie kombinacje klawiszy służą do przeszukiwania redagowanego tekstu. Poszukiwany ciąg znaków ustala się funkcją <ESC> <A>. Teraz każde naciśnięcie klawiszy <CONTROL> i <W> lub <C0NTR0L> i <Z> powoduje ustawienie kursora na pierwszym znaku poszukiwanego ciągu. Klawisz <W> wywołuje przy tym przeszukiwanie tekstu od aktualnej pozycji kursora w stronę początku pliku, zaś klawisz <Z> w stronę końca pliku.

1.2. Kompilator

Po wczytaniu Kyan Pascala na ekranie pojawia się jego nazwa oraz znak ">". W celu przejścia do kompilatora należy wpisać "D:PC" i nacisnąć klawisz <RETURN>. Powoduje to wczytanie oraz uruchomiania kompilatora (wersja 1.2), który zgłasza się podając swoją nazwę oraz napis "PC>". Należy po nim wpisać specyfikacją pliku (koniecznie razem z nazwą urządzenia, np. "D:PROGRAM.I") przeznaczonego do kompilacji oraz ewentualnie żądane warianty wykonania tej operacji (z myślnikiem przed każdym z nich).

Kompilator Kyan Pascal umożliwia ustalenie trzech niezależnych wariantów wykonania kompilacji oznaczonych odpowiednio literami "E", "L" i "O". Znaczenie ich jest następujące:

Wariant "E" służy do wyboru urządzenia, na które będą wyprowadzane komunikaty o błędach kompilacji. Normalnie urządzeniem tym jest ekran, a wpisanie "-EP" powoduje wydrukowanie wykazu błędów kompilacji na drukarce.

Wariant "L" służy do wyboru urządzenia, na które będzie wyprowadzony listing skompilowanego programu. Normalnie listing nie jest wyprowadzany. Po wpisaniu "-L" urządzeniem tym jest ekran, a "—LP" powoduje wydrukowanie listingu na drukarce.

Wariant "O" wybiera nazwę wynikowego pliku kompilacji. Przyjęte jest nadawanie programom źródłowym nazw z rozszerzeniem "I", zaś wynikowym — z rozszerzeniem "O". Brak wariantu "G" powoduje zapisanie skompilowanego programu w pliku o tak ustalonej nazwie. Wpisanie samego "-O" blokuje tworzenie kodu wynikowego - program zostaje skompilowany, lecz nie jest nigdzie zapamiętywany. W celu zapisania programu wynikowego w innym pliku należy wpisać "—O" ze specyfikacją tego pliku, na przykład: "Dl:PROGRAM.I-OD1:PROGRAM.OBJ". Pomiędzy literą "O" i specyfikacją nie może być odstępu.

Skompilowany program jest zapisany we wskazanym pliku tylko w przypadku, gdy kompilacja przebiegła bezbłędnie. W przeciwnym przypadku należy wczytać edytor, poprawić zasygnalizowane błędy, znowu wczytać kompilator i ponownie skompilować program. Procedura ta jest nieco uciążliwa i bardzo wygodne jest w tym przypadku korzystanie z ramdysku. Opis błędów sygnalizowanych przez kompilator znajduje się w rozdziale 1.7. "Meldunki błędów".

Po zakończeniu kompilacji następuje zawsze przejście do głównego menu Kyan Pascala. Ponowne uruchomienie kompilatora jest możliwe po wpisaniu "!", o ile nie był tymczasem wczytywany edytor lub inny program. W takim przypadku konieczne jest powtórne odczytanie kompilatora z dyskietki. Skompilowany program można uruchomić z poziomu menu głównego podając specyfikację pliku lub z poziomu DOS—u poprzez funkcję BINARY LOAD.

1.3. Technika programowania

Program w Pascalu musi być zbudowany dokładnie według ustalonego schematu. Każde odstępstwo od tego schematu powoduje błąd podczas kompilacji. Struktura programu w Pascalu jest jednak na tyle elastyczna, że umożliwia rozwiązanie niemal każdego problemu.

Każdy program w Pascalu składa się z nagłówka oraz bloku właściwego programu i musi być zakończony kropką. Poszczególne wiersze programu nie są numerowane. Instrukcje są oddzielane od siebie średnikami lub niektórymi słowami kluczowymi. Podział programu na wiersze nie ma znaczenia dla kompilatora, lecz służy, wyłącznie do poprawienia czytelności i zrozumiałości programu.

Nagłówek składa się ze słowa kluczowego "PROGRAM" i nazwy, która może być dowolnym poprawnym identyfikatorem. Po nazwie programu należy umieścić w nawiasie okrągłym identyfikatory (nazwy) plików, które będą wykorzystywane przez program. Jeśli program nie korzysta z żadnych plików, to ten element może być pominięty. Nagłówek programu musi kończyć się średnikiem.

Blok programu jest właściwym programem w Pascalu i składa się z następujących części (w nawiasie podane jest słowo kluczowe, które rozpoczyna daną część):

```
deklaracje etykiet (LABEL)
definicje stałych (CONST)
definicje typów (TYPE)
deklaracje zmiennych (VAR)
deklaracje procedur i funkcji
            (PROCEDURE i FUNCTION)
część operacyjna (od BEGIN do END)
```

Wymienione wyżej elementy programu muszą występować zawsze w podanej kolejności. Konieczny jest jednak tylko ostatni element (część operacyjna), a pozostałe są stosowane w razie potrzeby i mogą być pominięte.

Część operacyjna programu musi być ograniczona słowami kluczowymi "BEBIN" i "END". Między nimi znajdują się instrukcje programu, w tym również wywołania procedur i funkcji określonych wcześniej (w poprzedniej części). Pozostałe części programu są opisane w słowniku wraz z odpowiadającymi im słowami kluczowymi.

Instrukcje w Pascalu mogą być instrukcjami prostymi lub złożonymi. Instrukcje proste są to polecenia wykonania pojedynczych operacji, a ich opisy znajdują się w słowniku. Instrukcje złożone zawierają kilka instrukcji prostych oddzielonych od siebie średnikami i są ograniczone słowami "BEGIN" i "END".

Komentarze mogą Dyć umieszczane w dowolnym miejscu programu. Ponieważ zestaw znaków Atari nie zawiera nawiasów klamrowych, które w Pascalu oznaczają komentarz, to zostały one zastąpione symbolami "(*" i "*)", które muszą obejmować tekst komentarza.

W Pascalu - jak w każdym języku programowania - występują różnorodne wartości. Są to stałe i zmienne różnych typów. Ich deklaracje i definicje muszą być umieszczone w początkowej części bloku programu, procedury lub funkcji. W dalszej części tego rozdziału opisane są wszystkie rodzaje wartości dostępne w Kyan Pascalu oraz niektóre zależności między nimi.

Stałe

Stałe są w Pascalu oznaczeniami różnych wartości definiowanych przez programistę. Możliwe jest przy tym użycie stałych w postaci jawnej, czyli jako określonej wartości, na przykład: 4.5, -123, '100' lub napis . Stałym można też przypisać nazwy podobnie jak zmiennym. Symbolizują one wartości tych stałych w całym programie. Dzięki temu znacznie łatwiejsze jest modyfikowanie programu w przypadku, gdy konieczna jest zmiana tych wartości. Ponadto użycie nazw stałych znacznie zwiększa czytelność programu.

Nazwa stałej musi spełniać warunki określone dla nazw zmiennych. Sposób definiowania stałych jest podany w opisie słowa kluczowego "CONST".

Zmienne

Wartości zmienne, które są używane w programie, muszą być przed użyciem zadeklarowane. Deklaracja zmiennej określa jej nazwę (identyfikator) oraz zakres dozwolonych wartości. Zakres wartości nazywany jest typem zmiennej. Sposób deklarowania zmiennych jest podany w opisie słowa kluczowego "VAR".

Nazwa zmiennej może składać się z liter i cyfr. Pierwszym znakiem nazwy musi być litera. Kyan Pascal nie rozróżnia dużych i małych liter, jednak zwyczajowo przyjęte jest pisanie nazw zmiennych małymi literami. Nazwa zmiennej może mieć dowolną długość, lecz kompilator rozróżnia tylko 8 pierwszych znaków.

Każda zmienna musi mieć zadeklarowany typ, od którego zależą jej dozwolone wartości oraz dopuszczalne operacje. Typy zmiennych dzielą się na standardowe i niestandardowe. Typy standardowe mają w Kyan Pascalu określone nazwy i sposoby oznaczania wartości. Określenie wszystkich parametrów typu niestandardowego jest natomiast wykonywane przez programistę w części zawierającej definicje typów. Sposób definiowania typów niestandardowych jest podany w opisie słowa kluczowego "TYPE".

W Pascalu ściśle przestrzegana jest lokalność zmiennych. Każda zmienna jest określona tylko w tym bloku, dla którego została zdefiniowana. Zmiennymi globalnymi są więc zmienne zadeklarowane przed blokiem programu. Zmienne zdefiniowane przed blokiem procedury są lokalne i określone wewnątrz tej procedury oraz wewnątrz wszystkich procedur i funkcji przez nią wywoływanych.

Typy standardowe

Podstawowymi typami są w Pascalu typy proste, które dzielą się z kolei na typy porządkowe i typ rzeczywisty. Do typów porządkowych zalicza się natomiast typy: logiczny, całkowity i znakowy. Wszystkie te typy porządkowe są uporządkowanymi i skończonymi zbiorami wartości.

Typ rzeczywisty

Typ rzeczywisty obejmuje liczby rzeczywiste o wartościach z zakresu od -9.9999999999E+98 do -9.9999999999E-99 i od 9.9999999999E-99 do 9.9999999999E+98 oraz liczbę zero. Dokładność ich zapisu wynosi 13 cyfr dziesiętnych. Identyfikatorem typu rzeczywistego jest słowo kluczowe "REAL" (patrz opis).

Dla typu rzeczywistego dopuszczalne są operacje dodawania (+) , odejmowania (—), mnożenia (*) i dzielenia (/). Kolejność wykonywania tych operacji jest zgodna z przyjętą w matematyce. Ponadto zmiennej rzeczywistej można przypisać wartość zmiennej całkowitej.

Przy przypisywaniu zmiennej rzeczywistej wartości liczbowej w postaci jawnej należy stosować ustalony format zapisu tej wartości. Stała rzeczywista musi zawierać co najmniej jedną cyfrę przed punktem dziesiętnym i co najmniej jedną cyfrę za tym punktem. Na przykład: wartość 5 musi być zapisana jako 5.0, a wartość .1 - jako 0.1. Dozwolony jest również zapis w postaci wykładniczej, jednak z zachowaniem podanych wyżej warunków. Na przykład: liczba 5000000 może być zapisana jako 5.0E+6 lub 5.0E6, zaś liczba 0.0002 - jako 2.0E-4 (w każdym przypadku litera "E" może być mała, czyli "e").

Typ całkowity

Typ całkowity obejmuje liczby całkowite o wartościach z zakresu od —32768 do 32767. Dokładność ich zapisu wynosi 5 cyfr dziesiętnych. Identyfikatorem typu całkowitego jest słowo kluczowe "INTEGER" (patrz opis).

Dla typu całkowitego dopuszczalne są operacje dodawania (+), odejmowania (-), mnożenia (*), dzielenia (/) , dzielenia całkowitego (DIV) i reszty z dzielenia <M0D). Kolejność wykonywania tych operacji jest zgodna z przyjętą w matematyce, przy czym operatory DIV i MOD mają taki sam priorytet jak * i /. Trzeba pamiętać, że wynik operacji dzielenia (/) jest zawsze typu rzeczywistego, nawet gdy oba argumenty są całkowite i rezultat ma wartość całkowitą. Na przykład: 4 / 2 = 2.0.

Przy przypisywaniu zmiennej całkowitej wartości liczbowej w postaci jawnej nie wolno podawać liczb z punktem dziesiętnym. Dla typu całkowitego określona jest także standardowa stała MAXINT o wartości równej 32767.

Typ logiczny

Typ logiczny obejmuje dwie wartości logiczne: fałsz i prawda. Odpowiadają im dwie standardowe stałe FALSE i TRUE. Identyfikatorem typu logicznego jest słowo kluczowe "BOOLEAN" (patrz opis).

Dla typu logicznego dopuszczalne są operacje koniunkcji (AND), alternatywy (OR) i negacji (NOT). Ponadto rezultaty wszystkich operacji relacji są także typu logicznego. Kolejność wykonywania tych operacji jest następująca: NOT, AND, OR i operatory relacji (= , <, >, <>, <= i > =) .

Typ znakowy

Typ znakowy obejmuje wszystkie znaki kodu ATASCII, a więc wszystkie znaki dostępne w Atari, w tym także znaki sterujące (kontrolne). Znaki te są uporządkowane według wartości ich kodów ATASCII. Identyfikatorem typu znakowego jest słowo kluczowe "CHAR" (patrz opis).

Dla typu znakowego nie ma dopuszczalnych operacji standardowych. Istnieją jednak standardowe funkcje, których argumentem lub wynikiem może być wartość typu znakowego. W instrukcji przypisania stałe znakowe w postaci jawnej muszą być ujęte w apostrofy, na przykład: 'A', 'a', '&' lub '7'.

Typy niestandardowe

Przez programistę mogą być definiowane trzy rodzaje typów niestandardowych: proste, strukturalne i wskaźnikowe. Do typów prostych zalicza się typy wyliczeniowe i okrojone, zaś do typów strukturalnych - tablice, rekordy, zbiory i pliki. Sposób definiowania typów niestandardowych jest podany w opisie słowa kluczowego "TYPE".

Typy wyliczeniowe

Typ wyliczeniowy jest to uporządkowany i skończony zbiór wartości. Wartości te muszą być oznaczone identyfikatorami, które nie mogą się powtarzać zarówno w definicji jednego typu wyliczeniowego, jak i w definicjach różnych typów wyliczeniowych zawartych w tym samym bloku. Porządek elementów typu wyliczeniowego jest zgodny z kolejnością umieszczenia identyfikatorów w definicji.

Dla typu wyliczeniowego określone są operacje relacji oraz funkcje ORD, SUCC, PRED. Elementy typu wyliczeniowego mogą być ponadto użyte w instrukcji FOR.

Definicja typu wyliczeniowego składa sia z nazwy typu, znaku równości (=) oraz ujętej w nawiasy okrągłe listy identyfikatorów oddzielonych od siebie przecinkami. Na przykład:

```
dzień = (pon, wt, sr, czw, pt, sob, niedz);
pokój = (gabinet, sypialnia, kuchnia, łazienka);
płeć = (mężczyzna, kobieta);
```

Typy okrojone

Typ okrojony jest to typ zawierający wartości innego typu porządkowego, który jest w tym przypadku nazywany typem pierwotnym. Wartości te muszą być nie mniejsze niż ograniczenie dolne i nie większe niż ograniczenie górne.

Zmienna typu okrojonego ma wszystkie własności zmiennej typu pierwotnego, poza dozwolonym zakresem wartości. W szczególności wszystkie operacje i funkcje dopuszczalne dla typu pierwotnego są również dopuszczalne dla typu okrojonego. Rezultat takiej operacji lub funkcji może jednak mieć wartość nie należącą do zakresu określonego przez typ okrojony.

Definicja typu okrojonego składa się z nazwy typu, znaku równości <=) oraz dwóch stałych oddzielonych od siebie dwiema kropkami. Stałe te muszą być typu pierwotnego i pierwsza z nich musi być mniejsza lub równa drugiej. Na przykład:

```
duzelitery = 'A' .. 'Z';
masa = 0 .. 1000;
dzień = pon .. niedz;
alfa = 3 .. 3
```

Tablice

Typ tablicowy składa się z elementów innego typu, który jest zwany typem składowym. Wielkość tablicy jest określana przez wartości indeksowe, które muszą należeć do dowolnych typów porządkowych. Typem składowym może być także inna tablica lub inny typ strukturalny. Jeśli typem składowym jest typ znakowy, to otrzymujemy tablicą znakową, czyli napisową. Odpowiada to zmiennej lub tablicy tekstowej w innych językach programowania.

Zmienna będące elementami tablic mają wszystkie własności zmiennych typu składowego. W szczególności wszystkie operacje i funkcje dopuszczalne dla typu składowego są również dopuszczalne dla elementów tablic.

Definicja tablicy składa się z nazwy tablicy, znaku równości (=), słowa "ARRAY", określonych przez typy porządkowe wymiarów tablicy oddzielonych od siebie przecinkami i zamkniętych w nawiasy kwadratowe, słowa "OF" oraz oznaczenia typu składowego. Na przykład:

```
tekst = ARRAY [1 .. n] OF CHAR;
tensor = ARRAY [1 .. 2, 1 .. 2] OF REAL;
tensor = ARRAY [1 .. 2] OF ARRAY [1 .. 2] OF REAL;
alfa = ARRAY [BOOLEAN, 1 .. 10, litera] OF CHAR;
xx = ARRAY [-3 . . 3] OF RECORD a: REAL;
                                b: INTEGER;
                                c: CHAR
                          END
```

W celu wskazania elementu tablicy należy podać nazwę tej tablicy (ale nie nazwę typu tablicowego). Po niej trzeba umieścić wartości indeksów oddzielone od siebie przecinkami i

ujęte w nawias kwadratowy. Pokazuje to zamieszczony poniżej przykładowy fragment programu.

```
TYPE macierz = ARRAY [1..5,1..5] OF REAL;
VAR x,y: macierz;
BEGIN
  READLN x[1, 1];
  READLN y[1,1]
END
```

Rekordy

Typ rekordowy składa się z ustalonej liczby elementów różnych typów, które są zwane polami. Dla każdego pola musi być zdefiniowany jego typ oraz identyfikator, który służy do wyboru tego pola. Opis typu rekordowego może zawierać część zmienną, grupującą kilka wariantów i umieszczoną za częścią stałą. Wariant części zmiennej również może posiadać część zmienną. W takich przypadkach zmienne tego samego typu rekordowego mogą mieć różne struktury.

Definicja rekordu składa się z nazwy rekordu, znaku równości (=) , słowa "RECORD", listy pól i słowa "END". Lista pól musi zawierać część stałą i może zawierać część zmienną. Część stała jest złożona z rozdzielonych średnikami sekcji, które są utworzone przez listy identyfikatorów i oznaczenie typu. Część zmienna składa się. ze słowa "CASE", selektora wybierającego wariant, słowa "OF" i listy wariantów rozdzielonych średnikami. Selektorem części zmiennej jest typ, zwany tu typem znacznikowym, który ponadto może być poprzedzony identyfikatorem i dwukropkiem. Każdy wariant zawiera listę stałych typu znacznikowego, dwukropek oraz umieszczoną w nawiasie okrągłym listę pól. Na przykład:

```
zespolone = RECORD re, im: REAL END;
data = RECORD dziani 1..31;
       miesiąc: 1..12; rok:
       0..maxint END;
osoba - RECORD imię, nazw: tekst; urodz:
        data; CASE pl: plec OF kob:
        (nazwpan: tekst); mezcz: (wojsko:
        BOOLEAN) END;
```

Dostęp do pola zmiennej rekordowej jest możliwy przy pomocy instrukcji "WITH" lub przez podanie nazwy zmiennej i nazwy pola oddzielonych od siebie kropką. Na przykład, jeśli zmienna "z" jest typu rekordowego "zespolone" (patrz wyżej), to jej elementy oznaczane są jako "z.re" i "z.im".

Zbiory Typ zbiorowy jest zbiorem wszystkich podzbiorów typu

podstawowego, który musi być typem porządkowym. Maksymalna liczba elementów zbioru, czyli wartości typu podstawowego, wynosi 256. Wartościami typu zbiorowego są wszystkie możliwe kombinacje jego elementów, w tym również zbiór pusty oznaczany jako [].

Na wszystkich typach zbiorowych określone są operacje sumy (+), różnicy (-), części wspólnej (*) , równości (=), nierówności (<>) i zawierania się zbiorów << i >) oraz przynależności do zbioru (IN). Operacje +, — i * dają w wyniku zbiór, a rezultatem pozostałych operacji jest wartość logiczna.

Definicja zbioru składa się z nazwy zbioru, znaku równości (=), słów "SET OF" oraz identyfikatora typu porządkowego. Na przykład:

```
dyzur = SET OF dzien; swieto =
SET OF pt .. niedz; liczba = SET
OF 1 .. 10;
```

Wartość typu zbiorowego podaje się jako listą elementów zbioru ujętą w nawiasy kwadratowe. Na przykład, zmiennej "parzyste" typu zbiorowego "liczba" (patrz wyżej) można przypisać wartość w następujący sposób:

```
parzyste := [2, 4, 6, 8, 10];
```

Pliki

Typ plikowy składa się z ciągu elementów tego samego typu, zwanego typem składowym. Typ składowy nie może być typem plikowym, ani typem strukturalnym zawierającym elementy plikowe. Liczba elementów pliku nie jest określona, a najmniejszym możliwym plikiem jest plik pusty o długości zero. W Pascalu zdefiniowane są trzy pliki standardowe INPUT, OUTPUT i TEXT.

Normalnie zapisywanie i odczytywanie elementów pliku odbywa się kolejno od początku pliku. Ponadto plik może być albo zapisywany, albo odczytywany." Jednoczesny zapis i odczyt jest więc niedozwolony. W Kyan Pascalu, w odróżnieniu od standardu, możliwe są także pliki o dostępie swobodnym, czyli pozwalające na równoczesny zapis i odczyt oraz na wykonywanie tych operacji dla dowolnego elementu. Zabronione jest jednak dopisywanie na końcu istniejącego piiku. Pliki o dostępie swobodnym nie mogą być plikami tekstowymi, ani złożonymi z elementów typu logicznego.

Na typach plikowych określone są operacje GET, PUT, RESET, REWRITE i SEEK, przy czym operacja SEEK dotyczy tylko plików o dostępie swobodnym. Działanie tych operacji jest opisane w słowniku.

Definicja pliku składa się z nazwy pliku, znaku równości (=), słów "FILE OF" oraz identyfikatora typu składowego. Na przykład:

```
kartoteka = FILE OF osoba;
zesp = FILE OF RECORD re, im: REAL END;
dane = FILE OF INTEGER;
kalendarz = FILE OF data;
```

Typy wskaźnikowe

Wszystkie opisane poprzednio typy istnieją przez cały czas realizacji tej części programu, w której zostały zadeklarowane. Zmienne tych typów mają przydzielone stałe obszary pamięci, a identyfikatory służą do odwoływania się do tych obszarów. Takie zmienne nazywamy zmiennymi statycznymi. Istnieją jednak zagadnienia, w których struktury zmiennych zmieniają się w czasie realizacji i nie można z góry określić liczby i rodzaju elementów. Takie zmienne nazywamy zmiennymi dynamicznymi. Odwołanie do zmiennej dynamicznej nie jest realizowane przy pomocy identyfikatora, lecz poprzez wskaźnik, który wskazuje tą zmienną.

Typ wskaźnikowy jest zbiorem wskaźników identyfikujących zmienne typu wskazywanego. Początkowo zbiór ten zawiera tylko wskaźnik pusty "NIL", który nie wskazuje żadnej zmiennej. Tworzenie nowych wskaźników realizuje procedura NEW, a ich kasowanie procedura DISPOSE.

Definicja typu wskaźnikowego składa się z identyfikatora typu, znaku równości (=) , znaku potęgowania "^" oraz identyfikatora typu wskazywanego. Na przykład:

```
wsklisty = ^lista;
postac = ^osoba;
notes = ^kalendarz;
```

Zmienna typu wskaźnikowego jest więc w pewnym stopniu adresem innej zmiennej. Obiekt wskazywany przez tą zmienną oznaczany jest identyfikatorem zmiennej wskaźnikowej z dodanym na końcu znakiem "^". Na przykład: "alfa^" jest obiektem wskazywanym przez zmienną wskaźnikową "alfa". Dzięki takiemu rozróżnieniu można kopiować wkaźniki tego samego typu wskaźnikowego instrukcją przypisania:

```
alfa := beta albo
```

kopiować wskazywane obiekty poprzez:

```
alfa^ := beta^
```

W pierwszym przypadku po operacji wskazywane obiekty zachowują swoje wartości, lecz oba wskaźniki wskazują ten sam obiekt. Natomiast w drugim przypadku wskaźniki wskazują różne obiekty, ale obiekty te są identyczne.

Typ wskaźnikowy jest jedną z najważniejszych struktur Pascala. Służy on przede wszystkim do wykonywania operacji na listach. Dzięki temu można tworzyć skomplikowane struktury danych i dowolnie je przetwarzać. Sposób wykorzystania typu wskaźnikowego prześledzimy na przykładzie:

```
TYPE adres = ^dane;
     dane = RECORD dalej: adres;
       znak: CHAR END; VAR start,
licznik, pomoc: adres;
```

Teraz przez wywołanie procedury NEW(licznik) tworzy się

nieokreśloną zmienną typu "dane" oraz wskaźnik typu "adres", który jest wartością zmiennej "licznik". Aby nadać wartości elementom zmiennej "dane" trzeba wykonać, trzy instrukcje:

```
start := NIL;
licznik^.dalej := start;
licznik^.znak := 'A';
```

Efektem jest jednoelementowa lista A.NIL wskazywana przez zmienną "licznik". Dołączenie nowego elementu do listy wymaga zwolnienia tej zmiennej, gdyż ponowne wykonanie NEW(licznik) spowoduje utratę istniejącego elementu. W tym celu wartość zmiennej "licznik" należy przypisać zmiennej "start", a następnie można utworzyć nowy element:

```
start := licznik;
NEW(licznik) ;
licznik^.dalej := NIL;
licznik^.znak := 'C';
```

Rezultatem tej operacji są dwie listy jednoelementowe A.NIL i C.NIL, które są wskazywane odpowiednio przez zmienne "start" i "licznik". W celu ich połączenia, czyli uzyskania listy A.C.NIL, należy zmienić wartość zmiennej "start^.dalej", po czym można wykorzystać wskaźnik "licznik" do utworzenia nowego elementu:

```
start^.dalej := licznik;
```

Podobnie przebiega dołączenie elementu na początku listy. Nieco trudniejsze jest wstawienie elementu do środka istniejącej listy, gdyż trzeba najpierw odszukać właściwe miejsce. Można w tym celu wykorzystać zmienną "pomoc" i instrukcją "WHILE". Na przykład, z listy A.C.NIL utworzymy listą A.B.C.NIL:

```
pomoc := start; WHILE
pomoc^.znak <> 'A' DO pomoc :=
pomoc^.dalej;
```

Przeglądanie listy kończy się, gdy zmienna "pomoc" wskazuje wybrany element. Teraz tworzy sie. nowy element i ustawia jego wskaźnik na element, który będzie po nim następował. Na końcu należy wskazać nowy element w elemencie poprzedzającym go.

```
NEW(licznik);
licznik^.znak := 'B';
licznik^.dalej := pomoc^.dalej;
pomoc^.dalej := licznik;
```

Analogicznie realizowane jest dołączenie nowego elementu przed wybranym elementem listy. Bardzo podobnie wykonuje się usuniecie elementu listy, przy czym konieczne jest przepisanie wskaźnika dla zachowania ciągłości listy.

Wyrażenia

Wyrażenie jest konstrukcją Pascala oznaczającą wartość pewnego typu. Określenie tej wartości odbywa się przez wykonanie operacji wskazanych przez znajdujące się w wyrażeniu operatory. Wyrażenia są zbudowane ze stałych, zmiennych, operatorów, nazw funkcji i nawiasów okrągłych.

Przy konstruowaniu wyrażeń w Pascalu należy zwracać baczną uwagą na typy użytych, argumentów oraz typ uzyskanego wyniku. Zastosowanie typu, dla którego operator jest niedozwolony, spowoduje błąd. Błąd wywoła również próba przypisania wartości wyrażenia zmiennej, która ma inny typ niż wynik tego wyrażenia.

Poniżej zestawione są wszystkie operatory występujące w Kyan Pascalu oraz dozwolone dla nich typy argumentów i typ wyniku.

Operatory arytmetyczne

operator	operacja	typ argumentu	typ wyniku
+	znak	integer lub real	jak argument
-	znak	integer lub real	jak argument
+	dodawanie	integer lub real	| integer, gdy takie
-	odejmowanie	integer lub real	| są oba argumenty,
*	mnożenie	integer lub real	| inaczej real
div	dziel. całk.	integer	integer
/	dzielenie	integer lub real	real
mod	modulo	integer	integer

Operatory logiczne

operator	operacja	typ argumentu	typ wyniku
not	negacja	boolean	boolean
or	alternatywa	boolean	boolean
and	koniunkcja	boolean	boolean

Operatory relacji

operator	operacja	typ argumentu*	typ wyniku
=	równość	P, Z, W lub N	boolean
<>	nierówność	P, Z, W lub N	boolean
<	mniejszość	P lub N	boolean
>	większość	P lub N	boolean
<=	nie większość	P lub N	boolean
>-	nie mniejszość	P lub N	boolean
< lub >	zawieranie	Z	boolean
in	przynależność	Po i Z	boolean

* P=prosty, Z=zbiorowy, W=wskaźnikowy, N=napisowy, Po=porządkowy

Operatory zbiorowe

operator	operacja	typ argumentu	typ wyniku
+	suma	zbiorowy	zbiorowy
-	różnica	zbiorowy	zbiorowy
*	część wspólna	zbiorowy	zbiorowy

UWAGA: typy obu argumentów muszą być zgodne.

Priorytet operatorów, czyli kolejność wykonywania określonych przez nie operacji, jest dla operatorów arytmetycznych taki sam, jak przyjęty w matematyce. Operatory o jednakowym priorytecie są realizowane od lewej do prawej. Priorytet operatorów logicznych jest zgodny z kolejnością umieszczenia ich w powyższej tabeli. Wszystkie operatory relacji mają jednakowy priorytet.

W Pascalu operator równości (=) NIE jest elementem instrukcji przypisania. W odróżnieniu od innych języków programowania instrukcja przypisania jest oznaczana znakiem złożonym z dwukropka i znaku równości (:=). Na przykład: suma := a + b.

1.5. Słownik

Ten rozdział zawiera kompletny słownik słów kluczowych języka Kyan Pascal w kolejności alfabetycznej. Podana jest tu składnia każdego słowa, sposoby jego wykorzystania oraz liczne przykłady. Ponadto na początku zamieszczony jest spis tych słów.

W słowniku przyjęta została następująca kolejność opisus nazwa, typ, składnia, przykłady i działanie. Typ określa rodzaj słowa kluczowego: instrukcja, funkcja, procedura, dyrektywa, operator, typ lub stała. Składnia opisuje przy użyciu symboli podanych we wprowadzeniu dozwolone sposoby zapisu słowa kluczowego. Pozostałe punkty nie wymagają objaśnienia.

SŁOWA KLUCZOWE KYAN PASCAL

ABS	EOLN	ODD	SP
AND	EQU	OF	SQR
ARCTAN	EXP	OR	SQRT
ARRAY	FALSE	ORD	SUCC
ASSIGN	FILE	ORG	T
BEGIN	FOR	OUTPUT	TEXT
BOOLEAN	FORWARD	PACKED	THEN
CASE	FUNCTION	PRED	TO
CHAIN	GET	PROCEDURE	TRUE
CHAR	GOTO	PROGRAM	TRUNC
CHR	IF	PUT	TYPE
CONST	IN	READ	UNTIL
COS	INPUT	READLN	VAR
DB	INTEGER	REAL	WHILE
DISPOSE	LABEL	RECORD	WITH
DIV	LN	REPEAT	WRITE
DO	LOCAL	RESET	WRITELN
DOWNTO	MAX INT	REWRITE	#A
DW	MOD	ROUND	#I
ELSE	NEW	SEEK	(* *)
END	NIL	SET	: =
EOF	NOT	SIN	

ABS

TYP: funkcja
FORMAT: ABS(<wyraż_liczb>)
PRZYKŁADY:

```
x := ABS(y);
WRITELN(ABS(i*x/2)) ;
IF x = ABS(x) THEN WRITE('liczba dodatnia')
```

DZIAŁANIE: Zwraca wartość bezwzględną argumentu, który musi być wartością liczbową typu REAL lub INTEGER. Dla liczb dodatnich rezultatem jest wartość argumentu, dla zera - zero, a dla liczb ujemnych - wartość argumentu pomnożona przez -1. Wynikiem może być dowolna liczba dodatnia, której typ jest taki sam jak typ argumentu.

AND

TYP: operator logiczny
FORMAT: <wyraż_log> AND <wyraż_iog>
PRZYKŁADY:

```
x := a AND b;
WRITELN((a <> b) AND (x/2=Y));
IF (a>0) AND (x<=0) THEN WRITE('prawda')
```

DZIAŁANIE: Wykonuje operacje. koniunkcji (iloczynu logicznego) dwóch argumentów typu BOOLEAN. Gdy oba argumenty mają wartość TRUE (są prawdziwe), to wynikiem jest także TRUE, a w przeciwnym przypadku FALSE.

ARCTAN

TYP: funkcja
FORMAT: ARCTAN(<wyraż_liczb>)
PRZYKŁADY:

```
WRITELN(ARCTAN(1));
x := ARCTAN(y);
IF ARCTAN (z) = 0.0 THEN WRITELN('KAT = 0')
```

DZIAŁANIE: Zwraca wartość funkcji arcus tangens podanego argumentu liczbowego. Argument może być typu INTEGER lub REAL. Wynikiem jest liczba typu REAL z przedziału od -1.5708 do +1.5708.

ARRAY

TYP: deklaracja typu
FORMAT: <nazwa>[,<nazwa>] = [PACKED] ARRAY
[<typ_ind>[,<typ_ind>...]] OF <typ>
lub <nazwa>[,<nazwa>]: [PACKED] ARRAY
[<typ_ind>[,<typ_ind>...]] OF <typ>
PRZYKŁADY:

```
TYPE
tekst = PACKED ARRAY [1..n] OF CHAR;
tensor = ARRAY [1..2, 1..2] OF REAL;
tensor = ARRAY [1..2] OF ARRAY [1..2] OF REAL;
```

```
alfa = ARRAY [BOOLEAN, I..10, litera] OF CHAR;
xx = ARRAY [-3..3] OF RECORD a: REAL;
                             b: INTEGER;
                             c: CHAR
                          END;
opis = RECORD k: ARRAY [BOOLEAN] OF CHAR;
              l: ARRAY [1..100] OF INTEGER;
       END;
VAR
 tensor: ARRAY [1..2,1..2] OF REAL; alfa, beta,
 gamma: ARRAY [litera] OF BOOLEAN;
```

DZIAŁANIE: Definiuje podaną nazwę jako nazwą typu tablicowego (pierwszy format) lub nazwę zmiennej typu tablicowego <drugi format). Nazwa może być dowolnym, poprawnym identyfikatorem, a typ indeksowy musi być typem porządkowym. Słowo "ARRAY" może być użyte w części definiującej typy (po słowie "TYPE") w bloku programu, procedury lub funkcji (tylko według pierwszego formatu) albo w części definiującej zmienne (po sławie "VAR" - tylko według drugiego formatu). Ponadto słowo to może być wykorzystywane w definicjach innych zmiennych i innych typów.

ASSIGN

TYP: procedura
FORMAT: ASSIGN(<zm_wskaź>, <wyr_liczb>)
PRZYKŁADY:

```
ASSIGN(adres,miejsce);
ASSIGN(gdzie,100);
```

DZIAŁANIE: Umieszcza w miejscu pamięci określonym przez <wyr_liczb> wartość zmiennej, którą wskazuje <zm_wskaź>. Pierwszym parametrem procedury musi być zmienna wskaźnikowa, a drugim wartość typu INTEGER. Ponieważ największą wartością liczby INTEGER jest 32767, to miejsca o większych adresach są określane przez liczby ujemne otrzymane po odjęciu od adresu wartości 65536. Jeśli wskazywana zmienna zajmuje więcej niż jeden bajt, to podany adres jest adresem początkowym kolejnych miejsc pamięci, w których umieszczane są bajty wartości tej zmiennej.

BEGIN

TYP: instrukcja
FORMAT: BEGIN
PRZYKŁAD:

```
BEGIN
```

DZIAŁANIE: Instrukcja "BEGIN" rozpoczyna blok strukturalny będący częścią programu. Może to być właściwy blok programu, blok procedury, blok funkcji lub blok instrukcji złożonej. Każdy taki blok musi kończyć się instrukcją "END". Pomiędzy słowami "BEGIN" i "END" może znajdować się dowolna liczba instrukcji, w tym także inne bloki BEGIN/END. Możliwe jest również zbudowanie bloku pustego, w którym pomiędzy słowami "BEGIN" i "END" nie ma żadnej instrukcji.

BOOLEAN

TYP: deklaracja typu
FORMAT: <nazwa>[, <nazwa>...] = BOOLEAN;
lub <nazwa>[,<nazwa>...]: BOOLEAN;
PRZYKŁADY:

```
TYPE
stan = BOOLEAN; x1, x2, x3 = BOOLEAN;
opis = ARRAY [BOOLEAN] OF CHAR;
VAR
  stan, blad, koniec: BOOLEAN;
  alfa, beta, gamma: ARRAY [litera] OF BOOLEAN;
```

DZIAŁANIE: Definiuje podaną nazwę jako nazwą typu logicznego (pierwszy format) lub nazwą zmiennej typu logicznego (drugi format). Nazwa może być dowolnym, poprawnym identyfikatorem. Słowo "BOOLEAN" może być użyte w części definiującej typy (po słowie "TYPE" - tylko według pierwszego formatu) albo w części definiującej zmienne (po słowie "VAR" — tylko według drugiego formatu). Ponadto słowo to może być wykorzystywane w definicjach innych zmiennych i innych typów.

CASE

TYP: instrukcja
FORMAT: CASE <wyraż> OF <stała>[,<stała>...]:<instrukcja>;
[;<stała>[,<stała>...]:<instrukcja>...] END
PRZYKŁADY:

```
CASE miesiac OF
  1,3,5,7,8,10,12: dni := 31;
  4,6,9,11: dni := 30;
  2: IF przst(rok) THEN dni := 29 ELSE dni := 28
END
 CASE dzien OF
pon,wt,sr,czw,pt: WRITELN('roboczy');
śob,niedz: WRITELN('swiateczny') END
```

DZIAŁANIE: Wykonuje instrukcję wybraną przez aktualną wartość wyrażenia bądącego warunkiem. Dozwolone są dowolne instrukcje, w tym również puste, złożone oraz inne instrukcje CASE. Wartość wyrażenia warunkowego musi być typu porządkowego, a wszystkie możliwe wartości tego wyrażenia muszą być ujęte w listach stałych. Wystąpienie wartości wyrażenia, której nie ma w żadnej liście stałych spowoduje błąd. Niedozwolone jest także dwukrotne użycie stałej w jednej instrukcji CASE.

Słowo "CASE" służy ponadto do wyboru wariantu części zmiennej w deklaracji zmiennej rekordowej (patrz opis słowa "RECORD"). Jego działanie jest w tym przypadku bardzo zbliżone do działania normalnej instrukcji CASE.

CHAIN

TYP: procedura
FORMAT: CHAIN(<spec_pliku>)

PRZYKŁADY:

```
CHAIN( 'D:WARIANT.I')
CHAIN(nazwa)
```

DZIAŁANIE: Odczytuje z urządzenia zewnętrznego wskazany plik i uruchamia znajdujący się w nim skompilowany program. Umożliwia to łączenie programów w łańcuchy i pozwala na ominięcie ograniczenia pojemności pamięci dostępnej dla pojedynczego programu. Wartości wykorzystywanych zmiennych są przekazywane z programu wywołującego do wywoływanego według kolejności umieszczenia w programie, aż do wystąpienia niezgodności typów tych zmiennych. Jeśli dalej znajdują się jeszcze jakieś zmienne o zgodnych typach, to ich wartości nie zostaną już przekazane.

CHAR

TYP: deklaracja typu
FORMAT: <nazwa>[,<nazwa>...] = CHAR;
lub <nazwa>[,<nazwa>...]: CHAR;
PRZYKŁADY:

```
TYPE
znak = CHAR; x1, x2, x3 = CHAR;
opis = ARRAY [BOOLEAN] OF CHAR;
VAR
  znak, litera, code: CHAR; tekst:
  PACKED ARRAY [1..n] OF CHAR;
```

DZIAŁANIE: Definiuje podaną nazwę jako nazwę typu znakowego (pierwszy format) lub nazwę zmiennej typu znakowego (drugi format). Nazwa może być dowolnym, poprawnym identyfikatorem. Słowo "CHAR" może być użyte w części definiującej typy (po słowie "TYPE" — tylko według pierwszego formatu) albo w części definiującej zmienne (po słowie "VAR" - tylko według drugiego formatu). Ponadto słowo to może być wykorzystywane w definicjach innych zmiennych i innych typów.

CHR

TYP: funkcja
FORMAT: CHR(<wyraż_liczb>)
PRZYKŁADY:

```
WRITELN(CHR(65));
a := CHR(x)
x[4,4] := CHR(34)
IF CHR(z) = 'A' THEN WRITE('litera A')
x := ORD('A'); WRITE (CHR (x));
```

DZIAŁANIE: Zwraca znak typu CHAR, którego liczbą porządkową w typie jest wartość <wyraż_liczb>. Jako argument są dozwolone liczby typu INTEGER z zakresu określonego przez liczność typu wyniku (zwykle od 0 do 255). Funkcja ta jest odwrotnością funkcji ORD dla wartości typu CHAR (patrz piąty przykład).

CONST

TYP: identyfikator bloku
FORMAT: CONST <lista_defin_stałych>
PRZYKŁAD:

```
CONST
  pi = 3.1415;
  nie   =   FALSE;
  napis = ATARI ;
```

DZIAŁANIE: Słowo "CONST" rozpoczyna część definiującą stałe w bloku programu, procedury, funkcji lub instrukcji złożonej. Następują po nim oddzielone średnikami nazwy stałych i przypisane im wartości w postaci jawnej (użycie wyrażenia w definicji stałej jest niedozwolone).

COS

TYP: funkcja
FORMAT: COS(<wyraż_liczb>)
PRZYKŁADY:

```
WRITELN(C0S(45));
x   :=  COS(y);
IF COS(z) = 1.0 THEN WRITE('kat = 0')
```

DZIAŁANIE: Zwraca wartość funkcji cosinus podanego argumentu, który określa wielkość kąta w radianach. Wielkość argumentu jest dowolna, a wynik jest zawsze z zakresu od —1 do +1. Argument może być typu REAL lub INTEGER, a wynik jest zawsze typu REAL.

DB

TYP: dyrektywa asemblera
FORMAT: DB <wyraż_liczb>
PRZYKŁADY:

```
DB >516
DB <516
DB $6A
```

DZIAŁANIE: Zamienia podczas kompilacji podane wyrażenie liczbowe na jeden bajt danych kodu wynikowego. Wyrażenie może być liczbą szesnastkową. Słowo "DB" nie może być umieszczone w pierwszej kolumnie ekranu. Znaki ">" i "<" oznaczają odpowiednio młodszy i starszy bajt słowa dwubajtowego. Dyrektywa DB może się znajdować tylko wewnątrz procedury w języku maszynowym rozpoczynającej się od dyrektywy #A.

DISPOSE

TYP: procedura
FORMAT: DISPOSE(<zm_wskaźn>)
PRZYKŁAD:

```
DISPOSE(p)
```

DZIAŁANIE: Powoduje zniszczenie zmiennej wskazywanej przez wskaźnik będący parametrem procedury. W ten sposób wskazywana

zmienna przestaje je istnieć, a zajmowane przez nią miejsce w pamięci może być wykorzystane dla innych danych.

DIV

TYP: operator arytmetyczny
FORMAT: <wyraż_liczb> DIV <wyraż_liczb>
PRZYKŁADY:

```
x := 20 DIV 3; WRITELN(x DIV y);
IF a DIV 2 > 0 THEN oblicz(a)
```

DZIALANIE: Operator DIV realizuje dzielenie całkowite, czyli takie, którego wynikiem jest liczba całkowita, a reszta z dzielenia jest odrzucana. Zarówno argumentami, jak wynikiem mogą być tylko liczby typu INTEGER. Odpowiednikiem operacji a DIV b (jednak z innymi dozwolonymi typami argumentów) jest funkcja:

```
TRUNC(a/b)
```

DO

TYP: instrukcja
FORMAT: patrz opis słów FOR, WHILE i WITH.
DZIAŁANIE: Słowo "DO" stanowi integralną część składową instrukcji FOR, WHILE i WITH. Następuje po nim instrukcja, której wykonanie warunkuje pierwsze słowo kluczowe instrukcji. Po słowie "DO" dozwolone są dowolne instrukcje, w tym również instrukcje puste i złożone. Patrz też opis instrukcji FOR, WHILE i WITH.

DOWNTO

TYP: instrukcja
FORMAT: patrz opis instrukcji FOR
DZIAŁANIE: Słowo "DOWNTO" stanowi integralną cześć składową instrukcji FOR nakazującą zliczanie pętli w kierunku mniejszych wartości. Następujący po nim parametr określa dolną wartość graniczną pętli, czyli minimalną wartość licznika. Wartość graniczna nie może być zmieniana wewnątrz pętli. Patrz też opis instrukcji FOR.

DW

TYP: dyrektywa asemblera
FORMAT: DW <wyraż_liczb>
PRZYKŁADY:

```
DW 516
DW $02E7
```

DZIAŁANIE: Zamienia podczas kompilacji podane wyrażenie liczbowe na dwubajtowe słowo danych kodu wynikowego. Wyrażenie może być liczbą szesnastkową. Słowo "DW" nie może być umieszczone w pierwszej kolumnie ekranu. Dyrektywa DW może się

znajdować tylko wewnątrz procedury w języku maszynowym rozpoczynającej się od dyrektywy #A.

ELSE

TYP: instrukcja
FORMAT: patrz opis instrukcji IF
DZIAŁANIE: Słowo ELSE stanowi integralną część instrukcji IF. Następuje po nim instrukcja, wykonywana gdy warunek po IF jest fałszywy. Może to być także instrukcja złożona. Część instrukcji IF rozpoczynająca się od słowa "ELSE" może zostać pominięta (patrz też opis IF).

END

TYP: instrukcja
FORMAT: END
PRZYKŁAD:

```
END
```

DZIAŁANIE: Instrukcja "END" kończy blok strukturalny będący częścią programu. Może to być właściwy blok programu, blok procedury, blok funkcji lub blok instrukcji złożonej. Każdy taki blok musi rozpoczynać się instrukcją "BEGIN". Pomiędzy słowami "BEGIN" i "END" może znajdować się dowolna liczba instrukcji, w tym także inne bloki BEGIN/END. Możliwe jest również zbudowanie bloku pustego, w którym pomiędzy słowami "BEGIN" i "END" nie ma żadnej instrukcji.

Ponadto słowo "END" oznacza koniec instrukcji wyboru CASE oraz koniec definicji typu rekordowego lub zmiennej tego typu (patrz opis słów "CASE" i "RECORD").

EOF

TYP: funkcja
FORMAT: EOF[(<nazwa_pliku>)]
PRZYKŁADY:

```
WRITELN(EOF(fs));
IF NOT EOF(plik) THEN READ(plik,i)
IF NOT EOF THEN READ(i)
```

DZIAŁANIE: Zwraca wartość typu BOOLEAN, która sygnalizuje osiągnięcie końca pliku wskazanego przez argument będący zmienną typu plikowego. Gdy został już osiągnięty koniec pliku, to wynikiem jest wartość TRUE, a w przeciwnym przypadku FALSE. Jeśli argument funkcji został pominięty, to przyjmowana jest wartość INPUT.

EOLN

TYP: funkcja
FORMAT: EOLN[(<nazwa_pliku>)]
PRZYKŁADY:

```
WRITELN(EOLN(plik));
```

```
IF EOLN(INPUT) THEN READLN(a)
IF EOLN THEN READLN(a)
WHILE NOT EOLN(fs) DO BET(fs);
```

DZIAŁANIE: Zwraca wartość typu BOOLEAN, która sygnalizuje rodzaj aktualnie dostępnego elementu w pliku wskazanym przez argument będący zmienną typu plikowego. Gdy znakiem tym jest znak końca wiersza (RETURN - End Of Line), to wynikiem jest wartość TRUE, a w przeciwnym przypadku FALSE. Jeśli argument funkcji został pominięty, to przyjmowana jest wartość INPUT.

EQU

TYP: dyrektywa asemblera
FORMAT: <etykieta> EQU <wyraż_liczb>
PRZYKŁADY:

```
CONSOL EQU 53279
STOS EQU LOCAL
```

DZIAŁANIE: Przyporządkowuje etykiecie wartość dwubajtowej stałej. Dyrektywa EQU może być stosowana tylko w procedurach w języku maszynowym dołączanych dyrektywą #A.

EXP

TYP: funkcja
FORMAT: EXP(<wyraź_liczb>)
PRZYKŁADY:

```
WRITELN(EXP(y));
x := EXP(2*z+1);
a := EXP(5); WRITE(LN(a));
```

DZIAŁANIE: Zwraca liczbę będącą wynikiem podniesienia liczby e (2.71828) do potęgi określonej przez argument funkcji. Inaczej mówiąc:

```
EXP(a) = 2.71828
```

Argument funkcji EXP może być liczbą typu REAL lub INTEGER, a wynik jest zawsze liczbą typu REAL większą od zera.

FALSE

TYP: stała
FORMAT: FALSE
PRZYKŁADY:

```
WRITELN(FALSE);
stan := FALSE;
```

DZIAŁANIE: Stała typu BOOLEAN oznaczająca wartość logiczną "fałsz".

FILE

TYP: deklaracja typu
FORMAT: <nazwa>[,<nazwa>] = FILE OF <typ>
lub <nazwa>[,<nazwa>]: FILE OF <typ>

PRZYKŁADY:

```
TYPE
  kartoteka = FILE OF osoba;
  zesp = FILE OF RECORD re, im: REAL END;
  dane = FILE OF INTEGER;
  kalendarz = FILE OF data;
VAR
  kalendarz, notes: FILE OF data;
  wyniki: FILE OF REAL;
```

DZIAŁANIE: Definiuje podaną nazwą jako nazwą typu plikowego (pierwszy format) lub nazwę zmiennej typu plikowego (drugi format). Nazwa może być dowolnym, poprawnym identyfikatorem, a typ nie może być typem plikowym, ani zawierającym elementy typu plikowego. Słowo "FILE" może być użyte w części definiującej typy (po słowie "TYPE" - tylko według pierwszego formatu) albo w części definiującej zmienne (po słowie "VAR" — tylko według drugiego formatu).

FOR

TYP: instrukcja
FORMAT: FOR <zmienna>:=<wart_pocz> TO|DOWNTO <wart_konc> DO <instrukcja>
PRZYKŁADY:

```
FOR i := 1 TO 10 DO silnia(i)
FOR i := 10 DOWNTO 1 DO silnia(i)
FOR k :=  'A' TO znak DO BEGIN END
FOR dzien := pon TO pt DO WRITELN('roboczy')
FOR 1 := 2*x TO 8*x DO oblicz (x)
```

DZIAŁANIE: Tworzy pętlę programu, w której instrukcja znajdująca sią po słowie "DO" wykonywana jest kilka razy. Liczba przejść pętli oraz wartości licznika są ustalane przez parametry instrukcji FOR. Kolejno oznaczają one:
<zmienna> - nazwa zmiennej typu porządkowego wykorzystywanej jako licznik pętli (tzw. zmienna sterująca);
<wart_pocz> - początkowa wartość licznika pętli, przypisywana zmiennej sterującej przed rozpoczęciem pętli;
<wart_konc> - wartość graniczna licznika - pątla jest wykonywana tak długo, dopóki wartość zmiennej sterującej nie przekracza wartości granicznej;
<instrukcja> - instrukcja wykonywana w pętli - może to być dowolna instrukcja, w tym również instrukcja pusta lub złożona;
Licznik pętli jest po każdorazowym wykonaniu instrukcji zwiększany (gdy użyto słowa "TO") lub zmniejszany (gdy użyto "DOWNTO") do sąsiedniej wartości w typie porządkowym, do którego należy zmienna sterująca.

FORWARD

TYP: dyrektywa kompilatora
FORMAT: <nagłówek_procedury>|<nagłów«k_funkcji> FORWARD
PRZYKŁADY:

```
FUNCTION waga (z: INTEGER): REAL; FORWARD;
PROCEDURE zmiana (x,y,z: CHAR); FORWARD;
```

DZIAŁANIE: Każdy obiekt stosowany w Pascalu musi być zdefiniowany lub zadeklarowany przed użyciem. Czasami występują jednak sytuacje, w których dwie procedury lub funkcje wywołują się wzajemnie. Niemożliwe jest wtedy zdefiniowanie jednej z nich przed zdefiniowaniem drugiej. Rozwiązaniem tego problemu jest możliwość odwołania w przód. Polega to na zadeklarowaniu procedury lub funkcji tylko w postaci jej nagłówka, po którym następuje słowo "FORWARD". Tak określona struktura może być wykorzystywana w programie przed jej właściwą definicją. W nagłówku właściwej definicji należy natomiast pominąć wszystkie parametry oprócz nazwy procedury lub funkcji.

FUNCTION

TYP: identyfikator bloku
FORMAT: FUNCTION <nazwa>[(<lista_param_form>)]:<typ>
PRZYKŁADY:

```
FUNCTION czas: INTEGER;
FUNCTION dzien(d: data): dzientyg;
FUNCTION oblicz(a: INTEGER, b,c,d:REAL): REAL;
```

DZIAŁANIE: Rozpoczyna deklarację funkcji o nazwie podanej jako pierwszy parametr. Po nazwie może być umieszczona lista parametrów formalnych, która wylicza i opisuje wszystkie parametry danej funkcji. Na końcu podany jest typ wyniku zwracanego przez funkcję. Typ ten musi być podany w postaci identyfikatora typu, a nie jego opisu.

Opisane wyżej elementy stanowią nagłówek funkcji. Dalsza część deklaracji może się składać z części wymienionych przy opisie struktury programu w rozdziale 1.3. ("Technika programowania"). W bloku funkcji musi wystąpić przynajmniej jedna instrukcja przypisania, w której po lewej stronie znajdzie się nazwa funkcji. Co najmniej jedna taka instrukcja musi być wykonana podczas wywołania funkcji.

Wywołanie funkcji następuje przez podanie jej nazwy oraz zamkniętej w nawiasy okrągłe listy aktualnych parametrów przekazywanych do funkcji.

GET

TYP: procedura
FORMAT: GET(<zm_plik>)
PRZYKŁADY:

```
GET(x);
GET(plik);
IF NOT EOF(f) THEN GET(f)
```

DZIAŁANIE: Pobiera z pliku wskazanego przez parametr jeden element i przypisuje jego wartość zmiennej buforowej pliku. Z plików sekwencyjnych odczytywany jest element następujący po ostatnio dostępnym, zaś z plików o dostępie swobodnym element wskazany przez procedurę SEEK. Jeśli element taki nie istnieje, to zmienna buforowa ma wartość nieokreśloną, a funkcja EOF przyjmuje wartość TRUE. Jeśli natomiast w chwili wywołania GET funkcja EOF ma wartość TRUE, to wystąpi błąd.

GOTO

TYP: instrukcja
FORMAT: GOTO <etykieta>
PRZYKŁAD:

```
GOTO 100
```

DZIAŁANIE: Instrukcja GOTO pozwala na zmianę normalnej kolejności realizowania programu przez nakazanie wykonania instrukcji oznaczonej podaną etykietą, czyli nakazuje skok do tej instrukcji. Niedozwolone jest wykonywanie skoku spoza instrukcji strukturalnej (FOR, REPEAT, WHILE, CASE, IF) do jej wnętrza. Etykieta może być dowolną liczbą całkowitą bez znaku, która posiada nie więcej niż cztery cyfry. Ponadto etykieta musi być wcześniej zadeklarowana i jest ważna tylko w tym bloku, w którym znajduje się jej deklaracja.

IF

TYP: instrukcja
FORMAT: IF <wyraż_log> THEN <instrukcja>
[ELSE <instrukcja>]
PRZYKŁADY:

```
IF x THEN GOTO 100
IF y>0 THEN x := y ELSE x := -y
IF a=b THEN WRITELN('A = B')
IF a=b THEN WRITE('A = B') ELSE WRITE('A <> B')
```

DZIAŁANIE: Pozwala na wybranie sposobu postępowania w zależności od wartości wyrażenia logicznego (warunku). Jeżeli warunek ma wartość TRUE, to wykonywana jest instrukcja umieszczona po słowie "THEN". W przeciwnym przypadku wykonywana jest instrukcja znajdująca się po słowie "ELSE". Jeśli w instrukcji IF nie ma słowa "ELSE", to realizacja programu jest kontynuowana od następnej instrukcji. Instrukcja wykonywana przez IF może być dowolna, w szczególności dozwolone są instrukcje puste, złażone i inne instrukcje strukturalne.

IN

TYP: operator relacji
FORMAT: <wyr_porz> IN <wyr_zbior>
FRZYKŁADY:

```
WRITELN(x IN dane);
jest := 2*n IN zb1+zb2;
```

DZIAŁANIE: Zwraca wartość TRUE, gdy pierwszy argument jest elementem drugiego, a wartość FALSE w przeciwnym przypadku. Drugi argument musi być typu zbiorowego. Pierwszy argument musi być typu porządkowego zgodnego z typem podstawowym drugiego argumentu. Rezultat operacji jest zawsze typu BOOLEAN.

INPUT

TYP: zmienna
FORMAT: INPUT

PRZYKŁADY:

```
PROGRAM rownanie (INPUT);
PROGRAM obliczenie (INPUT, OUTPUT);
WRITELN(EOF(NPUT));
```

DZIAŁANIE: Słowo "INPUT" jest nazwą zmiennej typu plikowego TEXT, która odpowiada standardowemu urządzeniu wejścia, czyli klawiaturze. INPUT jest wiec standardowo zdefiniowanym plikiem, którego definicja ma postać:

```
VAR INPUT: TEXT;
```

Zmienna INPUT nie wymaga deklarowania w programie. Plik INPUT jest plikiem sekwencyjnym i nie może być traktowany jak plik o dostępie swobodnym (wykonanie na nim procedury SEEK spowoduje błąd). Słowo "INPUT" jest używane w zasadzie tylko do wskazania pliku wejściowego programu oraz w funkcjach EOF, EOLN i w procedurze GET.

INTEGER

TYP: deklaracja typu
FORMAT: <nazwa>[,(nazwa)...] = INTEGER;
lub <nazwa>[,(nazwa)...]: INTEGER;
PRZYKŁADY:

```
TYPE
  liczba = INTEGER; x1,
  x2, x3 = INTEGER;
  macierz = ARRAY [1..2,1..4] OF INTEGER;
VAR
  stos, blad, kod: INTEGER;
  alfa, beta, gamma: ARRAY [litera] OF INTEGER;
```

DZIAŁANIE: Definiuje podaną nazwę jako nazwę typu całkowitego (pierwszy format) lub nazwą zmiennej typu całkowitego (drugi format). Nazwa może być dowolnym, poprawnym identyfikatorem. Słowo "INTEGER" może być użyte w części definiującej typy (po słowie "TYPE" - tylko według pierwszego formatu) albo w części definiującej zmienne (po słowie "VAR" — tylko według drugiego formatu). Ponadto słowo to może być wykorzystywane w definicjach innych zmiennych i innych typów.

LABEL

TYP: identyfikator bloku
FORMAT: LABEL <lista_etykiet>
PRZYKŁAD:

```
LABEL 12, 15, 1000, 456;
```

DZIAŁANIE: Słowo "LABEL" rozpoczyna część deklarującą etykiety w bloku programu, procedury, funkcji lub instrukcji złożonej. Nastąpują po nim oddzielone przecinkami nazwy etykiet. Nazwą etykiety może być dowolna liczba całkowita bez znaku zawierająca mniej niż piąć cyfr.

LN

TYP: funkcja
FORMAT: LN(<wyraż_liczb>)
PRZYKŁADYz

```
WRITELN(LN(y));
x := LN(2*z+1);
a := EXP(5); WRITE(LN(a));
```

DZIAŁANIE: Zwraca wartość logarytmu naturalnego (przy podstawie e = 2.71828) podanego argumentu. Argumentem może być dowolna, większa od zera liczba typu REAL lub INTEGER, a wynik jest zawsze typu REAL. Funkcją odwrotną jest funkcja EXP (patrz trzeci przykład).

LOCAL

TYP: etykieta
FORMAT: LOCAL
DZIAŁANIE: Zdefiniowana w kompilatorze etykieta, która może być stosowana w procedurach napisanych w języku maszynowym. Wartość jej określa adres początku stosu, w którym przechowywane są wartości zmiennych.

MAXINT

TYP: stała
FORMAT: MAXINT
PRZYKŁADY:

```
WRITELN(MAXINT);
liczba := MAXINT;
```

DZIAŁANIE: Stała typu INTEGER równa 32767 i oznaczająca maksymalną dozwoloną wartość liczb typu INTEGER.

MOD

TYP: operator arytmetyczny
FORMAT: <wyraż_liczb> MOD <wyraż_liczb>
PRZYKŁADY:

```
x := 20 MOD 3;
WRITELN(x MOD y);
IF a MOD 2=0 THEN WRITE('liczba parzysta')
```

DZIAŁANIE: Operator MOD realizuje dzielenie modulo, czyli daje w rezultacie resztą z dzielenia pierwszego argumentu przez drugi. Zarówno argumentami, jak wynikiem mogą być tylko liczby typu INTEGER. Odpowiada to operacji:

```
a-b*TRUNC(a/b)
```

NEW

TYP: procedura
FORMAT: NEW(<zm_wskaźn>)

PRZYKŁADY:

```
NEW(p)
NEW(wskaznik)
```

DZIAŁANIE: Powoduje utworzenie nowej zmiennej typu wskazywanego przez wskaźnik będący parametrem procedury. Jednocześnie wskaźnik ten otrzymuje wartość wskazującą na utworzoną zmienną. Zmienna utworzona w ten sposób jest całkowicie nieokreślona. Przykłady użycia procedury NEW są podane w ustępie "Typy wskaźnikowe" (rozdział 1.4).

NIL

TYP: stała
FORMAT: NIL
PRZYKŁADY:

```
wskaźnik := NIL;
p^.nast := NIL;
```

DZIAŁANIE: Stała typu wskaźnikowego oznaczająca pusty wskaźnik.

NOT

TYP: operator logiczny
FORMAT: NOT <wyraż_log>
PRZYKŁADY:

```
x := NOT a;
WRITELN(NOT (a <> b)); IF NOT (a<=0)
THEN WRITE('dodatnia')
```

DZIAŁANIE: Wykonuje operację negacji logicznej argumentu typu BOOLEAN. Dla argumentu o wartości TRUE wynikiem jest FALSE, a dla FALSE wynikiem jest TRUE.

ODD

TYP: funkcja
FORMAT: ODD(<wyraż_liczb>)
PRZYKŁADY:

```
WRITELN(ODD(10)) ;
 IF ODD(x) THEN WRITE('nieparzysta')
```

DZIAŁANIE: Zwraca wartość typu BOOLEAN, która sygnalizuje parzystość argumentu będącego liczbą typu INTEGER. Jeśli argument jest liczbą parzystą, to wynikiem jest wartość FALSE, a w przeciwnym przypadku FALSE.

OF

TYP: instrukcja
FORMAT: patrz opis słów ARRAY, CASE, FILE, RECORD i SET
DZIAŁANIE: Słowo "OF" jest integralną częścią instrukcji ARRAY, CASE, FILE, RECORD i SET. Wymienione są po nim warianty lub elementy tych instrukcji. Patrz też opis słów ARRAY, CASE, 'FILE, RECORD i SET.

OR

TYP: operator logiczny
FORMAT: <wyraż_log> OR <wyraż_log>
PRZYKŁADY:

```
x := a OR b;
WRITELN((a <> b) OR (x/2=Y));
IF (a>0) OR (x<=0) THEN WRITE('prawda')
```

DZIAŁANIE: Wykonuje operację alternatywy (sumy logicznej) dwóch argumentów typu BOOLEAN. Gdy oba argumenty mają wartość FALSE (są fałszywe), to wynikiem jest także FALSE, a w przeciwnym przypadku TRUE.

ORD

TYP: funkcja
FORMAT: ORD(<wyr_typu_porządk>)
PRZYKŁADY:

```
WRITELN(ORD(znak));
IF ORD(n) = 0 THEN WRITE('pierwszy element')
x := ORD('A'); WRITE (CHR (x));
```

DZIAŁANIE: Zwraca wartość typu INTEGER, która określa liczbę porządkową argumentu w jego typie. Dozwolone są wyłącznie argumenty typu porządkowego. Dla typu CHAR odwrotnością ORD jest funkcja CHR.

ORG

TYP: dyrektywa asemblera
FORMAT: ORG <wyraż_liczb>
PRZYKŁADY:

```
ORB $0600
ORG 1536
ORG SP+1024
```

DZIAŁANIE: Powoduje umieszczenie kodu wynikowego procedury maszynowej od adresu określonego przez parametr. Parametr ten może być liczbą szesnastkową. Dyrektywa ORG może być umieszczana wyłącznie w procedurach języka maszynowego rozpoczynających się od dyrektywy #A, lecz nie może znajdować się w pierwszej kolumnie ekranu.

OUTPUT

TYP: zmienna
FORMAT: OUTPUT
PRZYKŁADY:

```
PROGRAM rownanie (OUTPUT); PROGRAM
obliczenie (INPUT, OUTPUT);
WRITELN(EOF(OUTPUT)) ;
```

DZIAŁANIE: Słowo "OUTPUT" jest nazwą zmiennej typu plikowego TEXT, która odpowiada standardowemu urządzeniu wyjścia, czyli ekranowi w trybie 0. OUTPUT jest więc standardowo zdefiniowanym plikiem, którego definicja ma postać:

```
VAR OUTPUT: TEXT;
```

Zmienna OUTPUT nie wymaga deklarowania w programie. Plik OUTPUT jest plikiem sekwencyjnym i nie może być traktowany jak plik o dostępie swobodnym (wykonanie na nim procedury SEEK spowoduje błąd). Słowo "OUTPUT" jest używane w zasadzie tylko do wskazania pliku wyjściowego programu oraz w funkcjach EOF, EOLN i w procedurze PUT.

PACKED

TYP: instrukcja
FORMAT: patrz opis słów ARRAY i RECORD
DZIAŁANIE: Słowo "PACKED" jest częścią deklaracji ARRAY i RECORD, która może być stosowana w razie potrzeby. Nakazuje ona gęstsze upakowanie informacji w pamięci komputera. Powoduje to zmniejszenie obszaru pamięci zajmowanego przez zmienne tablicowe i rekordowe, lecz zwiększa czas dostępu do nich. W Kyan Pascalu zapis danych w pamięci jest zawsze taki sam, a słowo PACKED pozostawiono jedynie dla zapewnienia zgodności ze standardowym Pascalem. Patrz też opis słów ARRAY i RECORD.

PRED

TYP: funkcja
FORMAT: PRED(<wyr_typu_porządk>)
PRZYKŁADY:

```
WRITELN(PRED(znak));
a := PRED(b);
IF PRED(n) = 'A' THEN WRITE('litera B)
x := PRED('3'); WRITE(SUCC(x));
```

DZIAŁANIE: Zwraca wartość, której liczba porządkowa w typie jest o jeden mniejsza od liczby porządkowej argumentu, czyli zwraca poprzednik argumentu. Argument musi być typu porządkowego, a wynik ma zawsze ten sam typ co argument.

PROCEDURE

TYP: identyfikator bloku
FORMAT: PROCEDURE <nazwa>[<<lista_param_form>)]
PRZYKŁADY:

```
PROCEDURE czas;
PROCEDURE punkt (x,y: INTEGER);
PROCEDURE dzien(d: data);
PROCEDURE oblicz<a: INTEGER, b,c,d:REAL);
```

DZIAŁANIE: Rozpoczyna deklarację procedury o nazwie podanej jako pierwszy parametr. Po nazwie może być umieszczana lista parametrów formalnych, która wylicza i opisuje wszystkie parametry danej procedury.
Opisane wyżej elementy stanowią nagłówek procedury. Dalsza część deklaracji może się składać z części wymienionych przy opisie struktury programu w rozdziale 1.3. ("Technika programowania").

Wywołanie procedury następuje przez podanie jej nazwy oraz zamkniętej w nawiasy okrągłe listy aktualnych parametrów przekazywanych do procedury.

PROGRAM

TYP: identyfikator bloku
FORMAT: PROGRAM <nazwa>[(<lista_plików>)]
PRZYKŁADY:

```
PROGRAM czas;
PROGRAM rysunek (INPUT,OUTPUT);
PROGRAM kalkulacja (INPUT,dane);
PROGRAM obliczenie (dane,wyniki);
```

DZIAŁANIE: Rozpoczyna treść programu o nazwie podanej jako pierwszy parametr. Nazwa programu nie ma żadnego znaczenia i służy jedynie do jego odróżnienia. Po nazwie może być umieszczona lista plików, które będą wykorzystywane podczas programu. Kyan Pascal, w odróżnieniu od standardu Pascala, nie wymaga bezwzględnie umieszczenia identyfikatorów pliku w nagłówku programu.

Opisane wyżej elementy stanowią nagłówek programu. Dalsza treść może się składać z części wymienionych przy opisie struktury programu w rozdziale 1.3. ("Technika programowania").

PUT

TYP: procedura
FORMAT: PUT(<zm_plik>)
PRZYKŁADY:

```
PUT(x);
PUT(plik);
IF EOF(f) THEN PUT(f)
```

DZIAŁANIE: Zapisuje do pliku wskazanego przez parametr jeden element, którego wartość określa zmienna buforowa pliku. Do plików sekwencyjnych element jest zapisywany na końcu pliku, zaś do plików o dostępie swobodnym w miejscu wskazanym przez procedurę SEEK. Jeśli w chwili wywołania PUT dla pliku sekwencyjnego funkcja EOF ma wartość FALSE, to wystąpi błąd.

READ

TYP: procedura
FORMAT: READ([<zm_plik>,]<zmienna>[,<zmienna>...])
PRZYKŁADY:

```
READ(alfa);
READ(plik,alfa);
READ(alfa,beta,gamma);
```

DZIAŁANIE: Pobiera z pliku wskazanego przez pierwszy parametr element lub elementy i przypisuje ich wartość zmiennym, które są dalszymi parametrami. Jest ona określona tylko dla plików sekwencyjnych. Procedurze READ(f,z) odpowiada instrukcja złożona:

```
BEGIN
z := f^;
GET(f) END
```

Jeżeli zostanie opuszczona zmienna plikowa, która wskazuje odczytywany plik, to przyjmowana jest standardowa zmienna INPUT. W przypadku odczytywania kilku elementów przez jedną procedurę READ separatorami są spacja i znak końca wiersza.

READLN

TYP: procedura
FORMAT: READLN([<zm_plik>,]<zmienna>[,<zmienna>...])
PRZYKŁADY:

```
READLN(alfa) ;
READLN(plik,alfa);
READLN(alfa,beta,gamma) ;
```

DZIAŁANIE: Pobiera z pliku wskazanego przez pierwszy parametr elementy umieszczone pomiędzy dwoma następnymi znakami końca wiersza i przypisuje ich wartość zmiennym, które są dalszymi parametrami. Jeśli odczytanych wartości jest więcej niż podanych dla nich zmiennych, to pozostałe wartości zostają utracone. Procedurze READLN(f,z) odpowiada wiec instrukcja złożona:

```
BEGIN
  z  :=  f^;
  GET(f);
  BEGIN
    WHILE NOT EOLN(f) DO GET(f);
      GET(f)
    END
  END
```

Jeżeli zostanie opuszczona zmienna plikowa, która wskazuj* odczytywany plik, to przyjmowana jest standardowa zmienna INPUT. W przypadku odczytywania kilku elementów przez jedną procedurą READLN separatorami są spacja i znak końca wiersza. READLN jest określona tylko dla plików sekwencyjnych.

REAL

TYP: deklaracja typu
FORMAT: <nazwa>[,(nazwa)...] = REAL;
lub <nazwa>[,<nazwa>...]: REAL;
PRZYKŁADY:

```
TYPE
  wynik = REAL;
  x1, x2,  x3  =  REAL,;
  wektor = ARRAY [1..2,1..2] OF REAL;
VAR
  suma, licznik, wynik: REAL;
  alfa, beta, gamma: ARRAY [litera] OF REAL;
```

DZIAŁANIE: Definiuje podaną nazwą jako nazwą typu

rzeczywistego (pierwszy format) lub nazwę zmiennej typu rzeczywistego (drugi format). Nazwa może być dowolnym, poprawnym identyfikatorem. Słowo "REAL" może być użyte w części definiującej typy (po słowie "TYPE" — tylko według pierwszego formatu) albo w części definiującej zmienne (po słowie "VAR" -tylko według drugiego formatu). Ponadto słowo to może być wykorzystywane w definicjach innych zmiennych i innych typów.

RECORD

TYP: deklaracja typu
FORMAT: <nazwa>[,<nazwa>] = [PACKED] RECORD
<lista_pól> END;
lub <nazwa>[,<nazwa>]: [PACKED] RECORD
<lista_pól> END;
PRZYKŁADY:

```
TYPE
  opis - PACKED RECORD a, b, c: napis END;
  data = RECORD dzien: 1..31;
         miesiac: 1..12;
         rok: 0..maxint END;
zesp = FILE OF RECORD re, im: REAL END;
osoba = RECORD imie, nazw: tekst;
                urodz: data;
                CASE pl: plec OF
                  kob: (nazwpan: tekst);
                  mezcz: (wojsko: BOOLEAN)
          END;
 VAR
  dana: RECORD CASE BOOLEAN OF
               FALSE: (i1:INTEGER);
               TRUE: (i2, i3: INTEGER)
        END;
  zl, z2, z3: RECORD re, im: REAL END;
  tabl: ARRAY [1..10] OF RECORD a: CHAR;
                              b: INTEGER
                         END;
```

DZIAŁANIE: Definiuje podaną nazwę jako nazwą typu rekordowego (pierwszy format) lub nazwę zmiennej typu rekordowego (drugi format). Nazwa może być dowolnym, poprawnym identyfikatorem. Szczegółowy opis formatu deklaracji typu lub zmiennej znajduje się w ustępie "Rekordy" (rozdział 1.4). Słowo "RECORD" może być użyte w części definiującej typy (po słowie "TYPE" - tylko według pierwszego formatu) albo w części definiującej zmienne (po słowie "VAR" - tylko według drugiego formatu). Ponadto słowo to może być wykorzystywane w definicjach innych zmiennych i innych typów.

REPEAT

TYP: instrukcja
FORMAT: patrz opis instrukcji UNTIL
DZIAŁANIE: Instrukcja REPEAT oznacza początek pętli

REPEAT/UNTIL. Następują po niej inne instrukcje, które są objęte tą pętlą (patrz opis instrukcji UNTIL).

RESET

TYP: procedura
FORMAT: RESET(<zm_plik>[,<spec_pliku>])
PRZYKŁADY:

```
RESET(plik);
RESET(plik,'D2:BLOK.DAT');
RESET(plik,nazwa);
```

DZIAŁANIE: Otwiera do odczytu plik wskazany przez pierwszy parametr i zmiennej buforowej przypisuje wartość pierwszego elementu pliku. Jeśli plik jest pusty, to wartość zmiennej buforowej jest nieokreślona. Dostąp do pliku umieszczonego w pliku dyskowym jest realizowany poprzez drugi parametr, określający specyfikację tego pliku. Jeśli po RESET zostanie wywołana procedura SEEK, to plik jest otwierany do zapisu i odczytu. Ponieważ w Pascalu nie ma procedury CLOSE, to można otworzyć tylko pięć plików zewnętrznych. Lista wykorzystywanych plików nie musi być umieszczana w nagłówku programu.

REWRITE

TYF: procedura
FORMAT: REWRITE(<zm_plik>[,<spec_pliku>])
PRZYKŁADY:

```
REWRITE(plik);
REWRITE(plik,'D2:BLOK.DAT');
REWRITE(plik,nazwa);
```

DZIAŁANIE: Otwiera do zapisu plik wskazany przez pierwszy parametr, przez co plik ten staje się plikiem pustym. Dostęp do pliku umieszczonego w pliku dyskowym jest realizowany poprzez drugi parametr, określający specyfikację tego pliku. Ponieważ w Pascalu nie ma procedury CLOSE, to można otworzyć tylko pięć plików zewnętrznych. Lista wykorzystywanych plików nie musi być umieszczana w nagłówku programu.

ROUND

TYP: funkcja
FORMAT: ROUND(<wyraż_liczb>)
PRZYKŁADY:

```
WRITELN(ROUND(2.55));
x := ROUND(y);
IF ROUND(y) = y THEN WRITE('calkowita')
```

DZIAŁANIE: Zwraca wartość typu INTEGER, która jest równa zaokrąglonej wartości argumentu. Dozwolone są wyłącznie argumenty typu REAL. Wynik funkcji ROUND jest obliczany według wzorów:

```
ROUND(x) = TRUNC(x+0.5)   dla X>=0
ROUND(x) = TRUNC(x-0.5)   dla x<0
```

SEEK

TYP: procedura
FORMAT: SEEK(<zm_plik>,<wyraż_liczb>)
PRZYKŁADY:

```
SEEK(plik,10);
SEEK(plik,licznik);
SEEK(f,n*2-m);
```

DZIAŁANIE: Ustawia wskaźnik pliku wskazanego przez pierwszy parametr na elemencie, którego numer jest drugim parametrem. Numer elementu może mieć wartość typu INTEGER z zakresu od 0 do liczby elementów pliku zmniejszonej o jeden. Wykonanie po SEEK procedury GET odczyta element wskazany przez SEEK, zaś wykonanie PUT zapisze element wskazany przez SEEK. Procedura ta jest określona tylko dla plików wcześniej utworzonych (otwartych przez RESET), przy czym nie mogą to być pliki tekstowe i z elementami typu BOOLEAN. Przypisanie zmiennej buforowej nowej wartości dla procedury PUT musi nastąpić dopiero po wywołaniu SEEK, gdyż w przeciwnym przypadku rezultat będzie nieprawidłowy.

SET

TYP: deklaracja typu
FORMAT: <nazwa>[,<nazwa>] = SET OF <typ>
lub <nazwa>[,<nazwa>]: SET OF <typ>
PRZYKŁADY:

```
TYPE
dyzur = SET OF dzien;
swieto = SET OF pt .. niedz;
liczba = SET OF 1 .. 10; VAR
  dyzur,  wolne:  SET  OF  dzien;
  swieto: SET OF pt .. niedz;
  11, 12, 13, 14: SET OF 1 .. 10;
```

DZIAŁANIE: Definiuje podaną nazwę jako nazwę typu zbiorowego (pierwszy format) lub nazwę zmiennej typu zbiorowego (drugi format). Nazwa może być dowolnym, poprawnym identyfikatorem, a typ podstawowy musi być typem porządkowym. Słowo "SET" może być użyte w części definiującej typy (po słowie "TYPE" - tylko według pierwszego formatu) albo w części definiującej zmienne (po słowie "VAR" - tylko według drugiego formatu). Ponadto słowo to może być wykorzystywane w definicjach innych zmiennych i innych typów.

SIN

IYP: funkcja
FORMAT: SIN(<wyraż_liczb>)
PRZYKŁADY:

```
WRITELN(SIN(45));
x := SIN(y);
IF SIN(z) = 0.0 THEN WRITE('kat = 0')
```

DZIAŁANIE: Zwraca wartość funkcji sinus podanego argumentu,

który określa wielkość kąta w radianach. Wielkość argumentu jest dowolna, a wynik jest zawsze z zakresu od -1 do +1. Argument może być typu REAL lub INTEGER, a wynik jest zawsze typu REAL.

SP

TYP: etykieta
FORMAT: SP
DZIAŁANIE: Zdefiniowana w kompilatorze etykieta, która może być stosowana w procedurach napisanych w języku maszynowym. Wartość jej określa zwiększony o trzy adres szczytu stosu, w którym przechowywane są wartości zmiennych.

SQR

TYP: funkcja
FORMAT: SQR(<wyraż_liczb>)
PRZYKŁADY:

```
WRITELN(SQR(4)) ;
kwadrat : = SQR(liczba);
IF SQR(x)<100 THEN obliczenie
p := SQR(2.5); WRITE(SQRT(p));
```

DZIAŁANIE: Zwraca wartość będącą kwadratem podanego argumentu, czyli podnosi argument do potęgi 2. Argument może być typu INTEGER lub REAL, a wynik jest zawsze tego samego typu co argument. Jeśli wynik przekracza wartość dozwoloną dla typu argumentu, to wystąpi błąd. Odwrotnością SQR jest funkcja SQRT (patrz czwarty przykład).

.

SQRT

TYP: funkcja
FORMAT: SQRT(<wyraż_liczb>)
PRZYKŁADY:

```
y := SQR(2*x);
WRITELN(SQR(ABS(SIN(z))));
IF a >= 0.0 THEN k := 2*SQR(a)
p := SQR(2.5); WRITE(SQRT(p));
```

DZIAŁANIE: Zwraca wartość będącą pierwiastkiem kwadratowym podanego argumentu, czyli podnosi argument do potęgi -2. Argument może być typu INTEGER lub REAL, a wynik jest zawsze typu REAL. Jeśli argument jest mniejszy od zera, to wystąpi błąd. Odwrotnością SQRT jest funkcja SQR (patrz czwarty przykład).

SUCC

TYP: funkcja
FORMAT: SUCC(<wyr_typu_porządk>)
PRZYKŁADY:

```
WRITELN(SUCC(znak));
a := SUCC(b);
```

```
IF SUCC(n) = 'C' THEN WRITE('litera B')
x := PRED('3'); WRITE(SUCC(x));
```

DZIAŁANIE: Zwraca wartość, której liczba porządkowa w typie jest o jeden większa od liczby porządkowej argumentu, czyli zwraca następnik argumentu. Argument musi być typu porządkowego, a wynik ma zawsze ten sam typ co argument.

T

TYP: etykieta
FORMAT: T
DZIAŁANIE: Zdefiniowana w kompilatorze etykieta, która może być stosowana w procedurach napisanych w języku maszynowym. Wartość jej określa adres pierwszego rejestru tymczasowego na stronie zerowej. Rejestrów tych jest szesnaście, a zajmują one adresy od T do T+15.

TEXT

TYP: deklaracja typu
FORMAT: <nazwa>[.<nazwa>...]: TEXT;
PRZYKŁADY:

```
TYPE
  opis = RECORD a, b, c: TEXT END;
VAR
  kartoteka: TEXT;
  kalendarz, notes: TEXT;
```

DZIAŁANIE: Definiuje podaną nazwę jako nazwę zmiennej typu plikowego, której elementy są typu CHAR. TEXT jest wiec standardowo zdefiniowanym typem plikowym, którego definicja ma postać:

```
TYPE TEXT = FILE OF CHAR;
```

Nazwa może być dowolnym, poprawnym identyfikatorem. Słowo "TEXT" może być użyte w części definiującej zmienne (po słowie "VAR") lub jako element deklaracji innego typu lub zmiennej (w tym przypadku także w części deklarującej typy — po słowie "TYPE"). Pliki typu TEXT są zawsze plikami sekwencyjnymi.

THEN

TYP: instrukcja
FORMAT: patrz opis instrukcji IF
DZIAŁANIE: Słowo THEN stanowi integralną część instrukcji IF. Następuje po nim instrukcja, wykonywana gdy warunek przed THEN jest prawdziwy. Może to być także instrukcja złożona. Patrz też opis instrukcji IF.

TO

TYP: instrukcja
FORMAT: patrz opis instrukcji FOR

DZIAŁANIE: Słowo "TO" stanowi integralną część składową instrukcji FOR nakazującą zliczanie pętli w kierunku większych wartości. Następujący po nim parametr określa górną wartość graniczną pętli, czyli maksymalną wartość licznika. Wartość graniczna nie może być zmieniana wewnątrz pętli. Patrz też opis instrukcji FOR.

TRUE

TYP: stała
FORMAT: TRUE
PRZYKŁADY:

```
WRITELN(TRUE);
stan := TRUE;
```

DZIAŁANIE: Stała typu BOOLEAN oznaczająca wartość logiczną "prawda".

TRUNC

TYP: funkcja
FORMAT: TRUNC(<wyraż_liczb>)
PRZYKŁADY:

```
WRITELN(TRUNC(2.55));
x := TRUNC(y);
IF TRUNC(y) = y THEN WRITE('calkowita')
```

DZIAŁANIE: Zwraca wartość typu INTEGER, która jest częścią całkowitą argumentu. Dozwolone są wyłącznie argumenty typu REAL. Jeśli argument jest mniejszy od zera, to wynikiem jest najmniejsza liczba całkowita nie mniejsza niż wartość argumentu, a w przeciwnym przypadku największa liczba całkowita nie większa niż argument.

TYPE

TYP: identyfikator bloku
FORMAT: TYPE <lista_defin_typów>
PRZYKŁAD:

```
TYPE
  stan, dalej = BOOLEAN;
  pliki = FILE OF REAL;
  wektor = ARRAY [1..2,1..2] OF REAL;
```

DZIAŁANIE: Słowo "TYPE" rozpoczyna część definiującą typy zmiennych w bloku programu, procedury, funkcji lub instrukcji złażonej. Następują po nim oddzielone średnikami nazwy typów i ich definicje.

UNTIL

TYP: instrukcja
FORMAT: REPEAT <ciąg_instr> UNTIL <wyraż_log>
PRZYKŁADY:

```
REPEAT oblicz(a) UNTIL stan;
```

```
REPEAT x:=2*a; y:=0.5*a UNTIL x+y>100;
REPEAT x:=x+1 UNTIL x>100;
```

DZIAŁANIE: Instrukcja UNTIL oznacza koniec pętli REPEAT/UNTIL i zamyka blok instrukcji objętych przez tą pętlę. Wszystkie instrukcje zawarte pomiędzy słowami "REPEAT" i "UNTIL" są wykonywane, dopóki <wyraż_log> ma wartość FALSE. Wartość TRUE powoduje przerwanie pętli i wykonanie instrukcji następującej pc UNTIL. Warunek ten jest sprawdzany na końcu pętli, więc zawarte w niej instrukcje muszą być wykonane co najmniej jeden raz.

VAR

TYP: identyfikator bloku
FORMAT: VAR <lista_deklar_zmiennych>
PRZYKŁAD:

```
VAR
  a,b: INTEGER;
  dzien: data;
  wektor: ARRAY [1..2,1..2] OF REAL;
```

DZIAŁANIE: Słowo "VAR" rozpoczyna część deklarującą zmienne w bloku programu, procedury, funkcji lub instrukcji . złożonej. Następują po nim oddzielone średnikami nazwy zmiennych oraz ustalone dla nich typy.

WHILE

TYR; instrukcja
FORMAT: WHILE <wyraż_log> DO <instrukcja>
PRZYKŁADY:

```
WHILE a<100 DO oblicz(a);
WHILE NOT EOF DO READLN;
WHILE x<100 DO x := x+1;
```

DZIAŁANIE: Instrukcja WHILE nakazuje wykonywanie instrukcji zawartej po słowie "DO", dopóki <wyraż_log> ma wartość TRUE. Wartość FALSE powoduje przerwanie pętli i wykonanie następnej instrukcji. Warunek ten jest sprawdzany na początku pętli, więc zawarte w niej instrukcje mogą wcale nie być wykonane. Po słowie "DO" może być umieszczona dowolna instrukcja, w tym również złożona lub pusta.

WITH

TYP: instrukcja
FORMAT: WITH <zm_rekord>C,<zm_rekord>. ..3 DO <instrukcja>
PRZYKŁADY:

```
WITH a[i] DO b := 5
WITH r1,r2,r3 DO cokolwiek
WITH r1 DO
    WITH r2 DO
        WITH r3 DO cokolwiek
```

DZIAŁANIE: Pozwala na wygodniejsze odwoływanie się do pól zmiennej rekordowej i zwiększa czytelność programu. Wewnątrz instrukcji WITH można bowiem używać samych identyfikatorów pól

bez identyfikatora rekordu. Instrukcja ta zastępuje więc odwołanie typu a[i].b := 5 (patrz pierwszy przykład). <zm_rekord> musi być w tym przypadku nazwą zmiennej rekordowej.

WRITE

TYP: procedura
FORMAT: WRITE([<zm_plik>,]<wyraż>[:<wyr_liczb[:<wyr_liczb]]
[,<wyraż>[:<wyr_liczb[:<wyr_liczb]]...])
PRZYKŁADY:

```
WRITE(alfa);
WRITE(plik,alfa) ;
WRITE(alfa,beta,gamma);
WRITE(alfa: x+y);
WRITE(alfa: 7: 2);
WRITE(alfa: 2, beta: n, gamma: 12: 4);
WRITE(plik,alfa: 7: 2*n) ;
```

DZIAŁANIE: Zapisuje do pliku wskazanego przez pierwszy parametr element lub elementy, których wartości są dalszymi parametrami. Jest ona określona tylko dla plików sekwencyjnych. Procedurze WRITE(f,z) odpowiada instrukcja złożona:

```
BEGIN
  f^ := z;
  PUT(f)
END
```

Jeżeli zostanie opuszczona zmienna plikowa, która wskazuje zapisywany plik, to przyjmowana jest standardowa zmienna OUTPUT. W przypadku zapisywania kilku elementów przez jedną procedurę WRITE są one zapisywane jeden za drugim bez odstępów.

Umieszczenie po zapisywanym wyrażeniu kolejnych wyrażeń oddzielonych dwukropkiem służy do ustalenia formy zapisu. Pierwszy z tych parametrów (zawsze typu INTEGER) określa szerokość pola, w którym musi zmieścić się zapisywana wartość. Jeśli ma ona mniej znaków, to przed nią dopisywane są spacje. W przeciwnym przypadku koniec napisu jest obcinany, a dla wartości liczbowych wystąpi błąd. Dla liczb typu REAL dozwolony jest jeszcze jeden parametr (także typu INTEGER). Wyznacza on dokładność zapisu wartości, czyli liczbę cyfr zapisywanych po punkcie dziesiętnym. W takim przypadku liczba jest zapisywana w normalnej postaci, zamiast w postaci wykładniczej.

WRITELN

TYP: procedura
FORMAT: WRITELNC[([<zm_plik>,][<wyraż>[:<wyr_liczb>
[:<wyr_liczb>]][,<wyraż>[:<wyr_liczb>
[:<wyr_liczb]]...]])]
PRZYKŁADY:

```
WRITELN;
WRITELN(plik);
WRITELN(alfa);
WRITELN(plik,alfa);
```

```
WRITELN (alfa, beta, gamma);
WRITELN(alfa: x+y) ;
WRITELN(alfa: 7: 2);
WRITELN(alfa: 2, beta: n, gamma: 12: 4);
WRITELN(plik,alfa: 7: 2*n);
```

DZIAŁANIE: Zapisuje do pliku wskazanego przez pierwszy parametr element lub elementy, których wartości są dalszymi parametrami. Na końcu zapisywanych danych zostaje zawsze umieszczony znak końca wiersza, a jeśli jedynym parametrem jest wskaźnik pliku, to zostaje zapisany tylko ten znak- Procedurze WRITELN(f,z) odpowiada więc instrukcja złożona:

```
BEGIN
  f^ := z;
PUT (f);
WRITELN(f; END
```

Jeżeli zostanie opuszczona zmienna plikowa, która wskazuje zapisywany plik, to przyjmowana jest standardowa zmienna OUTPUT. W przypadku zapisywania kilku elementów przez jedną procedurą WRITELN są one zapisywane jeden za drugim bez odstępów. WRITELN jest określona tylko dla plików sekwencyjnych.

Umieszczenie po zapisywanym wyrażeniu kolejnych wyrażeń oddzielonych dwukropkiem służy do ustalenia formy zapisu. Pierwszy z tych parametrów (zawsze typu INTEGER) określa szerokość pola, w którym musi zmieścić się zapisywana wartość. Jeśli ma ona mniej znaków, to przed nią dopisywane są spacje. W przeciwnym przypadku koniec napisu jest obcinany, a dla wartości liczbowych wystąpi błąd. Dla liczb typu REAL dozwolony jest jeszcze jeden parametr (także typu INTEGER). Wyznacza on dokładność zapisu wartości, czyli liczbę cyfr zapisywanych po punkcie dziesiętnym. W takim przypadku liczba jest zapisywana w normalnej postaci, zamiast w postaci wykładniczej.

#A

TYP: dyrektywa kompilatora
FORMAT: #A
PRZYKŁAD:

```
#A
   LDA #$40 STA
   $6A
#
```

DZIAŁANIE: Rozpoczyna blok procedury maszynowej napisanej w asemblerze procesora 6502. Cała treść tej procedury musi być umieszczona pomiędzy napisami "#A" i "#", które muszą znajdować się w pierwszej kolumnie ekranu. W treści procedury maszynowej należy stosować ogólnie przyjęte mnemoniki rozkazów (patrz "Poradnik programisty Atari"), przy czym w pierwszej kolumnie ekranu wolno umieszczać tylko etykiety. Ponieważ kompilator Kyan Pascal używa etykiet rozpoczynających się od litery L, to nie wolno ich stosować we własnej procedurze, gdyż spowoduje to błąd.

#I

TYP: dyrektywa kompilatora
FORMAT: #I <spec_pliku>
PRZYKŁADY:

```
#I D:PLOT.I
#I D8:SOUND.I
```

DZIAŁANIE: Dołącza do kompilowanego programu procedury zawarte w pliku dyskowym wskazanym w dyrektywie. Umożliwia to stworzenie biblioteki procedur i uproszczenie pisanych później programów. Ponadto dzięki tej dyrektywie możliwe jest dołączanie do programów dodatkowych procedur znajdujących się na dyskietce Kyan Pascala (patrz rozdział 1.6).

(* *)

TYP: instrukcja
FORMAT: (*[<dowolny ciąg znaków)]*}
PRZYKŁADY:

```
(**)
(* ** TO JEST KOMENTARZ ** *)
(* można wszystko: ,.;?#"$"% PROGRAM *)
oblicz(x) (* obliczenie silni *);
```

DZIAŁANIE: Znaki te ograniczają komentarz, który jest podczas kompilacji programu całkowicie ignorowany przez kompilator. Po napotkaniu komentarza dalsza kompilacja programu jest wykonywana od następnej instrukcji. Pozwala to na umieszczenie w programie komentarzy wyjaśniających działanie poszczególnych fragmentów programu, znaczenie zmiennych itp. W komentarzu dozwolone są wszystkie dostępne znaki.

:=

TYP: instrukcja
FORMAT: <zmienna> := <wyrażenie>
PRZYKŁADY:

```
x := 100;
nazwa := 'ATARI';
y[i] := EXP(a)+LN(2*z);
p^.nast := pomoc;
```

DZIAŁANIE: Przypisuje wskazanej zmiennej wartość podanego wyrażenia. Wartość wyrażenia jest obliczana przed wykonaniem przypisania. Jeśli typ wartości nie jest zgodny z typem zmiennej, to wystąpi błąd. Możliwe jest jednak przypisanie wartości typu INTEGER zmiennej typu REAL.

1. 6. Procedury dodatkowe

Dyskietka Kyan Pascala zawiera - poza właściwym edytorem i kompilatorem Pascala - dodatkowe procedury. Pozwalają one na wykorzystanie możliwości sprzęgowych Atari (głównie graficznych i dźwiękowych). Są one zapisane w postaci źródłowej w nastąpujących plikach:

```
CONCAT.I     - procedura CONCAT
DRAWTO.I     - procedura DRAWTO
GRAPHICS.I   - procedura GRAPHICS
 INDEX.I     - funkcja INDEX
LENGTH.I     - funkcja LENGTH
LOCATE.I     - procedura LOCATE
PLOT.I       - procedura PLOT
POSITION.I   - procedura POSITION
PR.I         - procedury PRON i PROFF
RANDOM.I     - funkcja RANDOM
SETCOLOR.I   - procedura SETCOLOR
SOUND.I      - procedura SOUND
SUBSTRIN.I   - funkcja SUBSTRING
```

Wszystkie te procedury i funkcje można dołączyć do własnego programu przez odczytanie ich do edytora lub poprzez wykorzystanie dyrektywy #I. Ponieważ stanowią one integralną cze.sc Kyan Pascala, to ich opis znajduje się w tym rozdziale.

CONCAT

TYP: procedura
FORMAT: CONCAT(<zm_tab1>,<zm_tab1>,<zm_tab1>)
PRZYKŁAD:

```
CONCAT(s1,s2,s3);
CONCAT(imię,nazw,osoba) ;
```

DZIAŁANIE: Łączy tablice tekstowe, których nazwy zostały podane jako dwa pierwsze parametry, a rezultat umieszcza w tablicy o nazwie będącej trzecim parametrem. Druga tablica jest dopisywana na końcu pierwszej, aż do osiągnięcia maksymalnej dozwolonej długości tablicy. Znaki przekraczające tą długość zostają pominięte. Procedura CONCAT wymaga wcześniejszego zadeklarowania stałej MAXSTRING oraz tablic, które powinny być typu string = ARRAY [1..maxstring] OF CHAR.

DRAWTO

TYP: procedura
FORMAT: DRAWTO(<wyraż_liczb>,<wyraż_liczb>,<wyraż_liczb>)
PRZYKŁADY:

```
DRAWTO(10,10,1);
DRAWTO(x,y,barwa);
DRAWT0(k*2,I+10,kolor);
```

DZIAŁANIE: Rysuje na ekranie linię prostą od aktualnego położenia kursora do punktu o współrzędnych będących dwoma

pierwszymi parametrami. Ostatni parametr wybiera kolor (w trybach bitowych) lub znak (w trybach znakowych, który zostanie użyty do narysowania linii. Wszystkie parametry muszą być typu INTEGER. Punkt (0,0) znajduje się w lewym, górnym rogu ekranu. Pierwsza liczba określa współrzędną poziomą, a druga pionową. Przekroczenie wartości współrzędnych dopuszczalnych w danym trybie graficznym powoduje błąd.

GRAPHICS

TYP: procedura
FORMAT: GRAPHICS(<wyraż_liczb>)
PRZYKŁADY:

```
GRAPHICS(1);
GRAPHICS(1+32) ;
GRAPHICS(2*tryb);
GRAPHICS(x);
```

DZIAŁANIE: Wybiera tryb wyświetlania obrazu według podanego parametru, który musi być typu INTEGER. System operacyjny Atari umożliwia uzyskanie szesnastu różnych trybów graficznych: pięciu znakowych i jedenastu bitowych. Wybierane są one przez wartości parametru z zakresu od 0 do 15. Ponadto zwiększenie numeru trybu o 16 ustala tryb graficzny bez okna tekstowego, które ma zawsze tryb GRAPHICS(0). Numer trybu zwiększony o 32 powoduje zachowanie niezmienionej zawartości obszaru pamięci obrazu. Oba te warianty mogą być zastosowane łącznie - numer trybu musi być wtedy zwiększony o 48. Wykaz dostępnych trybów oraz ich szczegółowy opis znajduje się w "Poradniku programisty Atari".

Ponieważ poniżej pamięci obrazu znajduje się biblioteka Kyan Pascala, to przy wybieraniu trybu wymagającego większego obszaru pamięci niż tryb O, należy obniżyć RAMTOP (adres 106) do wartości zapewniającej zabezpieczenie biblioteki przed zniszczeniem. Aby umożliwić poprawną pracę kompilatora i edytora, przed zakończeniem programu trzeba przywrócić poprzednią wartość rejestru RAMTOP (192).

INDEX

TYP: funkcja
FORMAT: INDEX(<zm_tabl>,<zm_tabl>)
PRZYKŁADY:

```
WRITELN(INDEX(s1,s2);
poz := (INDEX(osoba,imie);
```

DZIAŁANIE: Zwraca wartość typu INTEGER określającą miejsce, od którego druga tablica tekstowa zawiera się w pierwszej. Parametry określające tablice muszą być nazwami tych tablic. Jeśli druga tablica nie zawiera się w pierwszej, to wynikiem funkcji jest zero. Funkcja INDEX wymaga wcześniejszego zadeklarowania stałej MAXSTRING oraz tablic, które powinny być typu string = ARRAY [1..maxstring] OF CHAR.

LENGTH

TYP: funkcja
FORMAT: LENGTH(<zm_tabl>)
PRZYKŁADY:

```
WRITELN(LENGTH(s));
len := LENGTH(nazwa);
```

DZIAŁANIE: Zwraca wartość typu INTEGER określającą długość tablicy tekstowej, której nazwa jest parametrem funkcji. Jako ostatni znak tablicy funkcja traktuje pierwszą napotkaną spacje lub koniec tablicy, jeśli nie zawiera ona spacji. Funkcja LENGTH wymaga wcześniejszego zadeklarowania stałej MAXSTRING oraz tablicy, która powinna być typu:

```
string = ARRAY [1..maxstring] OF CHAR.
```

LOCATE

TYP: procedura
FORMAT: LOCATE (<wyraż_liczb>,<wyraż_liczb>, <zmienna>)
PRZYKKŁADY:

```
LOCATE(10,5,a);
LOCATE(x*2,y,z);
```

DZIAŁANIE: Odczytuje z miejsca ekranu, którego współrzędne są określone przez dwa parametry typu INTEGER, znajdujący się tam znak lub kolor i jego kod przypisuje wskazanej jako trzeci parametr zmiennej (także typu INTEGER). W trybie znakowym jest to kod ATASCII znaku, zaś w trybie bitowym - kod koloru. Wartości te odpowiadają użytym w instrukcjach PLOT i DRAWTO.

PLOT

TYP: procedura
FORMAT: PLOT(<wyraż_liczb>,<wyraż_liczb>,<wyraż_liczb>)
PRZYKŁADY:

```
PLOT(10,10);
PLOT(K,y);
PLOT(i*2,j+10);
```

DZIAŁANIE: Umieszcza na ekranie, w miejscu o współrzędnych podanych przez dwa pierwsze parametry, punkt lub znak określony przez trzeci parametr. Wszystkie parametry muszą być typu INTEGER. Punkt (0,0) znajduje się w lewym, górnym rogu ekranu. Pierwsza liczba określa współrzędną poziomą, a druga pionową. Przekroczenie wartości współrzędnych dopuszczalnych w danym trybie graficznym powoduje błąd.

POSITION

TYP: procedura
FORMAT: POSITION(<wyraż_liczb>,<wyraż_liczb>)
PRZYKŁADY:

```
POSITION(10,10);
POSITION(x,y);
POSITION(i*2,j + 10);
```

DZIAŁANIE: Umieszcza kursor na ekranie w miejscu o współrzędnych podanych przez parametry typu INTEGER. Punkt (0,0) znajduje się w lewym, górnym rogu ekranu. Pierwsza liczba określa współrzędną poziomą, a druga pionową. Przekroczenie wartości współrzędnych dopuszczalnych w danym trybie graficznym powoduje błąd.

PROFF

TYP: procedura
FORMAT: PROFF
PRZYKŁAD:

```
PROFF;
```

DZIAŁANIE: Zamyka kanał komunikacji z drukarką i w ten sposób przerywa wyprowadzanie danych na to urządzenie. Po wywołaniu PROFF wszystkie wyniki są ponownie wyświetlane na ekranie.

PRON

TYP: procedura
FORMAT: PRON
PRZYKŁAD:

```
PRON;
```

DZIAŁANIE: Otwiera kanał komunikacji z drukarką. Teraz wszystkie procedury WRITE i WRITELN wykonywane na standardowym pliku OUTPUT będą wyprowadzały informację na drukarkę, zamiast na ekran. Zakończenie pracy z drukarką umożliwia procedura PROFF.

RANDOM

TYP: funkcja
FORMAT: RANDOM
PRZYKŁADY:

```
WRITELN((RANDOM)*1000.0);
x := TRUNC(RANDOM)*11+10;
IF RANDOM<0.5 THEN WRITE('50% szansy')
```

DZIAŁANIE: Zwraca wartość losową typu REAL, która jest mniejsza od jeden i większa lub równa zero, czyli z przedziału <0,1). Funkcja ta nie posiada żadnych parametrów. W celu uzyskania innych liczb losowych funkcja RANDOM musi być uzupełniona dodatkowymi operacjami. W pierwszym przykładzie wynikiem będzie liczba z przedziału <0,1000), natomiast w drugim liczba całkowita od 10 do 20 (włącznie). Wzór ogólny dla uzyskania losowej liczby rzeczywistej o wartości z przedziału <a,b) jest następujący:

```
RANDOM*(b-a)+a;
```

SETCOLOR

TYP: procedura
FORMAT: SETCOLOR(<wyr_liczb>,<wyr_liczb>,<wyr_liczb>)
PRZYKŁADY:

```
SETCOLOR(2,0,0);
SETCOLOR(i,j,k);
```

DZIAŁANIE: Ustala barwę i stopień jasności wskazanego koloru. Wszystkie parametry muszą być typu INTEGER. Numer rejestru koloru jest określany przez pierwszy parametr, który może przyjmować wartości od 0 do 4. Drugi parametr oznacza barwą, zaś trzeci - stopień jasności koloru. Jasność może mieć wartości parzyste od 0 do 14. Wykorzystanie rejestrów koloru przez poszczególne elementy obrazu jest opisane w "Poradniku programisty Atari".

SOUND

TYP: procedura
FORMAT: SOUND(<wyraż_liczb>, <wyraż_liczb>,<wyraż_liczb>, <wyraż_liczb>)
PRZYKŁADY:

```
SOUND(0,200,10,10);
SOUND(k,10*l,m,15-n);
SOUND(k,0,0,0);
```

DZIAŁANIE: Ustawia generator dźwięku, którego numer jest pierwszym parametrem instrukcji (liczba z zakresu od 0 do 3). Jeśli wszystkie pozostałe parametry są zerami, to następuje wyłączenie generatora. Drugi parametr instrukcji może być z zakresu od 0 do 255 i określa okres dźwięku, a więc pośrednio jego częstotliwość. Parametr trzeci wybiera rodzaj zniekształceń tworzonego dźwięku, przy czym możliwe są tu liczby parzyste od 0 do 14 (nieparzyste wyłączają generator). Czysty ton uzyskuje się dla wartości 10 i 14. Ostatnim parametrem jest liczba z zakresu od 0 do 15, która ustala głośność generowanego dźwięku. Wszystkie parametry muszą być typu INTEGER.

SUBSTRING

TYP: funkcja
FORMAT: SUBSTRING (<zm_tabl >, <wyraż_liczb>, <wyraż_liczb>)
PRZYKŁADY:

```
WRITELN(SUBSTRING(nazwa,2,5));
imie := SUBSTRING(osoba,i,j);
```

DZIAŁANIE: Zwraca tablicę typu STRING będącą częścią tablicy, której nazwa zastała podana jako pierwszy parametr funkcji. Wielkość tego fragmentu określają dwa pozostałe parametry, które muszą być typu INTEGER. Pierwszy z nich wyznacza pierwszy znak fragmentu, a drugi jego długość. Procedura SUBSTRING wymaga wcześniejszego zadeklarowania stałej MAXSTRING oraz tablic, które powinny być typu

```
string = ARRAY [1..maxstring] OF CHAR.
```

1. 7. Meldunki błędów

Kompilator Kyan Pascal sygnalizuje błędy wykryte podczas kompilacji przez wskazanie miejsca wystąpienia i podanie skróconego opisu. Poza błędami w normalnej treści programu mogą także wystąpić błędy podczas asemblowania procedur w języku maszynowym dołączonych do programu. Również takie przypadki są sygnalizowane przez kompilator.

W czasie wykonywania programu mogą pojawić się błędy spowodowane użyciem niewłaściwych wartości lub wykonaniem instrukcji, które są w aktualnej chwili niepoprawne. Błędy te, zwane błędami wykonania, są sygnalizowane w taki sam sposób, lecz dopiero podczas działania programu.

Poniżej przedstawiane są możliwe błędy kompilacji i asemblacji wraz z tłumaczeniem.

SYNTAX ERROR - błąd składni
UNEXPECTED END OF INPUT - nieoczekiwany koniec danych
ARRAY DIMENSION EXPECTED - oczekiwany wymiar tablicy
"TO" OR "DOWNTO" EXPECTED - oczekiwane słowo "TO" lub "DOWNTO"
ORDINAL TYPE EXPECTED - oczekiwana definicja typu
":=" EXPECTED - oczekiwany znak ":="
":" EXPECTED — oczekiwany znak ":"
"," EXPECTED - oczekiwany znak ","
";" OR "END" EXPECTED - oczekiwany znak ";" lub słowo "END"
"DO" EXPECTED - oczekiwane słowo "DO"
" = " EXPECTED - oczekiwany znak "="
IDENTIFIER EXPECTED - oczekiwany identyfikator zmiennej
"[" EXPECTED - oczekiwany znak "["
CONSTANT EXPECTED - oczekiwana stała
"(" EXPECTED - oczekiwany znak "("
"OF" EXPECTED - oczekiwane słowo "OF"
TYPE IDENTIFIER EXPECTED - oczekiwany identyfikator typu
"." EXPECTED - oczekiwany znak "."
"PROGRAM" EXPECTED - oczekiwane słowo "PROGRAM"
"]" EXPECTED - oczekiwany znak "]"
")" EXPECTED - oczekiwany znak ")"
";" EXPECTED — oczekiwany znak ";"
".." EXPECTED - oczekiwany znak ".."
"THEN" EXPECTED - oczekiwane słowo "THEN"
UNSIGNED INTEGER EXPECTED - oczekiwana liczba całkowita bez znaku
FILE NAME EXPECTED - oczekiwana nazwa pliku
CANNOT OPEN FILE - nie można otworzyć pliku
ILLEGAL FILE NAME - błędna nazwa pliku
";" OR "UNTIL" EXPECTED - oczekiwany znak ";" lub słowo "UNTIL"
MISSING "END" STATEMENT(S) - brak instrukcji "END EXTRANEOUS "END"
STATEMENT(S) - zbędna instrukcja "END" ";" OR "CASE" EXPECTED - oczekiwany znak ";" lub słowo "CASE" EXPRESSION EXPECTED - oczekiwane wyrażenie ILL-FORMED BLOCK - niepoprawny format bloku
ERROR IN CONSTANT - błąd w definicji stałej WRONG TYPE - niewłaściwy typ
UNDECLARED IDENTIFIER - niezadeklarowany identyfikator NOT A RECORD - to nie rekord

NO SUCH FIELD - brak poszukiwanego pola
NOT A POINTER - to nie wskaźnik
ERROR IN LABEL - błędna etykieta
FORWARD ERROR - błąd dyrektywy FORWARD
NESTED INCLUDE - zagnieżdżone dołączenie pliku zewnętrznego
UNDECLARED TYPE ID - niezadeklarowany identyfikator typu
SHOULD BE TYPE ID - powinien być identyfikator typu
UNDEFINED IDENTIFIER - niezdefiniowana etykieta
ADDRESS MODE ERROR - błąd trybu adresowania
EQU NEEDS LABEL - dyrektywa EQU bez etykiety
DUPLICATE IDENTIFIER - powtórna definicja etykiety
SYMBOL TABLE FULL - brak miejsca w tablicy symboli
BRANCH OUT OF RANGE - skok poza dozwolony zakres

Poniżej przedstawione są możliwe błędy wykonania wraz z tłumaczeniem.

ERROR CODE xxx - błąd numer xxx
FILE NOT FOUND - plik nie odnaleziony
FILE NAME ERROR - błędna nazwa pliku
DISK # ERROR - błędny numer stacji dysków
NO DEVICE - urządzenie nie istnieje
WRONG FILE TYPE - niewłaściwy typ pliku
NO LIB - brak biblioteki (pliku "LIB")
TOO MANY OPEN FILES - zbyt dużo otwartych plików
RANGE ERROR - przekroczony dozwolony zakres
ARITHMETIC OVERFLOW - zbyt duża liczba lub dzielenie przez zero
READ PAST EOF - odczyt spoza końca pliku
BAD INDEX - błędny indeks

Podanie kodu błędu o wartości większej od 127 oznacza błąd, który wystąpił na poziomie systemu operacyjnego komputera. Znaczenie tych kodów jest opisane w książce "Poradnik programisty Atari".

Rozdział 2
DEEP BLUE C

Deep Blue C został napisany w roku 1982 przez amerykańskiego programistę Johna Howarda Palevicha jako rozwinięcie Small—C Compiler Rona Caina. Jest to implementacja języka C dla ośmiobitowych komputerów Atari. Zamieszczony dalej opis dotyczy wersji 1.1. Deep Blue C wczytywany jest z dyskietki, która zawiera kompilator, linker i bibliotekę. Na dyskietce znajdują się także pliki z dodatkowymi procedurami, które można dołączyć do własnego programu. Ponadto dostępna jest biblioteka funkcji matematycznych i trygonometrycznych MATHLIB napisana przez Franka Parisa (zwykle znajduje się ona na drugiej stronie dyskietki - patrz rozdział 2.7).

Deep Blue C jest implementacją języka wzorcowego C, lecz nie zawiera niektórych przewidzianych w nim struktur programowych. Do nauki programowania najlepszym podręcznikiem będzie więc opis języka wzorcowego pod tytułem "Język C" autorstwa B. W. Kernighana i D. M. Ritchie, a wydany przez WNT. Ponadto kursy programowania w C publikowane były w wielu czasopismach (m. in. "Bajtek", "IKS", "Młody Technik").

Kompilator i linker Deep Blue C odczytywane są z dyskietki przy pomocy funkcji BINARY LOAD DOS-u. Ponieważ Deep Blue C nie posiada własnego edytora, to wersje źródłowe programów trzeba pisać przy użyciu dowolnego edytora tekstów. Bardzo dobrze nadaje się do tego celu SpeedScript, gdyż umożliwia napisanie wszystkich znaków dostępnych w Atari. Programy napisane w Deep Blue C mogą być uruchamiane samodzielnie pod warunkiem, że biblioteka zawarta w pliku DBC.OBJ zostanie dołączona do programu (podczas scalania przez linker). Dotyczy to również programów zapisanych pod nazwą AUTORUN.SYS, które samoczynnie wczytują się i uruchamiają po włączeniu komputera.

Sposób wykorzystania pamięci komputera przez kompilator i linker Deep Blue C jest dla użytkownika zupełnie nieistotny. Ważne jest jedynie położenie w pamięci gotowego, skompilowanego i scalonego programu. Obszar od adresu 12288 ($3000) do 16383 ($3FFF) zajmuje biblioteka (DBC.OBJ). Dla programu oraz jego danych pozostaje miejsce od adresu 16384 ($4000) do 49151 ($BFFF), przy czym w górnej jego części znajduje się jeszcze pamięć obrazu. Jeśli została użyta biblioteka procedur matematycznych (MATHLIB), to zajmuje ona obszar od adresu 11712 ($2DC0) do 12287 ($2FFF). Położenie biblioteki MATHLIB można jednak zmienić korzystając z jej programu źródłowego zapisanego w pliku MATHLIB.ASM. Do dyspozycji użytkownika pozostaje ponadto szósta strona pamięci RAM (1536-1791 = $0600-$06FF) oraz obszar od szczytu DOS-u do początku biblioteki (12287 lub 11711).

Słowa kluczowe i operatory mogą być wprowadzane zarówno małymi, jak i dużymi literami - kompilator Deep Blue C nie rozróżnia ich bowiem między sobą. Przyjęte jest jednak wpisywanie wszystkich nazw instrukcji i zmiennych małymi literami, a stałych symbolicznych — dużymi literami.

2.1. Kompilator

W celu wczytania i uruchomienia kompilatora Deep Blue C należy wybrać funkcję BINARY LOAD DOS-u i podać nazwę pliku "CC.COM". Kompilator automatycznie uruchamia się po wczytaniu i wyświetla komunikat:

```
Deep Blue C Compiler version 1.1
(C)1982 John Howard Palevich

File to compile (or RETURN to exit)
```

Należy teraz wpisać nazwę pliku zawierającego program przeznaczony do kompilacji. Samo naciśnięcie klawisza <RETURN> powoduje opuszczenie kompilatora i powrót do DOS—u.

W Deep Blue C przyjęta jest konwencja określająca rozszerzenia nazw plików. Pliki zawierające programy źródłowe powinny mieć rozszerzenie ".C", a zawierające polecenia dla linkera - ".LNK". Programy skompilowane są zapisywane w plikach z rozszerzeniem ".CCC". Gotowe programy wynikowe otrzymują rozszerzenie ".COM". Ponadto dozwolone są pliki zawierające programy w kodzie maszynowym, które są oznaczone rozszerzeniem ".OBJ".

Przestrzeganie powyższej konwencji upraszcza pracę i pozwala na wyeliminowanie błędów. Na żądanie kompilatora wystarczy więc podanie samej nazwy, na przykład "PROGRAM", a kompilator odczyta i skompiluje plik o specyfikacji "Dl:PROGRAM.C". Kod wynikowy kompilacji zostanie zapisany w pliku "Dl:PROGRAM.CCC".

Podczas kompilacji nazwa aktualnie kompilowanej funkcji jest wyświetlana na ekranie. Jeżeli kompilacja przebiegła bez błędów, to pojawia się komunikat "No errors" i można przystąpić do kompilacji następnego programu lub przerwać pracę kompilatora.

W przypadku wystąpienia błędu w kompilowanym programie wyświetlany jest wiersz, w którym ten błąd został wykryty, oraz strzałka wskazująca miejsce jego wykrycia. Poniżej ukazuje się komunikat opisujący rodzaj znalezionego błędu. Opis błędów sygnalizowanych przez kompilator znajduje się w rozdziale 2.8. "Meldunki błędów".

Poprawienie błędu wymaga opuszczenia kompilatora i odczytania edytora, a następnie po dokonaniu zmiany ponownego wczytania kompilatora.

Skompilowanego programu nie można jeszcze uruchomić, gdyż wymaga on połączenia z biblioteką i ewentualnie z innymi programami. Do scalenia wszystkich części składowych w jedną pełną i samodzielną całość służy program zwany linkerem, czyli programem łączącym. Jest on opisany w następnym rozdziale.

2.2. Linker

W celu wczytania i uruchomienia linkera Deep Blue C należy wybrać funkcją BINARY LOAD DOS-u i podać nazwę pliku "CLNK.COM". Linker automatycznie uruchamia się po wczytaniu i wyświetla komunikat:

```
Deep Blue C Linker version 1.1
(C)1982 John Howard Palevich

Link program, Duplicate file or Quit
```

Wpisanie litery "Q" i naciśnięcie klawisza <RETURN> przerywa pracę linkera i powoduje powrót do DOS-u. Przez wpisania litery "D" i oddzielonej od niej spacją nazwy pliku można dokonać przeniesienia pliku z jednej dyskietki na drugą. Działanie tej funkcji jest identyczne jak działanie funkcji DOS-u DUPLICATE FILE, lecz przenoszony plik nie może zawierać więcej niż 5000 bajtów.

Właściwą operację łączenia programu wywołuje sie przez wpisanie litery "L" i oddzielonej od niej spacją nazwy pliku (bez rozszerzenia). Na przykład "L PROGRAM" poleca połączenie programu według informacji zawartych w pliku "Dl:PROGRAM.LNK". Gotowy program jest zapisywany w pliku z roszerzeniem ".COM", na przykład: "Dl:PROGRAM.COM".

Plik dla linkera jest zwykłym plikiem tekstowym, który zawiera nazwy wszystkich łączonych programów. Dla plików ze skompilowanymi programami podawanie rozszerzenia jest zbędne. Na przykład, w celu utworzenia gotowego programu o nazwie "PR0GRAM.COM" złożonego z programu "PROGRAM.CCC" i "AIO.CCC" należy w pliku dla linkera umieścić nazwy:

```
PROGRAM
AIO
DBC.OBJ
```

Umieszczenie nazwy pliku "DBC.OBJ" jest konieczne, gdyż znajduje się w nim biblioteka Deep Blue C. Jeżeli któryś z wymienionych programów znajduje się na dyskietce umieszczonej w stacji dysków o numerze innym niż 1, to jego nazwę trzeba poprzedzić nazwą urządzenia (np. "D8:DBC.OBJ"). W pliku dla linkera można ponadto wpisać nazwy innych plików z procedurami w języku maszynowym (w postaci wynikowej) wykonanymi przez dowolny asembler, np. MAC/65.

Jeżeli scalanie programu przebiegło bez błędów, to pojawia się komunikat "No errors" i można przystąpić do łączenia następnego programu, skopiować inny plik lub przerwać pracę linkera.

W przypadku napotkania w łączonym programie niezdefiniowanej nazwy wyświetlany jest komunikat "undefined labels:" (niezdefiniowane etykiety), a po nim lista tych nazw.

2.3. Technika programowania

Język C zastał skonstruowany w sposób umożliwiający łatwe przenoszenie utworzonych programów pomiędzy komputerami różnych typów. Niestety Deep Blue C jest bardzo prostym kompilatorem i nie posiada wszystkich możliwości dostępnych w standardowej wersji tego języka. Powoduje to konieczność dokonania dodatkowych przeróbek w programach pisanych dla komputerów innych niż Atari XL/XE. Elementami niedostępnymi w implementacji Deep Blue C są:

1) struktury i unie danych;
2) tablice wielowymiarowe;
3) liczby rzeczywiste (zmiennoprzecinkowe);
4) funkcje zwracające wartości inne niż całkowite;
5) operator jednoargumentowy "sizeof";
6) operator dwuargumentowy "typecasting".

Elementy wymienione w punktach 3 i 4 można jednakże uzyskać przy użyciu biblioteki matematycznej (MATHLIB).

Program w języku C nie musi być budowany według dokładnie ustalonego schematu. Elastyczność tego języka jest bardzo duża i kompilator potrafi przetłumaczyć nawet bardzo dziwne pomysły programisty. Poszczególne elementy programu muszą być jednak zgodne z regułami składni języka.

Program w C składa się zwykle z dużej liczby różnorodnych funkcji. Ich nazwy i kolejność umieszczenia w programie są dowolne, lecz definicja żadnej funkcji nie może znajdować się wewnątrz innej funkcji. Jedynym wyjątkiem jest funkcja "main". Musi być ona zawarta w programie (w dowolnym miejscu) i jest zawsze pierwszą wykonywaną przez program funkcją. Poszczególne wiersze programu nie są numerowane. Instrukcje są oddzielane od siebie średnikami, a w jednym wierszu można umieścić ich dowolną liczbę tak jednak, aby długość wiersza nie przekraczała 79 znaków. Podział programu na wiersze nie ma znaczenia dla kompilatora, lecz służy wyłącznie do poprawienia czytelności i zrozumiałości programu.

Struktura programu w języku C jest bardzo prosta. Każdy program może składać się z następujących trzech części: bloku dyrektyw, bloku deklaracji zmiennych i bloku deklaracji funkcji. Wymienione elementy muszą być umieszczone w programie w podanej wyżej kolejności. Konieczny jest tylko ostatni blok (deklaracje funkcji), a pozostałe są stosowane w razie potrzeby i mogą być pominięte.

Jedna i tylko jedna funkcja musi mieć nazwę "main". Od niej rozpoczyna się wykonywanie programu. Nazwy pozostałych funkcji są dowolne. Po nazwie należy umieścić w nawiasie okrągłym nazwy parametrów przekazywanych do funkcji. Jeśli funkcja nie wymaga żadnych parametrów, to po jej nazwie umieszcza się tylko pusty nawias "()". Jeśli parametry występują, to po nazwie trzeba wpisać ich deklaracje. Pozostała treść funkcji musi być ograniczona znakami "$("i "$)" zastępującymi nawiasy klamrowe,

które nie występują w zestawie znaków Atari. Między nimi znajdują się deklaracje zmiennych użytych w funkcji i instrukcje programu, w tym również wywołania funkcji określonych w innych miejscach programu. Wszystkie zmienne stosowane w funkcji muszą być zadeklarowane przed użyciem. Dlatego też powinna się wpisywać deklaracje przed pierwszą instrukcją w funkcji.

Instrukcje w C mogą być instrukcjami prostymi lub złożonymi. Instrukcje proste są to polecenia wykonania pojedynczych operacji, a ich opisy znajdują się w słowniku. Instrukcje złożone zawierają kilka instrukcji prostych oddzielonych od siebie średnikami i są ograniczone znakami "$(" i "$)".

Komentarze mogą być umieszczane w dowolnym miejscu programu (także między nazwą funkcji i deklaracjami jej parametrów. Komentarze są ograniczone symbolami "/*" i "*/", które muszą obejmować tekst komentarza. Niedozwolone jest zagnieżdżanie komentarzy (umieszczanie jednego komentarza w drugim).

Poniżej znajduje się przykład prostego programu w języku C, który pięciokrotnie wyświetla napis. Uwidocznione są tu podstawowe zasady składni C.

```
main() /* przykład */ $(
  int i;
  for (i=1;i<=5;i=i+1)
   printf("S.O.E.T.O.\n");
$)
```

2.4. Wartości

W Deep Blue C występują stałe i zmienne różnych typów. Ich deklaracje powinny być umieszczone w początkowej części programu lub funkcji. W dalszej części tego rozdziału opisane są wszystkie rodzaje wartości dostępne w Deep Blue C oraz niektóre zależności miedzy nimi.

Stałe

Stałe są wartościami definiowanymi przez programistę i występującymi najczęściej w postaci jawnej. Możliwe jest także użycie stałych w postaci nazw symbolicznych (stałych symbolicznych), które symbolizują wartości tych stałych w całym programie i są zamieniane na wartości w procesie kompilacji. Dzięki temu znacznie łatwiejsze jest modyfikowanie programu w przypadku, gdy konieczna jest zmiana tych wartości. Ponadto użycie stałych symbolicznych znacznie zwiększa czytelność programu.

W Deep Blue C możliwe jest użycie kilku rodzajów stałych. Opis typów tych stałych znajduje się poniżej.

Liczby dziesiętne są to liczby całkowite z zakresu od -32768 do 32767. Pierwsza cyfra liczby dziesiętnej musi być różna od zera.

Liczby ósemkowe są liczbami całkowitymi zapisanymi w systemie ósemkowym i oznaczane są przez umieszczenie zera jako pierwszej cyfry. Przyjmują one wartości z zakresu od —0100000 do 077777.

Liczby szesnastkowe mają także wartości całkowite i są oznaczone przez umieszczenie na początku znaków "0x". Zakres ich wartości wynosi od 0x0 do 0xffff.

Stałe znakowe są ujętymi w apostrofy znakami kodu ASCII (np. : 'a', '3', '%') i reprezentują wartości liczbowe tego kodu (od 0 do 255).

Stałe tekstowe są to dowolne ciągi znaków ujęte w cudzysłowy. Ciąg taki nie może zawierać znaku o kadzie zero (serce), gdyż jest on używany do rozpoznawania końca ciągu. Znakiem o specjalnym znaczeniu jest ponadto znak "\", który służy do uzyskiwania innych znaków:

\f — czyszczenie ekranu (dear screen), \g - dźwięk brzęczyka (buzzer), \b — usunięcie znaku (backspace), \n - koniec wiersza (End Of Line), \r — usunięcie wiersza (delete iine), W - znak "\" (backslash), V - apostrof, \" — cudzysłów, \t — tabulacja, \### - dowolny znak ATASCII o kodzie określonym przez liczbę ósemkową (###) zawierającą od jednej do trzech cyfr.

Zmienne

Wartości zmienne, które są używane w programie, muszą być przed użyciem zadeklarowane. Deklaracja zmiennej określa jej typ i nazwę (identyfikator) oraz ewentualnie wartość początkową. W Deep Blue C dozwolone są zmienne typu całkowitego (int), znakowego (char) i wskaźnikowego (pointer) .

Nazwa zmiennej może składać się z liter, cyfr i znaku podkreślenia (_). Pierwszym znakiem nazwy musi być litera. Deep Blue C rozróżnia duże i małe litery, zwyczajowo przyjęte jest pisanie nazw zmiennych małymi literami, a stałych symbolicznych dużymi. Nazwa zmiennej może mieć dowolną długość, lecz kompilator rozróżnia tylko 8 pierwszych znaków.

W języku C ściśle przestrzegana jest lokalność zmiennych. Każda zmienna jest określona tylko w tej funkcji, w której została zdefiniowana. Zmiennymi globalnymi są więc zmienne zdefiniowane poza wszystkimi definicjami funkcji. Zmienne zdefiniowane wewnątrz funkcji są lokalne i określone tylko wewnątrz tej funkcji. Użycie zmiennej wewnątrz innej funkcji (wywoływanej z poziomu definicji zmiennej) wymaga zadeklarowania typu zmiennej w tej funkcji przy użyciu słowa "extern".

Tablice

Tablicą jest zmienna zawierająca kilka elementów danego typu. Dozwolone są tablice liczbowe (typu int) oraz znakowe (typu char). Wielkość tablicy jest określana przez wartości indeksowe, które mogą mieć dowolną wartość całkowitą. Rzeczywista wielkość tablic zależy jednak od wielkości pozostałego do dyspozycji obszaru pamięci.

Definicja tablicy jest bardzo zbliżona do definicji zmiennej prostej: składa się z nazwy typu, nazwy tablicy i jej rozmiaru ujętego w nawiasy kwadratowe. Natomiast deklaracja tablicy zewnętrznej (zdefiniowanej poza funkcją) zawiera tylko nawiasy kwadratowe bez umieszczonej w nich wartości.

Elementy tablicy są zawsze numerowane od 0, a rozmiar tablicy określa liczbę elementów. W celu wskazania elementu tablicy należy podać nazwę tej tablicy, a po niej wartość indeksu ujętą w nawias kwadratowy. Na przykład tablica zdefiniowana jako "tab[10]" zawiera elementy od "tab[0]" do "tab[9].

Wskaźniki

Wskaźnik jest zmienną, która zawiera adres innej zmiennej. Znaczenie zmiennych tego typu jest w C podobne do występującego w Pascalu. Wskaźniki nie są odrębnym typem, gdyż przyjmują wartości odpowiadające liczbom całkowitym bez znaku (od 0 do 65535). Są one natomiast określane poprzez typ wskazywany.

Definicja wskaźnika składa się z identyfikatora typu wskazywanego oraz nazwy wskaźnika poprzedzonej gwiazdką (*). Szczególnie użyteczne jest stosowanie wskaźników do tablic.

Zwiększenie wartości wskaźnika o jeden powoduje w tym przypadku wskazanie następnego elementu tablicy, niezależnie od faktycznego rozmiaru elementu tablicy w pamięci. Na przykład, jeśli p:< wskazuje na początek tablicy x, czyli *px oznacza x[0], to *(px+1) oznacza x[1]. Ponieważ nazwa tablicy reprezentuje położenie jej pierwszego elementu, to wynika z tego równość px==&x[0]==x. Podobnie zamiast odwołania do elementu tablicy x[i] można użyć odwołania *(x+i). Prawdziwy jest także zapis odwrotny: jeśli px wskazuje na tablicę x, to px[i]==*(px+i).

W pokazanych wyżej przypadkach istnieje ważna różnica pomiędzy wskaźnikiem a nazwą tablicy. Nazwa tablicy jest stała, więc nie można jej przypisywać wartości, natomiast wskaźnik jest zmienną, której wartość można dowolnie modyfikować.

Wyrażenia

Wyrażenie jest konstrukcją języka C oznaczającą wartość pewnego typu. Określenie tej wartości odbywa się przez wykonanie operacji wskazanych przez znajdujące się w wyrażeniu operatory. Wyrażenia są zbudowane ze stałych, zmiennych, operatorów, nazw funkcji i nawiasów okrągłych.

Przy konstruowaniu wyrażeń w C można dowolnie mieszać typy "int" i "char". Uzyskany wynik będzie w takim przypadku typu zdefiniowanego dla zmiennej wynikowej. Zmienna typu "char" jest zawsze bez znaku, więc przy zamianie na typ znakowy nie można otrzymać wartości ujemnych.

Język C oferuje zestaw operatorów stosowanych w wyrażeniach znacznie bogatszy niż inne języki programowania. Ponadto wiele z nich może występować jako operatory jednoargumentowe oraz jako dwuargumentowe. Z tych względów konieczne jest bliższe ich opisanie.

Operatory jednoargumentowe

Wszystkie operatory jednoargumentowe mają ten sam priorytet i są wykonywane jako pierwsze przed innymi działaniami, lecz po obliczeniu wyrażeń umieszczonych w nawiasach okrągłych i kwadratowych. Operatory te są prawostronnie łączne, czyli w jednym wyrażeniu są wykonywane od prawej strony do lewej. Operatorami jednoarguraentowymi w Deep Blue C są:

```
+  : znak liczby
-  : znak liczby
*  : odwołanie pośrednie
&  : adres obiektu
++ : zwiększenie
-- : zmniejszenie
!  : negacja logiczna $-
: negacja bitowa
      (w standardzie C użyty jest znak "~")
```

Operatory "+" i "-" określają znak argumentu, przy czym "+" nie zmienia znaku, zaś "-" zmienia znak argumentu na przeciwny.

Operator "#" zwraca wartość obiektu, którego wskaźnikiem jest argument operatora.

Operator "&" zwraca wartość wskaźnika do obiektu, który jest argumentem.

Operatory "++" i "--" powodują odpowiednio zwiększenie lub zmniejszenia o jeden wartości argumentu. Jeśli argumentem jest wskaźnik, to zmiana wartości argumentu odpowiada długości elementu wskazywanego. Oba te operatory mogą być stosowane przed i po argumencie. Zastosowanie operatora przed argumentem zwiększa jego wartość przed użyciem, a zastosowanie po argumencie zwiększa wartość po użyciu. Na przykład, dla y równego 5 przypisanie x=++y nadaje x wartość 6, a przypisanie x=y++ nadaje x wartość 5.

Operator "!" dla argumentu zerowego daje w wyniku 1, a dla wszystkich pozostałych wartości argumentu wynikiem jest zero. Typem wyniku jest zawsze int.

Operator "$-" zwraca dopełnienie jedynkowe argumentu, czyli wykonuje operację negacji dla poszczególnych bitów argumentu.

Operatory wieloargumentowe Operatorami

wieloargumentowymi w Deep Blue C są:

```
  *   :       mnożenie
  /   :  dzielenie całkowite
  "%             :     reszta z dzielenia
  +   :       dodawanie
  -   :      odejmowanie
<< >> :operatory przesunięcia
< <=  |
> >=  |  operatory relacji
== !=  |
 & ^ |: operatory bitowe
&& || : operatory logiczne
  ?:  : operator warunkowy
  =   ; operator przypisania
  ,   : operator przecinkowy
```

Operator warunkowy wymaga trzech argumentów, natomiast wszystkie pozostałe są dwuargumentowe.

Operatory arytmetyczne mają działanie zgodne z ogólnie przyjętym w matematyce i nie wymagają bliższego opisu.

Operatory przesunięcia powodują przesunięcie bitów lewego argumentu o liczbę miejsc określoną przez prawy argument, który może być liczbą z zakresu od 1 do 7. Opróżnione miejsca są wypełniane zerami. Operator "<<" przesuwa bity w lewo, zaś operator ">>" w prawo.

Operatory relacji służą do porównań dwóch argumentów i dają wynik 1, jeśli relacja jest prawdziwa, a zero w przeciwnym przypadku. W odróżnieniu od innych języków operator nierówności jest zapisywany jako "!=", a operator równości jako "==" (dla odróżnienia od operatora przypisania " = ").

Operatory bitowe wykonują operacje logiczne na poszczególnych bitach podanych argumentów. Są to operacje

koniunkcji (& - AND) , alternatywy (| - OR) i różnicy symetrycznej (^ — XOR) .

Operatory logiczne wykonują operacje logiczne (koniunkcji — && i alternatywy - ||) na całych wartościach argumentów. Jeśli operacja jest prawdziwa, to wynikiem jest 1, a w przeciwnym przypadku zero.

Operator warunkowy "?:" jest operatorem trójargumentowym. Jeśli wartość pierwszego argumentu jest różna od zera, to wynikiem jest wartość , drugiego argumentu, zaś w przeciwnym przypadku wartość trzeciego argumentu. Składnia operacji jest przy tym następująca:

<argument_l> ? <argument_2> : <argument_3>

Operator przecinkowy (,) rozdziela parę wyrażeń, które obliczane są od lewej do prawej. Wynikiem jest wartość prawego argumentu, a wartość lewego argumentu przepada. Na przykład, wartością wyrażenia x=2,x+5 jest 7.

Operator przypisania (=) przypisuje lewemu argumentowi wartość prawego wyrażenia. W języku C istnieje możliwość skróconego zapisu operatorów przypisania wykorzystujących po prawej stronie poprzednią wartość lewego argumentu. Zamiast pełnego zapisu w postaci:

<zmienna> = <zmienna> <oper> <wyrażenie>

stosuje się zapis w formie:

<zmienna> <oper>= <wyrażenie>

Na przykład, wyrażenie x=x*(y+1) jest równoważne wyrażeniu x*=y+1. W skróconym formacie przypisania dozwolone jest stosowanie operatorów arytmetycznych. bitowych i operatorów przesunięcia.

Priorytet operatorów, czyli kolejność wykonywania określanych przez nie operacji, jest dla operatorów wieloargumentowych przedstawiony w poniższej tabeli. Operatory o jednakowym priorytecie są umieszczone w jednym wierszu tabeli, a w wyrażeniu są realizowane od lewej do prawej. Wyjątkiem są operatory "=", "+=", "-=" i "?:", które są wykonywane od prawej do lewej.

```
* / %
 + —
<< >>
< <= > >=
== !=
  &
  ^

  |
 &&
   ||
 ?:
 = += -=

  i
```

Ten rozdział zawiera kompletny słownik słów kluczowych jeżyka Deep Blue C w kolejności alfabetycznej. Podana jest tu składnia każdego słowa, sposoby jego wykorzystania oraz liczne przykłady. Ponadto na początku zamieszczony jest spis tych słów.

W słowniku przyjęta została następująca kolejność opisu: nazwa, typ, składnia, przykłady i działanie. Typ określa rodzaj słowa kluczowego: instrukcja, definicja typu lub dyrektywa. Składnia opisuje przy użyciu symboli podanych we wprowadzeniu dozwolone sposoby zapisu słowa kluczowego. Pozostałe punkty nie wymagają objaśnienia.

SŁOWA KLUCZOWE DEEP BLUE C

asm	define	include
break	do	int
case	else	return
char	extern	switch
continue	for	while
default	if	/* */

asm

TYP: instrukcja
FORMAT: asm <adres>;
PRZYKŁADY:

```
asm 1536;
asm 0x600;
```

DZIAŁANIE: Powoduje skok do procedury w języku maszynowym rozpoczynającej się od adresu będącego parametrem instrukcji. Instrukcja "asm" nie przekazuje do procedury żadnych parametrów. Jeśli procedura kończy się rozkazem RTS, to następuje powrót do instrukcji następującej po "asm".

break

TYP: instrukcja
FORMAT: break;
PRZYKŁADY:

```
for (i=0;i<100;i++)
$(
  if (i>9)
    break;
  printf("SOET\n"); $)

switch(day)
$(
  case '7':
  printf ("wolny\n");
break; default:
     printf("roboczy\n");
     break;
 $)
```

DZIAŁANIE: Powoduje natychmiastowe opuszczenie struktury, w której się znajduje i przejście da wykonywania instrukcji następującej bezpośrednio po przerwanej. Służy do przerywania pętli "for", "while" i "do" oraz do kończenia wariantów instrukcji "switch".

case

TYP: instrukcja
FORMAT: case <stała>:[case <stała>:]... <instrukcje>;
PRZYKŁADY:

```
switch(day)
$(
  case '1':
  case '2':
  case '3':
  case '4':
  case '5':
  case '6':
    printf("roboczy\n");
```

```
      break;
    case '7':
      printf ("wolny\n");
      break
    ;
    default:
      printf("pomylka\g\n");
      break;
  $)
```

DZIAŁANIE: Wskazuje w instrukcji "switch" grupę instrukcji, które są wykonywane, gdy warunek ma wartość równą podanej stałej. Dozwolone są dowolne instrukcje, w tym również puste i złożone, jednak ostatnią instrukcją w grupie musi być "break", "return" lub "continue". Stała w instrukcji "case" musi być typu całkowitego. Niedozwolone jest dwukrotne użycie takiej samej stałej w jednej instrukcji "switch". Patrz też opis instrukcji "default" i "switch".

char

TYP: definicja typu
FORMAT: char <nazwa>[[<wyraż>]|=<wyraż>][,<nazwa>...];
PRZYKŁADY:

```
char znak; char *wskaznik;
char x1,x2,x3;
char alfa[100],beta[20];
char eol='\n',clear='\f';
```

DZIAŁANIE: Definiuje podane nazwy jako nazwy zmiennych znakowych. Nazwa może być dowolnym, poprawnym identyfikatorem. Definiowanej zmiennej można jednocześnie nadać wartość przy użyciu operatora przypisania. Umieszczenie po nazwie wartości w nawiasie kwadratowym jest natomiast definicją tablicy znakowej. W przypadku definiowania wskaźników definiowany typ odnosi się do wartości wskazywanych, a nie do samego wskaźnika.

continue

TYP: instrukcja
FORMAT: continue;
PRZYKŁAD:

```
for(i=0;i<n;i++)
$(
   if(a[i]<0)
     continue;
   x=a[i];
   silnia(x);
$)
```

DZIAŁANIE: Powoduje przerwanie aktualnie wykonywanej pętli "for", "do" lub "while" i realizacje tej pętli od początku następnego kroku. W instrukcji "for" powoduje to przejście do części modyfikującej licznik, zaś w instrukcjach "do" i "while" powoduje natychmiastowe sprawdzenie warunku zatrzymania.

default

TYP: instrukcja
FORMAT: default: <instrukcje>;
PRZYKŁADY:

```
switch(day)
$(
  case '7':
    printf("wolny\n");
    break;
  default:
printf <"roboczy\n");
break; $)
```

DZIAŁANIE: Wskazuje w instrukcji "switch" grupę instrukcji, które są wykonywane, gdy warunek nie ma wartości równej żadnej ze stałych umieszczonych w instrukcjach "case". Dozwolone są dowolne instrukcje, w tym również puste i złożone, jednak ostatnią instrukcją w grupie musi być "break", "return" 'lub "continue". W instrukcji "switch" może znajdować się tylko jedna instrukcja "default", ale można ją też pominąć całkowicie. Patrz również opis instrukcji "case" i "switch".

define

TYP: dyrektywa
FORMAT: #define <identyfikator> <(stała>
PRZYKŁADY:

```
#define EOL 155
#define FIRMA "S.O.E.T.O"
#define RAMTOP 0x6a
#define CLS  '\f'
```

DZIAŁANIE: Powoduje zastąpienie podczas kompilacji programu wszystkich napotkanych dalej wystąpień identyfikatora przez określoną dla niego wartość stałą. Zastosowanie tej dyrektywy znacznie ułatwia późniejsze modyfikowanie programów dzięki wyeliminowaniu konieczności przeszukiwania całego programu w celu zmiany wprowadzonej wartości. Po dyrektywie "define" nie wpisuje się średnika (patrz przykłady).

do

TYP: instrukcja
FORMAT: do <instrukcja> while(<wyrażenie>);
PRZYKŁADY:

```
do x-=2
while(x>0);

do
$(
  arg%=k;
  k/=10;
$) while(k/10>0);
```

DZIAŁANIE: Instrukcja "do" rozpoczyna pętlę do/while.

Wszystkie instrukcje zawarte pomiędzy słowami "do" i "while" są wykonywane, dopóki <wyrażenie> ma wartość różną od zera. Wartość zero powoduje przerwanie pętli i wykonanie instrukcji następującej po "while". Warunek jest sprawdzany na końcu pętli, więc zawarte w niej instrukcje muszą być wykonane co najmniej jeden raz.

Else

TYP: instrukcja
FORMAT: patrz opis instrukcji "if"
DZIAŁANIE: Słowo "else" stanowi integralną część instrukcji "if". Następuje po nim instrukcja, wykonywana gdy warunek po "if" jest fałszywy. Może to być także instrukcja złożona. Część instrukcji "if" rozpoczynająca się od słowa "else" może zostać pominięta (patrz też opis "if").

extern

TYP: definicja typu
FORMAT: extern <definicja_zmiennej>;
PRZYKŁADY:

```
extern char znak;
extern int x1,x2,x3;
extern int alfa[],beta[];
```

DZIAŁANIE: Słowo "extern" określa następującą po nim definicję zmiennej jako deklarację zmiennej globalnej wewnątrz funkcji, w której umieszczona jest deklaracja. Taka konstrukcja umożliwia dostęp funkcji do zmiennej zdefiniowanej poza funkcją, nie tworzy ona jednak zmiennych i nie przydziela im obszaru pamięci. Przy deklarowaniu tablic oznacza się je tylko nawiasami kwadratowymi bez umieszczonego w nich rozmiaru tablicy. Najczęściej definicja ta jest stosowana przy umieszczaniu programu źródłowego w kilku różnych plikach.

for

TYP: instrukcja
FORMAT: for([<wyraż_1>];[<wyraż_2>];[<wyraż_3>]) <instr>;
PRZYKŁADY:

```
for(i=1;i<11;i++) silnia(i);
for(i=10;i>0;i--) silnia(i);
for(l=2*x;l<8*x;l+=5) oblicz (x) ;
for(;;) printf("petla bez konca\n");
for(;i<11;i++) funkcja(i);
for(i=1;i<11;) funkcja(i);
for(i=1;;i++) funkcja(i);
```

DZIAŁANIE: Tworzy pętlę programu zawierającą instrukcję wykonywaną kilka razy. Liczba przejść pętli oraz wartości licznika są ustalane przez parametry instrukcji "for". Kolejność działań jest następująca:

1. obliczenie <wyraż_1>,
2. obliczenie <wyraż_2>,

3.jeśli <wyraż_2> jest równe zero, to koniec,
4.wykonanie <instr>,
5.obliczenie <wyraż_3>,
6.przejście do punktu 2.

Z powyższego schematu wynika, że instrukcja "for" jest równoważna instrukcji "while" zapisanej w postaci:

```
<wyraż_1>
while(<wyraż_2>)
$(
  <instr>;
  <wyraż_3>;
$)
```

Z takiego przedstawienia wynika możliwość opuszczenia dowolnego wyrażenia w instrukcji "for". Niezbędne jest jednak pozostawienie rozdzielających średników. Gdy zostanie opuszczone <wyraż_2>, to przyjmowana jest wartość 1 i instrukcja uzyskuje postać:

```
for(<wyraż_1>;1;<wyraż_3>) <instrukcja>;
```

przez co staje się pętlą bez końca, którą można przerwać tylko przy użyciu instrukcji "break" lub "return".

Znajdująca się w pętli "for" instrukcja może być dowolną instrukcją, w tym również instrukcją pustą lub złożoną.

if

TYP: instrukcja
FORMAT: if <wyraż> <instrukcja_1>; [else <instrukcja_2>];
PRZYKŁADY:

```
if(x) oblicz(x);
if(y>0) x=y; else x=-y;
if(a=b) printf("A = B\n");
if(a=b*b) printf("A = B^2\n");
```

DZIAŁANIE: Pozwala na wybranie sposobu postępowania w zależności od wartości wyrażenia (warunku). Jeżeli warunek ma wartość różną od zera, to wykonywana jest <instrukcja_1>. W przeciwnym przypadku wykonywana jest <instrukcja_2> znajdująca się po słowie "else". Jeśli, w instrukcji "if" nie ma słowa "else", to realizacja programu jest kontynuowana od następnej instrukcji. Instrukcja wykonywana przez "if" może być dowolna, w szczególności dozwolone są instrukcje złażone i inne instrukcje strukturalne.

include

TYP: dyrektywa
FORMAT: #include <"specyfi kacja_pliku">
lub #include <<specyfikacja_pliku>>
PRZYKŁADY:

```
#include "D:PLOT.C"
#include <D8:SOUND.C>
```

DZIAŁANIE: Dołącza do kompilowanego programu procedury zawarte w pliku dyskowym wskazanym w dyrektywie. Umożliwia to stworzenie biblioteki procedur i uproszczenie pisanych później programów. Ponadto dzięki tej dyrektywie możliwe jest kompilowanie programów, które w całości nie mogą być skompilowane ze względu na znaczną objętość. Dyrektywy "include" nie mogą być zagnieżdżane, to znaczy w pliku dołączanym przez "include" nie można umieszczać takiej dyrektywy.

int

TYP: definicja typu
FORMAT: int <nazwa>[[<wyraż>]|=<wyraż>][,<nazwa>...];
PRZYKŁADY:

```
int liczba;
int *wskaznik;
int x1,x2,x3;
int alfa[100],beta[20];
int start=0,end=100;
```

DZIAŁANIE: Definiuje podane nazwy jako nazwy zmiennych całkowitych. Nazwa może być dowolnym, poprawnym identyfikatorem. Definiowanej zmiennej można jednocześnie nadać wartość przy użyciu operatora przypisania. Umieszczenie po nazwie wartości w nawiasie kwadratowym jest natomiast definicją tablicy liczbowej. W przypadku definiowania wskaźników definiowany typ odnosi się do wartości wskazywanych, a nie do samego wskaźnika.

return

TYP: instrukcja
FORMAT: return [<wyrażenie>];
PRZYKŁADY:

```
return;
return i;
return (c<'A')||(c>'Z');
```

DZIAŁANIE: Przerywa wykonywanie funkcji, w której się znajduje, i powoduje powrót do miejsca wywołania. Wartość wyrażenia znajdującego się w instrukcji "return" jest przekazywana do miejsca wywołania funkcji. Jeśli w instrukcji "return" nie ma żadnego wyrażenia lub funkcja wraca do miejsca wywołania po osiągnięciu kończącego ją nawiasu "$)" (bez użycia "return"), to zwracana wartość jest nieokreślona i może spowodować w programie błędy.

switch
TYP: instrukcja
FORMAT: switch(<wyrażenie>) $(

```
          [[case <stala>:]...]
        [case <stała>:] [<instrukcja>;]
        [default: <instrukcja>;] $)
```

PRZYKŁAD:

```
switch(digit)
$(
  case '2':
  case '4':
  case '6':
  case '8':
    printf ("parzysta\n");break;
  case 'O':
    printf("zero\n");break;
  default:
    printf("nieparzysta\n");break;
$)
```

DZIAŁANIE: Powoduje wykonanie instrukcji, która występuje po słowie "case" ze stałą o wartości równej wartości wyrażenia lub po słowie "default". Wartości stałych nie mogą się powtarzać. Po wykonaniu instrukcji rozpatrywane są następne przypadki, należy więc w każdym przypadku zakończyć grupę instrukcji przez "break", "continue" lub "return". Dotyczy to również ostatniego wariantu instrukcji "switch" - musi ona być zawsze przerwana jedną z wymienionych wyżej instrukcji.

while

TYP: instrukcja
FORMAT: while(<wyrażenie>) <instrukcja>;
PRZYKŁADY:

```
while(a<100) oblicz(a);
while(x<100) x+=1;
while(--lim>0&&(c=getchar())!=EOF&&c!='\n')
  s[i++]=c;
```

DZIAŁANIE: Instrukcja "while" nakazuje wykonywanie zawartej w niej instrukcji, dopóki <wyrażenie> ma wartość różną od zera. Wartość zero powoduje przerwanie pętli i wykonanie następnej instrukcji. Warunek ten jest sprawdzany na początku pętli, więc zawarte w niej instrukcje mogą wcale nie być wykonane. W instrukcji "while" może być umieszczona dowolna instrukcja, w tym również złożona lub pusta.

Ponadto słowo "while" służy do oznaczenia warunku wykonywania pętli "do" (patrz opis instrukcji "do").

/* */

TYP: instrukcja
FORMAT: /* [<dowolny ciąg znaków>] */
PRZYKŁADY:

```
/* ** TO JEST KOMENTARZ ** */
oblicz(x); /* obliczenie silni */
```

DZIAŁANIE: Znaki te ograniczają komentarz, który jest podczas kompilacji programu całkowicie ignorowany. Po jego napotkaniu dalsza kompilacja programu jest wykonywana od następnej instrukcji. W komentarzu dozwolone są wszystkie znaki, z wyjątkiem znaków "/*" i "*/".

2.6. Biblioteka standardowa

Język C charakteryzuje się ogromną prostotą między innymi z powodu pominięcia w nim całkowicie funkcji zależnych od sprzętu. Funkcje te zawarte są w dodatkowych plikach tworzących bibliotekę. Nazwy funkcji są zbliżone we wszystkich implementacjach języka, lecz ich realizacja jest całkowicie odmienna. Deep Blue C posiada cztery dodatkowe pliki biblioteczne: AIO.C, GRAPHICS.C, PMG.C i PRINTF.C. Pozwalają one na pełne wykorzystanie możliwości sprzętowych komputera.

Ten rozdział zawiera kompletny słownik funkcji znajdujących się w plikach bibliotecznych Deep Blue C opisanych w kolejności alfabetycznej. Podana jest tu nazwa każdej funkcji, jej definicja, sposoby wykorzystania oraz liczne przykłady. Ponadto na początku zamieszczony jest spis tych funkcji.

W słowniku przyjęta została następująca kolejność opisu: nazwa, plik, typ, definicja, przykłady i działanie. Typ określa rodzaj funkcji. Definicja opisuje sposób wywołania i deklaracje parametrów wywołania. Pozostałe punkty nie wymagają objaśnienia.

UWAGI:

1.Wszystkie wartości zwracane przez funkcje zawarte w bibliotece języka Deep Blue C są typu całkowitego (int).

2.W celu wykorzystania funkcji zawartych w plikach GRAPHICS.C, PMG.C i PRINTF.C niezbędne jest dołączenie do programu także pliku AIO.C. Praktycznie niemożliwe jest napisanie programu, który nie korzystałby z funkcji zawartych w AIO.C.

3.Użycie w programie dowolnej funkcji z pliku PMG.C wymaga uprzedniego zainicjowania grafiki graczy i pocisków przez wywołanie (najlepiej na początku programu) funkcji pmcinit.

4.Niektóre funkcje zwracają ujemny kod błędu. Oznacza to, że standardowy kod błędu sygnalizowany przez system operacyjny Atari ma znak zmieniony na przeciwny. Na przykład, wystąpienie błędu IOCB NOT OPEN (kod 133) powoduje zwrócenie przez funkcję wartości -133.

FUNKCJE W PLIKU AIO.C

cclose	cputc	hval	strcpy
cgetc	cputs	move	strlen
ciov	dpeek	normalize	tolower
clear	dpoke	open	toupper
close	find	peek	usr
copen	getchar	poke	val
cprints	gets	putchar	

FUNKCJE W PLIKU GRAPHICS.C

color	hstick	position	sound
drawto	locate	ptrig	stick
fill	paddle	rnd	strig
graphics	plot	setcolor	vstick

FUNKCJE W PLIKU PMG.C

chget	hitpʏpf	pmcflush	pmgraphics
choget	hitpʏpl	pmcinit	pmwidth
chput	pladdr	pmclear	
hitclear	plroove	pmcolor	

FUNKCJE W PLIKU PRINTF.C

fprintf	printf

cclose

PLIK: AIO.C
TYP: funkcja I/O
DEFINICJA: cclose(i)
int i;
PRZYKŁADY:

```
err = cclose(channel);
status = cclose(1);
cclose(file);
```

DZIAŁANIE: Zamyka plik, który został otwarty przy pomocy funkcji copen. Argumentem funkcji musi być wartość zwrócona przez copen. Po prawidłowym zamknięciu pliku funkcja cclose zwraca wartość 1, a w przeciwnym przypadku ujemną wartość kodu błędu, który wystąpił.

cgetc

PLIK: AIO.C TYP: funkcja
I/O DEFINICJA:
cgetc(unit) int unit;
PRZYKŁADY:

```
x = cgetc(1);
key = cgetc(iocb);
putchar(cgetc(file));
```

DZIAŁANIE: Zwraca wartość bajtu danych (z zakresu od 0 do 255) pobranego z kanału o numerze, który jest argumentem funkcji. W przypadku wystąpienia błędu wynikiem jest ujemny kod tego błędu. Kanał transmisji musi być uprzednio otwarty. Funkcja cgetc musi pobrać bajt, więc gdy go nie ma, to zawsze czeka na jego dostarczenie. Przy odczycie z klawiatury zatrzymuje to pracę programu.

Funkcja cgetc jest dokładnym odpowiednikiem instrukcji GET w Atari Basic.

chget

PLIK: PMG.C
TYP: funkcja graficzna
DEFINICJA: chget(c,s)
char c,*s;
PRZYKŁADY:

```
chget ('T',pattern);
chget(20,array);
chget(chr,table);
```

DZIAŁANIE: Przepisuje bajty tworzące wzór wskazanego pierwszym argumentem znaku z aktualnie wykorzystywanego zestawu znaków do pierwszych ośmiu bajtów tablicy, której wskaźnikiem jest drugi argument. Wybór znaku jest dokonywany według kolejności w generatorze znaków, czyli według kodu wewnętrznego, a nie ATASCII.

choget

PLIK: PMG.C
TYP: funkcja graficzna
DEFINICJA: choget(c,s)
char c,*s;
PRZYKŁADY:

```
choget('T',pattern);
choget (20, array) ;
choget(chr,table);
```

DZIAŁANIE: Przepisuje bajty tworzące wzór wskazanego pierwszym argumentem znaku z wbudowanego, standardowego zestawu znaków do pierwszych ośmiu bajtów tablicy, której wskaźnikiem jest drugi argument. Wybór znaku jest dokonywany według kolejności w generatorze znaków, czyli według kodu wewnętrznego, a nie ATASCII.

chput

PLIK: PMG.C
TYP: funkcja graficzna
DEFINICJA: chput(c,s)
char c,*s;
PRZYKŁADY:

```
chput('T',pattern) ;
chput(20,"\0\0\0\60\60\0\0\0");
chput(chr,table);
```

DZIAŁANIE: Przepisuje bajty tworzące wzór wskazanego pierwszym argumentem znaku z pierwszych ośmiu bajtów tablicy, której wskaźnikiem jest drugi argument do aktualnie wykorzystywanego zestawu znaków. Wybór znaku jest dokonywany według kolejności w generatorze znaków, czyli według kodu wewnętrznego, a nie ATASCII.

ciov

PLIK: AIO.C
TYP: funkcja I/O
DEFINICJA: ciov (iocb, com, bad, blen, ax1, ax2)

```
int iocb,com,blen,ax1,ax2;
char *bad;
```

PRZYKŁADY:

```
status = ciov(1,254,"D:",2,000);
ciov(chn,35,file,strlen(file),0,0);
r = ciov(io,cmd,fn,k+1,a1,a2);
```

DZIAŁANIE: Funkcja ciov służy do wykonania poprzez kanał iocb operacji I/O, której kod jest podany jako com. Prawie wszystkie operacje, które dotyezą standardowych urządzeń systemu Atari - ekran, drukarka, magnetofon i stacja dysków (oprócz formatowania), mogą być zrealizowane przez inne funkcje, więc ciov jest w programach pisanych w Deep Blue C wykorzystywana rzadko. Stanowi ona natomiast składnik wielu pozostałych funkcji bibliotecznych, które są realizowane poprzez odpowiednie wywołanie ciov.

Funkcja ciov jest dokładnym odpowiednikiem instrukcji XIO w Atari Basic. Szczegółowy opis jej działania oraz znaczenia poszczególnych parametrów znajduje się w książce "Poradnik programisty Atari".

clear

PLIK: AIO.C
TYP: funkcja operacyjna
DEFINICJA: clear(a,len)
char *a; i n t len;
PRZYKŁADY:

```
clear(array,length);
clear(table,20);
clear(string,strlen(string));
```

DZIAŁANIE: Zeruje tablicę wskazaną pierwszym argumentem od bajtu 0 do bajtu len-1. Funkcja ta jest najczęściej wykorzystywana do kasowania znacznych obszarów pamięci i inicjowania dużych tablic. Przy inicjowaniu tablic całkowitych jako Jen należy podać podwojony rozmiar tej tablicy, gdyż każda liczba całkowita zajmuje dwa bajty.

close

PLIK: AIO.C
TYP: funkcja I/O
DEFINICJA: close(i)

```
char i;
```

PRZYKŁADY:

```
err = close(channel);
status = close(1);
close(iocb);
```

DZIAŁANIE: Zamyka kanał IOCB, który został otwarty przy pomocy funkcji open. Parametrem funkcji musi być numer zamykanego kanału (z zakresu od 0 do 7). Po prawidłowym zamknięciu pliku funkcja close zwraca wartość 1, a w przeciwnym przypadku ujemną wartość kodu błędu.

color

PLIK: GRAPHICS. C
TYP: funkcja graficzna
DEFINICJA: color(a)

```
char a;
```

PRZYKŁADY:

```
color(3);
color (x) ;
color('#');
```

DZIAŁANIE: Wybiera kolor lub znak, który będzie umieszczany na ekranie funkcjami plot i drawto. Wartość argumentu przekraczająca dostępną liczbę kolorów jest przez nią dzielona. Reszta z tego dzielenia wybiera rejestr koloru. Użyte tu liczby

są INNE niż w funkcji setcoior: zero oznacza zawsze kolor tła, a dalsze wartości — kolejne kolory.

Funkcja color jest dokładnym odpowiednikiem instrukcji COLOR w Atari Basic.

copen

PLIK: AIO.C
TYP: funkcja I/O
DEFINICJA: copen(fn,mode)
char *fn,mode;
PRZYKŁADY:

```
file = copen("K:",'r');
iocb = copen("P:",'w');
copen(file,'a');
channel = copen(name,type);
```

DZIAŁANIE: Otwiera plik wskazany pierwszym argumentem do odczytu, zapisu lub dopisania danych, w zależności od wartości drugiego argumentu (w zestawieniu podana jest także forma odpowiedniej funkcji open):

```
'r' - odczyt z pliku: open(iocb,4,0,fn)
'w' - zapis do pliku: open(iocb,8,0,fn)
'a' - dopisanie do pliku: open(iocb,12,0,fn)
```

Po prawidłowym otwarciu pliku funkcja copen zwraca numer kanału IOCB, który został użyty dla tego pliku. Numer ten powinien być zapamiętany dla umożliwienia późniejszego dostępu do pliku. Gdy wszystkie kanały IOCB są już otwarte i otwarcie kolejnego pliku jest niemożliwe, to wartością funkcji jest -1. Wystąpienie każdego innego błędu jest sygnalizowane przez ujemny kod tego błędu.

cprints

PLIK: AIO.C
TYP: funkcja I/O
DEFINICJA: cprints(str)
char *str;
PRZYKŁADY:

```
cprints("KOMPUTER");
stat = cprints(text);
dummy = cprints("Atari\n");
```

DZIAŁANIE: Wyświetla na ekranie ciąg znaków (tablicę znakową) wskazany przez argument. Normalnie nie zapisuje znaku końca wiersza EOL (kod 155), więc należy użyć w tekście symbolu "\n" lub po cprints wywołać funkcję putchar(155). Ponadto cprints nie pozwala na bezpośrednie wyświetlanie liczb . i kilku ciągów jednocześnie. Po poprawnej realizacji zapisu wartość zwracana przez funkcję jest równa 1, zaś w przeciwnym wypadku zwracany jest ujemny kod błędu.

Funkcja cprints jest w przybliżeniu odpowiednikiem instrukcji PRINT w Atari Basic.

cputc

PLIK: AIO.C
TYP: funkcja I/O
DEFINICJA: cputc(c,unit);
char c; int unit;
PRZYKŁADY:

```
stat  =  cputc(155,6);
err  =  cputc(byte,1);
cputc(peek(x),chan);
dummy = cputc('A',io);
```

DZIAŁANIE: Zapisuje, przez kanał wskazany drugim argumentem (z zakresu od 0 do 7), bajt podany jako pierwszy argument. Kanał IOCB musi być uprzednia otwarty, a dozwolone są numery od 0 do 7. Funkcja zwraca ujemny kod błędu, a gdy zapis był prawidłowy, to wartość 1.

Funkcja cputc jest dokładnym odpowiednikiem instrukcji PUT w Atari Basic.

cputs

PLIK: AIQ.C
TYP: funkcja I/O
DEFINICJA: cputs(str,i)
char *str; int i;
PRZYKŁADY:

```
cputs("KOMPUTER",2);
stat = cputs(text,chan);
dummy = cputs("Atari\n",file);
```

DZIAŁANIE: Zapisuje ciąg znaków (tablicę znakową) wskazany pierwszym argumentem przez kanał IOCB określony drugim argumentem. Normalnie nie zapisuje znaku końca wiersza EOL (kod 155), więc należy użyć w tekście symbolu "\n" lub po cputs wywołać funkcję cputc(155,i). Zwracana przez funkcje wartość jest po poprawnej realizacji równa 1, zaś w przeciwnym wypadku zwracany jest ujemny kod błędu.

Funkcja cputs jest w przybliżeniu odpowiednikiem instrukcji PRINT # w Atari Basic.

dpeek

PLIK: AIO.C
TYP: funkcja operacyjna
DEFINICJA: dpeek(i)
char *i;
PRZYKŁADY:

```
printf ( "%d", dpeek (x);
sc = dpeek(88); sm =
dpeek(dpeek(560)+4);
```

DZIAŁANIE: Zwraca liczbę całkowitą stanowiącą zawartość dwóch kolejnych komórek pamięci, z których pierwsza ma adres wskazany przez argument. Funkcja ta zawsze odczytuje wartość

dwubajtową, co odpowiada wykonaniu funkcji (patrz drugi przykład):

```
sc = peek(88) + 256 * peek(89);
```

dpoke

PLIK: AIO.C
TYP: funkcja operacyjna
DEFINICJA: dpoke(i,w)
```
char *i; int w;
```
PRZYKŁADY:
```
dummy = dpoke(560,0x600);
old = dpoke(addr,32768);
dpoke(836,dpeek(836)+128);
```
DZIAŁANIE: Umieszcza w dwóch kolejnych komórkach pamięci o adresie wskazanym przez pierwszy argument wartość, która jest drugim argumentem. Wartość ta jest umieszczana w normalnej kolejności bajtów - młodszy, starszy (LSB, MSB). Poprzednia zawartość tych komórek jest zwracana jako wynik funkcji. Funkcja dpoke umożliwia wiec zmiany wartości dwubajtowych.

drawto

PLIK: GRAPHICS.C
TYP: funkcja graficzna
DEFINICJA: drawto(x,y)
```
int x; char y;
```
PRZYKŁADY:
```
err = drawto(10,10);
status = drawto(h,v);
drawto(i*2,j+10);
```
DZIAŁANIE: Rysuje na ekranie linie prostą od aktualnego położenia kursora do punktu o podanych współrzędnych. Pierwsza liczba określa współrzędną poziomą, a druga pionową. Wybór koloru lub znaku do rysowania jest wykonywany przez color. Funkcja drawto zwraca wartość 1 lub ujemny kod błędu.

Funkcja drawto jest dokładnym odpowiednikiem instrukcji DRAWTO w Atari Basic.

fill

PLIK: GRAPHICS.C
TYP: funkcja graficzna
DEFINICJA: fill(x,y,c)
```
int x; char y,c;
```
PRZYKŁADY:
```
dummy = fill(20,10,2);
stat = fill(hor,vert,col);
fill(i*2,j+10,peek(763));
```

DZIAŁANIE: Wypełnia tło obrazu w prawo od każdego narysowanego punktu. Wypełnianie danej linii kończy się po napotkaniu punktu o kolorze innym niż kolor tła. Dwa pierwsze argumenty mają znaczenie podane w opisie funkcji drawto. Numer koloru, który będzie użyty do wypełniania, jest trzecim argumentem funkcji i odpowiada użytemu w funkcji color. Poniższy przykład jest przerobioną wersją przykładu ilustrującego użycie XIO 18, a zamieszczonego w "Poradniku programisty Atari".

```
main()
$(
  char y1,y2,c,f=1;
  int x1,x2;
  graphics(31);
  for(,,)
  $(
    x1=rnd(120)+40;
    y1=rnd(162)+30;
    do x2=rnd(140)
      while((x2>=x1) || <x2<x1-60));
   do y2=rnd<180)
      while( (y2>=Y1) || (y2<y1-60)) ;
    c++;
    if c>3 c=1;
    f++;
     if f>3 f=1;
     color(c);
     plot(x1,y1);
     drawto(x1,y2);
     drawto(x2,y2);
     fill(x2,y1,f>;
     while(peek(53279)!=6)
        $(
        $)
  $)
$)
```

find

PLIK: AIO.C
TYP: funkcja operacyjna
DEFINICJA: find(addr,len,ch)
```
          char *addr,ch;
          int len;
```
PRZYKŁADY:
```
        count = find("ATARI",5,'R');
        pos = find(text,strlen(text),97);
        pos = find(txt,len-2,char);
```
DZIAŁANIE: Przeszukuje obszar pamięci od adresu addr do addr+len-i i znajduje pierwsze wystąpienie znaku będącego trzecim argumentem. Jeśli znak taki nie zostanie znaleziony, to zwraca wartość -1. W przeciwnym przypadku rezultatem funkcji jest położenie znaku w badanym obszarze (wartość z zakresu od 0 do len-1).

printf

PLIK: PRINTF.C
TYP: funkcja I/O
DEFINICJA: fprintf(io,s, ...)
```
int io;
char *s;
```
PRZYKŁADY:
```
fprintf(1,"atari") ; fprintf
(chan, "%s",text) ;
fprintf(iocb,format,x,y,z);
fprintf (2,"%c %d %x",65,65,65) ;
```
DZIAŁANIE: Zapisuje przez kanał IOCB wskazany pierwszym argumentem i w formacie określonym przez drugi argument dane będące dalszymi argumentami. Jest to wiec funkcja wykonująca formatowany zapis przez kanał IOCB. Sposób formatowania zapisu jest wyjaśniony w opisie funkcji printf.

getchar

PLIK: AIO.C
TYP: funkcja I/O
DEFINICJA: getchar()
PRZYKŁADY:
```
code = getchar()
key = toupper(getchar());
getchar ()
```
DZIAŁANIE: Zwraca kod znaku odczytanego z edytora (IOCB 0). W przypadku wystąpienia błędu wynikiem funkcji jest ujemny kod tego błędu.

gets

PLIK: AIO.C
TYP: funkcja I/O
DEFINICJA: gets(str)
```
char *str;
```
PRZYKŁADY:
```
dummy = gets(answer);
len = gets(text);
gets(option);
```
DZIAŁANIE: Odczytuje z edytora cały wiersz logiczny wpisany przez użytkownika i umieszcza go w tablicy znakowej wskazanej przez argument. Użytkownik wprowadza informacją poprzez klawiaturę, a jest ona przyjmowana przez komputer dopiero po naciśnięciu klawisza <RETURN>. Maksymalna długość wiersza jest równa 120 znaków. Jeśli przy realizacji funkcji wystąpi błąd, to zwraca ona ujemny kod tego błędu. W przeciwnym przypadku wynikiem funkcji jest długość odczytanego wiersza (z zakresu od 0 do 120).

Funkcja gets jest w przybliżeniu odpowiednikiem instrukcji INPUT w Atari Basic.

graphics

PLIK: GRAPHICS.C
TYP: funkcja graficzna
DEFINICJA: graphics(n)
char n;
PRZYKŁADY:

```
status = graphics(1);
graphics(17);
dummy = graphics(1+32);
err = graphics(x);
```

DZIAŁANIE: Wybiera tryb wyświetlania obrazu według podanego argumentu. Poszczególnym trybom odpowiadają wartości argumentu z zakresu od 0 do 15. Ponadto zwiększenie numeru trybu o 16 ustala tryb graficzny bez okna tekstowego. Numer trybu zwiększony o 32 powoduje zachowanie niezmienionej zawartości obszaru pamięci obrazu. Oba te warianty mogą być zastosowane łącznie - numer trybu musi być wtedy zwiększony o 48. Rezultatem funkcji graphics jest 1 lub ujemny kod błędu. Wykaz dostępnych trybów oraz ich szczegółowy opis znajduje się w "Poradniku programisty Atari".

Funkcja graphics jest dokładnym odpowiednikiem instrukcji GRAPHICS w Atari Basic.

hitclear

PLIK: PMG.C
TYP: funkcja P/MG
DEFINICJA: hitclear()
PRZYKŁAD:

```
hitclear;
```

DZIAŁANIE: Kasuje wszystkie rejestry kolizji graczy i pocisków, a więc odpowiada dokładnie funkcji poke(53278,0). Zerowanie rejestrów kolizji po ich odczycie przez funkcje hitp2pf i hitp2pl jest niezbędne, aby zostały zarejestrowane następne kolizje duszków.

hitp2pf

PLIK: PMG.C
TYP: funkcja P/MG
DEFINICJA: hitp2pf(f,t)
char f,t;
PRZYKŁADY:

```
a = hitp2pf(0,2);
gol = hitp2pf(m,pf) ;
if hitp2pf(pl,2) hitting(pl);
```

DZIAŁANIE: Zwraca wartość określającą wystąpienie kolizji pomiędzy graczem wskazanym przez pierwszy argument i polem gry wskazanym przez drugi argument. Jeden oznacza wystąpienie kolizji, a zero jej brak. Oba argumenty mogą być z zakresu od 0 do 3. Wykonanie funkcji hitp2pf nie zmienia stanu rejestrów kolizji i przed ponownym użyciem trzeba je wyzerować przy pomocy hitclear.

hitp2pl

PLIK: PMG.C
TYP: funkcja P/MG
DEFINICJA: hitp2pl(f,t)
char f,t;
PRZYKŁADY:

```
a = hitp2pl(0,2);
gol = hitp2pl(tn,pl);
if hitp2pl(p1,2) hitting(pl);
```

DZIAŁANIE: Zwraca wartość określającą wystąpienie kolizji pomiędzy graczami wskazanymi przez argumenty. Jeden oznacza wystąpienie kolizji, a zero jej brak. Oba argumenty mogą być z zakresu od 0 do 3, a gdy są sobie równe, to wynikiem jest zawsze wartość 1. Wykonanie funkcji hitp2pl nie zmienia stanu rejestrów kolizji i przed ponownym użyciem trzeba je wyzerować przy pomocy hitclear.

hstick

PLIK: GRAPHICS.C
TYP: funkcja manipulatorów
DEFINICJA: hstick(n) char n;
PRZYKŁADY:

```
printf ("%d",hstick(1));
x = hstick(y); if
hstick(a) == 1 h-=1;
```

DZIAŁANIE: Zwraca wartość określającą położenie w płaszczyźnie poziomej joysticka przyłączonego do portu o numerze wskazywanym przez argument. Wynik jest równy zero, gdy joystick jest w położeniu neutralnym. Wychylenie joysticka w lewo daje wartość -1, zaś w prawo 1. Zależność miedzy funkcjami stick i hstick jest więc następująca:

```
stick>8 i stick<12  =>  hstick=-1
    stick>12        =>  hstick=0
    stick<8         =>  hstick=1
```

hval

PLIK: AIO.C
TYP: funkcja liczbowa
DEFINICJA: hval(s)
char *s;
PRZYKŁADY:

```
number = hval(a);
x = hval("OX3E4");
```

DZIAŁANIE: Zamienia ciąg znaków wskazany przez argument na reprezentowaną przez niego liczbę szesnastkową. Zamiana jest dokonywana znak po znaku, od lewej do prawej, aż do napotkania niedozwolonego znaku.

locate

PLIK: GRAPHICS.C
TYP: funkcja graficzna
DEFINICJA: locate(x,y)
int x;
char y;
PRZYKŁADY:

```
a = locate(10,5);
code = locate(x*2, y);
```

DZIAŁANIE: Umieszcza kursor w punkcie ekranu o podanych współrzędnych i zwraca kod znajdującego się tam znaku lub koloru. W trybie znakowym jest to kod ASCII znaku, zaś w trybie bitowym — kod koloru. W przypadku wystąpienia błędu wynikiem jest ujemny kod tego błędu.

Funkcja locate jest dokładnym odpowiednikiem instrukcji LOCATE w Atari Basic.

move

PLIK: AIO.C
TYP: funkcja operacyjna
DEFINICJA: move(a,b,len)
char *a,*b;
int len;
PRZYKŁADY:

```
move(57344,chb,1024);
move(pm,pm+2,256);
move(adr_1,adr_2,count);
```

DZIAŁANIE: Przemieszcza blok pamięci o długości określonej przez trzeci argument z obszaru rozpoczynającego się od adresu a do obszaru od adresu b. Jeśli wybrane obszary pamięci się pokrywają i a jest mniejsze od b, to część przepisywanego bloku ulegnie zniszczeniu. W takim przypadku należy użyć pomocniczego obszaru i funkcję move wywołać dwukrotnie.

normalize

PLIK: AIO.C
TYP: funkcja znakowa
DEFINICJA: normalize(fname,fext)
char *fname,*fext;
PRZYKŁADY:

```
normalize(file,"DAT");
normalize(name,ext);
```

DZIAŁANIE: Zamienia nazwę pliku wskazaną pierwszym argumentem na poprawną* postać (wymaganą przez system operacyjny). Najpierw wszystkie litery nazwy są zamieniane na duże. Następnie na początku nazwy dopisywany jest symbol stacji dysków "D:" (jeśli go nie ma). Na zakończenie, gdy nazwa nie zawiera kropki, na jej końcu dodawana jest kropka i trzy znaki wskazane przez drugi argument. Na przykład, podana przez użytkownika nazwa "program" po wykonaniu funkcji pokazanej w pierwszym przykładzie przybierze postać "D:PROGRAM.DAT".'

open

PLIK: AIO.C
TYP: funkcja I/O
DEFINICJA: open(iocb,ax1,ax2,fname)
char iocb,ax1,ax2,*fname;
PRZYKŁADY:

```
dummy = open(1,4,0,"K:");
status = open(chn,x,0,"P;");
err = open(2,4,0,"D:NAZWA.DAT");
stat = open(k,a,b,file);
dir = open(1,6,0,"Ds*.*">;
open(5,12,7,"S;");
```

DZIAŁANIE: Otwiera kanał IOCB do komunikacji z urządzeniem zewnętrznym. Kolejne argumenty funkcji określają zadane parametry transmisji. Funkcja zwraca ujemną wartość kodu błędu lub wartość 1, gdy operacja przebiegła poprawnie.

Funkcja open jest dokładnym odpowiednikiem instrukcji OPEN w Atari Basic. Znaczenie wszystkich argumentów open jest więc identyczne, jak parametrów OPEN i jest opisane w "Poradniku programisty Atari".

paddle

PLIK: GRAPHICS.C
TYP: funkcja manipulatorów
DEFINICJA: paddle(n) char n;
PRZYKŁADY:

```
printf("%d", paddle(2));
x = paddle(y);
if paddle(a)>100 change(a);
```

DZIAŁANIE: Zwraca wartość określającą położenie suwaka potencjometru przyłączonego do portu o numerze wskazywanym przez argument. Dozwolone są wartości argumentu z zakresu od 0 do 3, a wynik może przyjmować wartości z zakresu od 1 do 228.

Funkcja paddle jest dokładnym odpowiednikiem funkcji PADDLE w Atari Basic.

peek

PLIK: AIO.C
TYP: funkcja operacyjna
DEFINICJA: peek(i)
char *i;
PRZYKŁADY:

```
printf("%c",peek(x+5));
sc = peek(88)+256*peek(89);
while(peek(53279)==7) $( $)
```

DZIAŁANIE: Zwraca liczbę całkowitą stanowiącą zawartość komórki pamięci, której adres jest wskazany przez argument. Dozwolone są wartości argumentu od 0 do 65535, zaś uzyskany wynik jest z zakresu od 0 do 255. W celu odczytania wartości dwubajtowej (z zakresu od 0 do 65535) należy zastosować funkcję

dpeek lub wywołać peek dwukrotnie (przykład drugi).
Funkcja peek jest dokładnym odpowiednikiem funkcji PEEK w Atari Basic.

pladdr

PLIK: PMG.C
TYP: funkcja P/MG
DEFINICJA: pladdr(n)
char n;
PRZYKŁADY:

```
p0 = pladdr(1);
player = pladdr(x);
move(pattern,pladdr(n),10);
```

DZIAŁANIE: Zwraca adres, od którego rozpoczyna się obszar pamięci wykorzystywany przez dane gracza o numerze będącym argumentem. Dozwolone są wartości argumentu z zakresu od 0 do 4. Dla argumentu równego 4 wynikiem jest adres obszaru pocisków.

plmove

PLIK: PMG.C
TYP: funkcja P/MG
DEFINICJA: plmove(n,x,y,shape)
char n,x,y,*shape;
PRZYKŁADY:

```
plmove(0, 100,2, "\5\0\0\60\0\0");
plmove(i,h,v,pattern);
plmovefi,h,++v,pattern);
```

DZIAŁANIE: Umieszcza na ekranie gracza wskazanego przez pierwszy argument. Dozwolone są wartości tego argumentu z zakresu od 0 do 3. Drugi argument określa poziome położenie lewej krawędzi gracza, a trzeci pionowe położenie jego górnej krawędzi. Współrzędne te są podane w wartościach odpowiadających rozdzielczości wybranej przez funkcję pmgraphics. Wzór gracza jest pobierany z tablicy wskazanej przez czwarty argument funkcji. Pierwszy bajt tej tablicy określa liczbę bajtów wzoru, a pozostałe go tworzą. Na przykład, jeśli size==shape[0], to wzór jest zapisany od shape[1] do shape[size].

plot

PLIK: GRAPHICS.C
TYP: funkcja graficzna
DEFINICJA: plot(x,y)
int x;
char y;
PRZYKŁADY:

```
dummy = plot(10,10);
status = plot<h,v);
plot<i*2,j+10>;
```

DZIAŁANIE: Umieszcza na ekranie punkt lub znak w miejscu o podanych współrzędnych i zwraca wartość 1 lub ujemny kod błędu.

Punkt 0,0 znajduje się w lewym, górnym rogu ekranu. Pierwsza liczba określa współrzędną poziomą, a druga pionową. Wybór koloru (w trybach bitowych) lub znaku (w trybach znakowych), który zostanie narysowany, jest dokonywany przez funkcją color.

Funkcja plot jest dokładnym odpowiednikiem instrukcji PLOT w Atari Basic.

pmcflush

PLIK: PMG.C
TYP: funkcja P/MS
DEFINICJA: pmcflush ()
PRZYKŁAD:

```
pmcflush ();
```

DZIAŁANIE: Wyłącza grafiką graczy i pocisków i kasuje obszar pamięci zajmowany przez nią oraz przeznaczony dla dodatkowego zestawu znaków. Po wykonaniu pmcflush następuje przejście do trybu graficznego 0. Funkcja ta powinna być wywoływana tylko jeden raz, tuż przed końcem programu.

pmcinit

PLIK: PMB.C
TYP: funkcja P/MG
DEFINICJA: pmcinit()
PRZYKŁAD:

```
pmcinit ();
```

DZIAŁANIE: Inicjuje grafiką graczy i pocisków oraz dodatkowy zestaw znaków, który zostaje przepisany z obszaru ROM. Wymaga to zajęcia dodatkowego obszaru 4 KB pamięci. Funkcja pmcinit może być użyta tylko jeden raz i musi to nastąpić przed wywołaniem jakiejkolwiek innej funkcji z pliku PMG.C, a najlepiej przed pierwszym wywołaniem graphics.

pmclear

PLIK: PMG.C
TYP: funkcja P/MG
DEFINICJA: pmclear(n)

```
char n;
```

PRZYKŁADY:

```
pmclear(1);
pmclear (x);
```

DZIAŁANIE: Zeruje obszar pamiąci przeznaczony na dane gracza określonego przez argument. Dozwolone są wartości argumentu z zakresu od 0 do 4. Dla argumentu równego 4 kasowane są dane wszystkich pocisków.

Pmcolor

PLIK: PMG.C
TYF: funkcja P/MG

DEFINICJA: pmcolor(n,c,i)
char n,c,i;
PRZYKŁADY:
pmcolor(2,0,0)$
pmcolor(i, j , k) ;
DZIAŁANIE: Ustala barwę i stopień jasności gracza wskazanego przez pierwszy argument (dotyczy to zarówno gracza, jak i pocisku). Drugi argument określa barwę, a trzeci stopień jasności koloru. Argumenty te są identyczne ze stosowanymi w funkcji setcolor.
Ponieważ rejestry koloru są zwykłymi komórkami pamięci RAM, to zamiast funkcji pmcolor(i,j,k) można zastosować poke(704*i,16+j+k>.

pmgraphics

PLIK: PMG.C
TYP: funkcja P/MG
DEFINICJA: pmgraphics(i)
int i;
PRZYKŁADY:
pmgraphics(0);
pmgraphics(1);
pmgraphics(res) ;
DZIAŁANIE: Ustala rozdzielczość grafiki graczy i pocisków według wartości argumentu. Dla argumentu równego 1 P/MG ma rozdzielczość jednoliniową, dla równego 2 dwuliniową, zaś argument równy .zero powoduje wyłączenie duszków. Funkcja pmgraphics musi być użyta po każdym wywołaniu funkcji graphics.

pmwidth

PLIK: PM6.C
TYP: funkcja P/MG
DEFINICJA: pmwidth(n,w)
char n,w;
PRZYKŁADY:
pmwidth(2,1);
pmwidth(x,3);
pmwidth(0,y);
DZIAŁANIE: Ustala wybraną szerokość wyświetlania gracza wskazanego przez pierwszy argument. Wartości tego argumentu mogą być z zakresu od 0 dp 4. Dla argumentu równego 4 ustalana jest szerokość wszystkich pocisków równocześnie. Drugi argument oznacza szerokość duszka: 0 - pojedynczą, 1 - podwójną, a 3 -poczwórną.

poke

PLIK: AIO.C
TYP: funkcja operacyjna
DEFINICJA: poke(i,d)
char *i,d;

PRZYKŁADY:

```
old = poke(708,2);
dummy = poke(addr,0);
poke(712,peek(712)+16);
```

DZIAŁANIE: Umieszcza w komórce pamięci o adresie wskazanym przez pierwszy argument wartość, która jest drugim argumentem. Wynikiem zwracanym przez tą funkcją jest poprzednia zawartość zmienianej komórki. Jeżeli konieczne jest wpisanie wartości dwubajtowej (od 0 do 65535), to trzeba zastosować funkcję dpoke.

position

PLIK: GRAPHICS.C
TYP: funkcja graficzna
DEFINICJA: position(x,y)

```
int x;
char y;
```

PRZYKŁADY:

```
position(10,10);
position(h,v);
position(i*2,j+10);
```

DZIAŁANIE: Umieszcza kursor na ekranie w miejscu o podanych współrzędnych. Punkt 0,0 znajduje się w lewym, górnym rogu ekranu. Pierwszy argument określa współrzędną poziomą, a drugi pionową. Po wykonaniu funkcji położenie kursora pozostaje bez zmian, aż do następnej instrukcji wyjścia na ekran.

Printf

PLIK: PRINTF.C
TYP: funkcja I/O DEFINICJA: printf(s, ...)

```
char *s;
```

PRZYKŁADY:

```
printf("atari");
printf("%s",text);
printf(format,x,y,z);
printf ("%c %d %x",65,65,65);
```

DZIAŁANIE: Wyświetla na ekranie w formacie określonym przez pierwszy argument dane będące dalszymi argumentami. Jest to więc funkcja wykonująca formatowany zapis na ekranie. W ciągu formatującym specjalne znaczenie ma znak "X". Oznacza on, że znajdujący się po nim znak będzie definiował format. Możliwe są następujące określenia formatu:

```
d - liczba dziesiętna
x - liczba szesnastkowa
c - znak
s - ciąg znaków
% - znak "%"
```

Liczba wyświetlanych argumentów nie może być mniejsza niż liczba znaków "%", gdyż spowoduje to wyświetlenie "śmieci".

Pomiędzy znakiem "%" i literą określającą format można umieścić liczbę, która oznacza szerokość pola przeznaczonego na wyświetlany argument. Jeśli liczba ta jest dodatnia, to argument będzie przesunięty w prawo, a w przeciwnym przypadku w lewo. Wszystkie pozostałe znaki znajdujące się w ciągu określającym format są wyświetlane normalnie.

Poniżej podane są przykłady ilustrujące działanie funkcji printf:

```
            funkcja                        rezultat

      printf("atari");                 atari
      printf("=%s=","atari");          =atari=
      printf ("=%5d=",77);             =   77=
      printf ("=%-5d=",77);            =77   -
      printf ("%c %d %x" ,65,65,65) ;A 65 41
```

ptrig

PLIK: GRAPHICS.C
TYP: funkcja manipulatorów
DEFINICJA: ptrig(n)
```
            char n;
```
PRZYKŁADY:
```
         printf("%d",ptrig(2));
         x = ptrig(y);
         if ptrig(a) fire(a);
```
DZIAŁANIE: Zwraca wartość określającą stan przycisku potencjometru wskazanego przez argument. Gdy przycisk jest wciśnięty, to wynikiem jest wartość zero, a w przeciwnym przypadku jeden. Dopuszczalne są wartości argumentu od 0 do 3.

Funkcja ptrig jest dokładnym odpowiednikiem funkcji PTRIG w Atari Basic.

putchar

PLIK: AIO.C
TYP: funkcja I/O
DEFINICJA: putchar(c)
```
            char c;
```
PRZYKŁADY:
```
         dummy = putchar(155);
         status = putchar(x);
         putchar('#');
```
DZIAŁANIE: Umieszcza znak będący argumentem na ekranie edytora. Wynikiem funkcji jest wartość 1 lub ujemny kod błędu.

rnd

PLIK: GRAPHICS.C
TYP: funkcja liczbowa
DEFINICJA: rnd(n)
```
            int n;
```

PRZYKŁADY:

```
printf ("%d", 1000);
x = rnd(11)+10;
if rnd(100)<50 cprints("50 % szansy\n");
```

DZIAŁANIE: Zwraca wartość losową, która jest liczbą całkowitą mniejszą od argumentu i większą lub równą zero, czyli z przedziału <0,n). W celu uzyskania innych liczb losowych funkcja md musi być uzupełniona dodatkowymi operacjami. Wzór ogólny dla uzyskania liczby losowej z przedziału <a,b> jest następujący:

```
rnd(b-a+1)+a;
```

setcolor

PLIK: GRAPHICS.C
TYP: funkcja graficzna
DEFINICJA:

```
setcolor(reg,hue,lum)
        int reg;
        char hue,lum;
```

PRZYKŁADY:

```
setcolor(2,0,0);
setcolor(i,j,k);
```

DZIAŁANIE: Ustala barwę i stopień jasności wskazanego koloru. Numer rejestru koloru jest określany przez pierwszy argument, który może przyjmować wartości od 0 do 4. Drugi argument oznacza barwę, zaś trzeci stopień jasności koloru. Jasność może mieć wartości parzyste od 0 do 14.

Funkcja setcolor jest dokładnym odpowiednikiem instrukcji SETCOLOR w Atari Basic. Ponieważ rejestry koloru są zwykłymi komórkami pamięci RAM, to zamiast setcolor (i,j,k) można zastosować poke(708+i,16*j+k).

sound

PLIK: GRAPHICS.C
TYP: funkcja dźwiękowa
DEFINICJA:

```
sound(vc,p,d,vi)
      char vc,p,d,vl;
```

PRZYKŁADY:

```
sound(0,200,10,10);
sound(k,10*l,m,15-n);
sound(k,0,0,0);
```

DZIAŁANIE: Ustawia generator dźwięku, którego numer jest pierwszym argumentem funkcji (z zakresu od 0 do 3). Jeśli wszystkie pozostałe argumenty są zerami, to następuje wyłączenie generatora. Drugi argument funkcji (od 0 do 255) określa okres dźwięku, a więc pośrednio jego częstotliwość. Argument trzeci wybiera rodzaj zniekształceń tworzonego dźwięku (wartości parzyste od 0 do 14 — nieparzyste wyłączają generator). Czysty ton uzyskuje się dla wartości 10 i 14. Ostatnim argumentem jest liczba z zakresu od 0 do 15, która ustala głośność dźwięku.

Funkcja sound jest dokładnym odpowiednikiem instrukcji SOUND w Atari Basic.

stick

PLIK: GRAPHICS.C
TYP: funkcja manipulatorów
DEFINICJA: stick(n)
char n;
PRZYKŁADY:

```
printf ("%d",stick (1));
x = stick(y);
if stick(a)=13 h--;
```

DZIAŁANIE: Zwraca wartość określającą połażenie joysticka przyłączonego do portu o numerze wskazywanym przez argument. Wynik ten może przyjmować wartości z zakresu od 5 do 7, od 9 do 11 oraz od 13 do 15 według zamieszczonego poniżej schematu.

```
10 14  6
  \ | /
11 - 15 - 7
  / | \
 9 13  5
```

Funkcja stick jest dokładnym odpowiednikiem funkcji STICK w Atari Basic.

Strcpy

PLIK: AIO.C
TYP: funkcja znakowa
DEFINICJA: strcpy(a,b)
char *a,*b;
PRZYKŁADY:

```
len = strcpy(dest,source);
dummy = strcpy(name,"atari");
strcpy(file,"D:OBRAZ.DAT");
```

DZIAŁANIE: Kopiuje zawartość tablicy znakowej wskazanej drugim argumentem do tablicy wskazanej przez pierwszy argument. Jako rezultat funkcji zwracana jest liczba przepisanych bajtów (bez zera kończącego tablice).

strig

FLIK: GRAPHICS.C
TYP: funkcja manipulatorów
DEFINICJA: strig(n)
char n;
PRZYKŁADY:

```
printf ("%d",strig (0));
x = strig(y);
if strig(a)==0 fire(a);
```

DZIAŁANIE: Zwraca wartość określającą stan przycisku joysticka przyłączonego do gniazda wskazanego przez argument. Gdy przycisk jest wciśnięty, to wynikiem jest zero, a w przeciwnym przypadku jeden. Argument również może mieć wartości 0 lub 1.

Funkcja strig jest dokładnym odpowiednikiem funkcji STRIG w Atari Basic.

strlen

PLIK: AIO.C
TYP: funkcja znakowa
DEFINICJA: strlen(str)
char *str;
PRZYKŁADY:

```
len = strlen("Atari");
size = strlen(array);
```

DZIAŁANIE: Zwraca wartość określającą liczbą znaków zawartych w tablicy wskazanej przez argument. Do wyniku nie wlicza się zera kończącego tablicę.

tolower

PLIK: AIO.C
TYP: funkcja znakowa
DEFINICJA: tolower(c)
char c;
PRZYKŁADY:

```
chr = tolower(chr);
key = tolower(cgetc(1));
```

DZIAŁANIE: Jeżeli argumentem jest duża litera, to funkcja zwraca odpowiadającą jej małą literę. W przeciwnym przypadku wynikiem jest argument funkcji.

toupper

PLIK; AIO.C
TYP: funkcja znakowa
DEFINICJA: toupper(c)
char c;
PRZYKŁADY:

```
chr = toupper(chr);
key = toupper(cgetc(1));
```

DZIAŁANIE: Jeżeli argumentem jest mała litera, to funkcja zwraca odpowiadającą jej dużą literę. W przeciwnym przypadku wynikiem jest argument funkcji.

usr

PLIK: AIO.C
TYP: funkcja operacyjna
DEFINICJA: usr(addr, ...)
char *addr;
PRZYKŁADY:

```
x = usr(1536);
usr(pmg,2,hpos,vpos);
usr(ml,t,strlen(t),h,v);
```

DZIAŁANIE: Powoduje wywołanie procedury w języku maszynowym. Adres tej procedury musi być podany jako pierwszy argument funkcji. Przed rozpoczęciem realizacji procedury, na stosie procesora odkładane są pozostałe argumenty w postaci liczb dwubajtowych (w kolejności: LSB - MSB). Procedura musi je więc odczytywać w odwrotnej kolejności. Jako ostatni odkładany jest na stos jeden bajt, który określa liczbę dodatkowych argumentów <gdy jest tylko adres, to zero). Funkcja usr może mieć do 120 dodatkowych argumentów.

Funkcja usr jest dokładnym odpowiednikiem funkcji USR w Atari Basic.

val

PLIK: AIO.C
TYP: funkcja liczbowa
DEFINICJA: val(s)
```
        char *s;
```
PRZYKŁADY:
```
number = val(a);
x = val("1234");
```
DZIAŁANIE: Zamienia ciąg znaków wskazany przez argument na reprezentowaną przez niego liczbę dziesiętną. Zamiana jest dokonywana znak po znaku, od lewej do prawej, aż do napotkania niedozwolonego znaku.

Funkcja val jest dokładnym odpowiednikiem - funkcji VAL w Atari Basic.

vstick

PLIK: GRAPHICS.C

TYP: funkcja manipulatorów
DEFINICJA: vstick(n)
```
      char n;
```
PRZYKŁADY:
```
vert = vstick(1);
x = vstick(y);
if vstick(a)=1 v--;
```
DZIAŁANIE: Zwraca wartość określającą położenie w płaszczyźnie pionowej joysticka przyłączonego do portu o numerze wskazywanym przez argument. Wynik jest równy zero, gdy joystick jest w położeniu neutralnym. Wychylenie joysticka w dół daje wartość -1, zaś w górę 1. Zależność między funkcjami stick i vstick jest więc następująca:

```
stick=5 lub stick=9  lub stick=13  =>  vstick=-1
stick=7 lub stick=11 lub stick=15  =>  vstick=0
stick=6 lub stick=10 lub stick=14  =>  vstick=1
```

Dopuszczalne są wartości argumentu 0 lub 1.

2.7. Biblioteka matematyczna

Standardowy język C umożliwia korzystanie z wartości znakowych (char), całkowitych (int) i rzeczywistych (float). Deep Blue C nie posiada tej ostatniej możliwości. W celu zbliżenia da standardu i zwiększenia użyteczności języka została opracowana przez Franka Parisa biblioteka zmiennoprzecinkowych funkcji matematycznych i trygonometrycznych. Jest ona zawarta w plikach MATHLIB.C i TRIG.C.

Użycie funkcji matematycznych wymaga pewnego przygotowania. Przede wszystkim do programu musi być dołączona biblioteka standardowa zawarta w pliku AIO.C. Na początku programu trzeba wywołać funkcję c_ial inicjującą bibliotekę matematyczną. W przypadku stosowania funkcji trygonometrycznych z pliku TRIG.C należy dodatkowo wywołać funkcję c_itrig.

Biblioteka MATHLIB używa standardowych liczb rzeczywistych Atari. Liczby te są w Deep Blue C zapisywane w sześcioelementowych tablicach znakowych. Z tego powodu każda zmienna rzeczywista wymaga zdefiniowania dla niej odpowiedniej tablicy, na przykład:

```
char number[6]; char
x[6],y[6],z[6];
```

Wykonanie podobnej operacji jest niezbędne również dla zdefiniowania stałych rzeczywistych. W tym przypadku należy użyć funkcji c_afp, która dokonuje zamiany ciągu znaków ASCII na liczbę rzeczywistą. Na przykład, utworzenie stałej o wartości równej liczbie pi wymaga przeprowadzenia następującej operacji:

```
char pi[6],*number;
number = "3.14159265";
c_afp(number,pi);
```

Zapisane w tablicach znakowych liczby rzeczywiste mają standardowy format liczb rzeczywistych Atari i w tej postaci nie można 'ich wyświetlić, ani wydrukować. Dla zrealizowania tych operacji konieczne jest wykonanie odwrotnej czynności, czyli zamiany liczby rzeczywistej na ciąg znaków przy użyciu funkcji c_fasc. Uzyskany w ten sposób tekst można wyświetlać i drukować tak, jak każdy inny tekst w Deep Blue C.

Ten rozdział zawiera kompletny słownik funkcji znajdujących się w bibliotece matematycznej i trygonometrycznej Deep Blue C opisanych w kolejności alfabetycznej. Podana jest tu nazwa każdej funkcji, jej definicja, sposoby wykorzystania oraz liczne przykłady. Ponadto na początku zamieszczony jest spis tych słów.

W słowniku przyjęta została następująca kolejność opisu: nazwa, plik, typ, definicja, przykłady i działanie. Typ określa rodzaj funkcji. Definicja opisuje sposób wywołania i deklaracje parametrów wywołania. Pozostałe punkty nie wymagają objaśnienia.

FUNKCJE W PLIKU MATHLIB.C

c_abs	c_fasc	c_irol
c_afp	c_fdiv	c_int
c_alog	c_fmul	C_LOG
c_alog10	c_fpi	c_log10
c_chs	c_fpsi	c_move
c_cmp	c_frac	c_sifp
c_exp	c_fsub	c_sqrt
c_fadd	c_ifp	c_zero

FUNKCJE W PLIKU TRIO_C

c_atan	c_dr	c_rd
c_cos	c_itrig	c_sin
c_ddms	c_rad	c_tan
c_dfflsd		

c_abs

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_abs(fpn,absfpn)
char *fpn,*absfpn;
PRZYKŁAD:

```
char *pntr,nbr[6],absnbr[6],answer[17];
pntr = "-15.7895";
c_afp(pntr,nbr);
c_abs(nbr,absnbr);
c_fasc(absnbr,answer);
printf("%s",answer);
```

DZIAŁANIE: Oblicza wartość bezwzględną liczby wskazanej przez pierwszy argument i umieszcza ją w tablicy wskazanej drugim argumentem. Dla liczb dodatnich jest to wartość argumentu, dla zera - zero, a dla liczb ujemnych - wartość argumentu pomnożona przez -1 . Oba argumenty mogą być tą samą zmienną. Funkcja ta nie zwraca statusu wykonanej operacji.

Funkcja c_abs jest dokładnym odpowiednikiem funkcji ABS w Atari Basic.

c_afp

PLIK: MATHLIB.C
TYP: funkcja transformacji
DEFINICJA: c_afp(atascii,fp)
char *atascii,*fp;
PRZYKŁADY:

```
c_afp(text,number);
status = c_afp(alfa,ax);
if c_afp (table, temp)==0 cprints("liczba");
```

DZIAŁANIE: Zamienia ciąg znaków wskazany pierwszym argumentem na liczbę rzeczywistą umieszczoną w tablicy wskazanej przez drugi argument. Konwersja jest prowadzona, aż do napotkania znaku, który nie jest poprawnym elementem liczby. Jeśli pierwszy znak ciągu jest niewłaściwy, to funkcja zwraca wartość -1, a w przeciwnym przypadku zero.

c_alog

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_alog(exp,result)
char *exp,*resułt;
PRZYKŁAD:

```
char *pntr,nbr[6],log[6],answer[17];
int status;
pntr = "5.5471754";
c_afp(pntr,nbr) ;
status = c_alog(nbr,log);
c_f asc(log,answer);
printf ("%d %s",status,answer);
```

DZIAŁANIE: Podnosi liczbę e (2.71828) do potęgi określonej

przez pierwszy argument i umieszcza wynik w tablicy wskazanej drugim argumentem. Argumentem funkcji c_ałog może być dowolna liczba rzeczywista, a wynik jest zawsze liczbą dodatnią. Oba argumenty mogą być tą samą zmienną. Funkcją odwrotną do c_alog jest c_log. Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

Funkcja c_alog jest dokładnym odpowiednikiem funkcji EXP w Atari Basic.

c_alog10

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_alog10(exp,result)
char *exp,*result;
PRZYKŁAD:

```
char *pntr,nbr[6],log[6],answer[17];
int status; pntr = "5.5471754";
c_afp(pntr,nbr);
status = c_alogl0(nbr,log);
c_fasc(log, answer);
printf("%d %s",status,answer);
```

DZIAŁANIE: Podnosi liczbą 10 do potęgi określonej przez pierwszy argument i umieszcza wynik w tablicy wskazanej drugim argumentem. Argumentem funkcji c_alog10 może być dowolna liczba rzeczywista, a wynik jest zawsze liczbą dodatnią. Oba argumenty mogą być tą samą zmienną. Funkcją odwrotną do c_alog10 jest c_log10. Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

Funkcji c_alog10 odpowiada w Atari Basic operacja potęgowania liczby 10.

c_atan

PLIK: TRIG.C
TYP: funkcja trygonometryczna
DEFINICJA: c_atan(x,atan);
char *x,*atan;
PRZYKŁAD:

```
char *pntr,deg45[6],atan45[6],answer[17];
int status; rad(0); pntr = "1";
c_afp(pntr,deg45);
status = c_atan(deg45,atan45);
c_fasc(atan45,answer);
printf ("%d %s",status,answer) ;
```

DZIAŁANIE: Oblicza wartość funkcji arcus tangens pierwszego argumentu i umieszcza ją w tablicy wskazanej drugim argumentem. Pierwszy argument może mieć dowolną wartość dozwoloną w Atari. Wynikiem jest liczba z przedziału od -1.5708 do +1.5708 przy wykonywaniu obliczeń w radianach lub od -90 do +90 przy obliczaniu w stopniach (patrz opis funkcji c_rad). Jeśli podczas

obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

Funkcja c_atan jest dokładnym odpowiednikiem funkcji ATN w Atari Basic.

c_chs

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_chs(fpn,negfpn)

```
char *fpn,*negfpn;
```

PRZYKŁAD:

```
char *pntr,nbr[6],output[6],answer[17];
pntr = "15.7895";
c_afp <pntr,nbr);
c_chs(nbr,output);
c_fasc(output,answer);
printf ("%s" , answer) ;
```

DZIAŁANIE: Zmienia znak liczby wskazanej przez pierwszy argument i umieszcza wynik w tablicy wskazanej drugim argumentem. Oba argumenty mogą być tą samą zmienną. Funkcja ta nie zwraca statusu wykonanej operacji.

c_cmp

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_cmp(fpn1,fpn2)

```
char *fpn1,*fpn2;
```

PRZYKŁAD:

```
char *pntr,fp1[6],fp2[6];
int status;
pntr = "27.12";
c_afp(pntr,fp1);
pntr = "17.89";
c_afp(pntr,fp2);
status = c_cmp (fp1, fp2);
printf ("%d" , status) ;
```

DZIAŁANIE: Zwraca wartość określającą wynik porównania dwóch liczb rzeczywistych wskazanych przez argumenty. Wynikiem jest wartość -1, gdy pierwszy argument jest mniejszy od drugiego; 1, gdy większy; zaś zero ,gdy oba argumenty są równe.

c_cos

PLIK: TRIG.C
TYP: funkcja trygonometryczna
DEFINICJA: c_cos(angle,result)

```
char *angle,*result;
```

PRZYKŁAD:

```
char *pntr,nbr[6],cosine[6],answer[17];
int status;
rad(0);
```

```
pntr = "30";
c_afp(pntr,nbr);
status = c_cos(nbr,cosine);
c_fasc(cosine,answer);
printf("%d %s",status,answer);
```

DZIAŁANIE: Oblicza wartość funkcji cosinus pierwszego argumentu i umieszcza ją w tablicy wskazanej drugim argumentem. Pierwszy argument może mieć dowolną wartość dozwoloną w Atari. Wynikiem jest liczba z przedziału od -1 do +1, a jej wartość zależy od jednostek, w których wykonywane jest obliczenie (stopnie lub radiany - patrz opis funkcji c_rad) . Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość —1, a w przeciwnym przypadku zero.

Funkcja c_cos jest dokładnym odpowiednikiem funkcji COS w Atari Basic.

c_ddms

PLIK: TRIG.C
TYP: funkcja transformacji
DEFINICJA: c_ddms(dd,degrees,minutes,seconds)

```
char *dd,*degrees,*minutes,*seconds;
```

PRZYKŁAD:

```
char *pntr,deg[6],min[6],sec[6],ddeg[6]
char *ans1[17],ans2[17],ans3[17];
int status; pntr = "30.42694444";
c_afp(pntr,ddeg);
status = c_ddms(ddeg,deg,min,sec);
c_fasc(deg,ans1); c_fasc(min,ans2);
c_fasc(sec,ans3);
printf("%d %s %s %s",status, ans1,ans2,ans3);
```

DZIAŁANIE: Zamienia wartość kąta wyrażoną w stopniach i ich ułamkach umieszczoną w tablicy wskazanej pierwszym argumentem na wyrażoną w stopniach, minutach i sekundach oraz przypisuje ją zmiennym będącym dalszymi trzema argumentami. Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

c_dmsd

PLIK: TRIG.C
TYP: funkcja transformacji
DEFINICJA: c_dmsd(decrees,minutes,seconds,dd)

```
char *degrees,*minutes,*seconds,*dd;
```

PRZYKŁAD:

```
char
*pntr,deg[6],min[6],sec[6],ddeg[6],answer[17];
int status;
pntr = "30";
c_afp(pntr,deg);
pntr = "25";
c_afp(pntr,min);
```

```
pntr = "37";
c_afp(pntr,sec);
 status = c_dmsd (deg,mm,sec,ddeg);
cfasc(ddeg,answer);
printf("%d %s",status,answer);
```

DZIAŁANIE: Zamienia wartość kąta wyrażoną w stopniach, minutach i sekundach umieszczonych w tablicach wskazanych pierwszymi trzema argumentami na wyrażoną w stopniach i ich ułamkach oraz przypisuje ją zmiennej będącej czwartym argumentem. Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

c_dr

PLIK: TRIG.C
TYP: funkcja transformacji
DEFINICJA: c_dr(degrees,rad)

```
  char *degrees,*rad;
```

PRZYKŁAD:

```
char *pntr,rad[6],deg[6],answer[17];
int status; pntr = "45";
c_afp(pntr,deg);
status = c_dr(deg,rad);
c_fasc(rad,answer);
printf ("%d %s",status,answer);
```

DZIAŁANIE: Zamienia wartość kąta wyrażoną w stopniach oraz ich częściach ułamkowych i umieszczoną w tablicy wskazanej pierwszym argumentem na wyrażoną w radianach i przypisuje ją zmiennej będącej drugim argumentem. Oba argumenty mogą być tą samą zmienną. Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

c_exp

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_exp(base,exponent,result)

```
 char *base,*exponent,*result;
```

PRZYKŁAD:

```
char *pntr,fp1[6],fp2[6],output[17];
int status; pntr = "2.12";
c_afp(pntr,fp1);
pntr = "7.89";
c_afp(pntr,fp2);
status = c_exp(fpi,fp2,fp2);
c_fasc(fp2,output);
printf("%d %s",status,output);
```

DZIAŁANIE: Podnosi liczbę wskazaną pierwszym argumentem do potęgi określonej przez drugi argument i umieszcza wynik w tablicy wskazanej drugim argumentem. Oba pierwsze argumenty mogą być tą samą zmienną, a trzeci argument może być również tą samą zmienną, co jeden z nich. Jeśli podczas obliczania funkcji

wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.
Funkcji c_exp odpowiada w Atari Basic potęgowanie (^).

c_fadd

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_fadd(a,b,sum)
```
char *a,*b,*sum;
```
PRZYKŁAD:
```
char *pntr,fp1[6],fp2[6],output[17];
int status;
pntr = "321.12";
c_afp(pntr,fp1);
pntr = "21.123";
c_afp(pntr,fp2);
status = c_fadd(fp1,fp2,fp2);
c_fasc(fp2,output);
printf ("%d %s",status,output);
```
DZIAŁANIE: Oblicza sumę dwóch liczb rzeczywistych wskazanych przez dwa pierwsze argumenty i umieszcza wynik w tablicy wskazanej trzecim argumentem. Oba pierwsze argumenty mogą być tą samą zmienną, a trzeci argument może być również tą samą zmienną, co jeden z nich. Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.
Funkcji f_add odpowiada w Atari Basic dodawanie (+).

c_fasc

PLIK: MATHLIB.C
TYP: funkcja transformacji
DEFINICJA: c_fasc(fp, atascii)
```
char *fp,*atascii;
```
PRZYKŁADY:
```
c_fasc(number,text);
c_fasc(temp,table);
```
DZIAŁANIE: Zamienia liczbą rzeczywistą umieszczoną w tablicy wskazanej pierwszym argumentem na ciąg znaków ATASCII i zapisuje go w tablicy wskazanej przez drugi argument. Tablica przeznaczona na wynik musi mieć co najmniej 17 znaków długości. Funkcja c_fasc nie zwraca statusu wykonanej operacji.

c_fdiv

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_div(dividend,divisor,result)
```
char *dividend,*divisor,*result;
```
PRZYKŁAD:
```
char *pntr,fp1[6],fp2[6],output[17];
int status;
```

```
pntr = "321.12";
c_afp(pntr,fp1);
pntr = "21.123";
c_afp(pntr,fp2) ;
status = c_fdiv(fp1,fp2,fp2);
c_fasc(fp2,output);
printf("%d %s",status,output);
```

DZIAŁANIE: Oblicza iloraz dwóch liczb rzeczywistych wskazanych przez dwa pierwsze argumenty i umieszcza wynik w tablicy wskazanej trzecim argumentem. Trzeci argument może być tą samą zmienną, co jeden z dwóch pierwszych. Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

Funkcji f_div odpowiada w Atari Basic dzielenie (/).

c_fmul

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_fmul(a,b,product)

```
char *a,*b,*product;
```

PRZYKŁAD:

```
char *pntr,fp1[6],fp2[6],output[17];
int status;
pntr = "321.12";
c_afp(pntr,fp1);
pntr = "21.123";
c_afp(pntr,fp2);
status = c_fmul(fp1,fp2,fp2);
c_fasc(fp2,output);
printf("%d %s",status,output) ;
```

DZIAŁANIE: Oblicza iloczyn dwóch liczb rzeczywistych wskazanych przez dwa pierwsze argumenty i umieszcza wynik w tablicy wskazanej trzecim argumentem. Oba pierwsze argumenty mogą być tą samą zmienną, a trzeci argument może być również tą samą zmienną, co jeden z nich. Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

Funkcji f_mul odpowiada w Atari Basic mnożenie (*).

c_fpi

PLIK: MATHLIB.C
TYP: funkcja transformacji
DEFINICJA: c_fpi(fp,integer)

```
char *fp;
int integer;
```

FRZYKŁADY:

```
c_fpi("60000",number);
c_fpi(string,value);
status = c_fpi(string,value);
```

DZIAŁANIE: Zamienia liczbę rzeczywistą umieszczoną w tablicy wskazanej pierwszym argumentem na liczbę całkowitą bez znaku (z zakresu od 0 do 65535) i przypisuje ją zmiennej będącej drugim

argumentem. Podczas konwersji podana liczba jest zaokrąglana do wartości całkowitej. Poprawna zamiana powoduje zwrócenie zera. Gdy liczba rzeczywista jest większa lub równa 65535.5, to funkcja zwraca wartość -1, a gdy ujemna, wartość -2.

UWAGA: Deep Blue C nie rozpoznaje liczb całkowitych bez znaku. Każda taka liczba większa od 32767 jest traktowana jako ujemna liczba całkowita. Oznacza to, że zamiast liczb od 32768 do 65535 uzyskamy liczby od -1 do -32768.

c_fpsi

PLIK: MATHLIB.C
TYP: funkcja transformacji
DEFINICJA: c_fpsi(float,integer)
int integer;
char *float;
PRZYKŁADY:

```
c_fpsi("60000",number);
c_fpsi(string,value);
status = c_fpsi(string,value);
```

DZIAŁANIE: Zamienia liczbą rzeczywistą umieszczoną w tablicy wskazanej pierwszym argumentem na liczbą całkowitą ze znakiem (z zakresu od -32768 do 32767) i przypisuje ją zmiennej będącej drugim argumentem. Podczas konwersji podana liczba jest zaokrąglana do wartości całkowitej. Poprawna zamiana powoduje zwrócenie zera. Gdy wartość absolutna liczby rzeczywistej jest większa lub równa 32767.5, to funkcja zwraca wartość —1.

c_frac

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_frac(nbr,fracpor)
char *nbr,*fracpor;
PRZYKŁAD:

```
char *pntr,nbr[6],fracp[6],answer[17];
int status;
pntr = "1234.5678";
c_afp(pntr,nbr);
status = c_frac(nbr,fracp);
c_fasc(fracp,answer);
printf ("%d %s",status,answer) ;
```

DZIAŁANIE: Oblicza część ułamkową liczby wskazanej przez pierwszy argument i umieszcza wynik w tablicy wskazanej drugim argumentem. Oba argumenty mogą być tą samą zmienną. Zwracany przez funkcję c_frac status nie sygnalizuje błędu, lecz specjalne przypadki działania funkcji. Jeśli wartość bezwzględna liczby jest mniejsza od 1, to zwracana jest wartość -1, gdy liczba jest zerem, wartość -2, a we wszystkich pozostałych przypadkach zero.

c_fsub

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_fsub(minuend,subtrahend,difference)
char *minuend,*subtrahend,*difference;
PRZYKŁAD:

```
char *pntr,fp1[6],fp2[6],output[17];
int status;
pntr = "321.12";
c_afp(pntr,fp1);
pntr = "21.123";
c_afp(pntr,fp2);
status = c_fsub(fp1,fp2,fp2);
c_fasc(fp2,output);
printf ("%d %s",status,output);
```

DZIAŁANIE: Oblicza różnicą dwóch liczb rzeczywistych wskazanych przez dwa pierwsze argumenty i umieszcza wynik w tablicy wskazanej trzecim argumentem. Trzeci argument może być tą samą zmienną, co jeden z dwóch pierwszych. Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.
Funkcji f_sub odpowiada w Atari Basic odejmowanie (-).

c_ifp

PLIK: MATHLIB.C
TYP: funkcjatransformacji
DEFINICJA: c_ifp(integer,fp)

```
int integer;
char *fp;
```

PRZYKŁAD:

```
char fpn[6],output[17];
int integer;
integer = 5000;
c_ifp(integer,fpn);
c_fasc(fpn,output);
printf ("%s",output) ;
```

DZIAŁANIE: Zamienia liczbą całkowitą bez znaku (z zakresu od 0 do 65535) będącą pierwszym argumentem na liczbą rzeczywistą umieszczoną w tablicy wskazanej przez drugi argument. Funkcja ta nie zwraca statusu wykonanej operacji.
UWAGA: Deep Blue C nie rozpoznaje liczb całkowitych bez znaku. Każda taka liczba wiąksza od 32767 jest traktowana jako ujemna liczba całkowita. Oznacza to, że zamiast liczb od 32768 do 65535 uzyskamy liczby od -1 do -32768.

c_iml

PLIK: MATHLIB.C
TYP: funkcja operacyjna
DEFINICJA: c_iml()
PRZYKŁAD:

```
c_iml();
```

DZIAŁANIE: Inicjuje stałe i zmienne konieczne do realizacji funkcji zawartych w bibliotece MATHLIB. Funkcja ta musi być wywołana przed pierwszym użyciem każdej innej funkcji z pliku MATHLIB.C.

c_int

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_int(nbr,intpor)
char *nbr,*intpor;
PRZYKŁAD:

```
char *pntr,nbr[6],intp[6],answer[17];
int status;
pntr = "1234.5678";
c_afp(pntr,nbr);
status = c_int(nbr,intp);
c_fasc(intp,answer);
printf ("%d %s",status,answer);
```

DZIAŁANIE: Oblicza część całkowitą liczby wskazanej przez pierwszy argument i umieszcza wynik w tablicy wskazanej drugim argumentem. Wynik jest zawsze mniejszy lub równy argumentowi. Oba argumenty mogą być tą samą zmienną. Zwracany przez funkcję c_int status nie sygnalizuje błędu, lecz specjalne przypadki działania funkcji. Jeśli liczba jest całkowita, to zwracana jest wartość -1, gdy jest zerem, wartość -2, a we wszystkich pozostałych przypadkach zero.

Funkcja c_int jest w przybliżeniu odpowiednikiem funkcji INT w Atari Basic.

c_itrig

PLIK: TRIG.C
TYP: funkcja operacyjna
DEFINICJA: c_itrig()
PRZYKŁAD:

```
c_itrig ();
```

DZIAŁANIE: Inicjuje stałe i zmienne konieczne do realizacji funkcji trygonometrycznych zawartych w bibliotece MATHLIB. Funkcja ta musi być wywołana przed pierwszym użyciem każdej innej funkcji z pliku TRIG.C, lecz po wywołaniu c_iml.

c_log

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_log(nbr, log)
char *nbr,*log;
PRZYKŁAD:

```
char *pntr,nbr[6],log[6],answer[17];
int status;
pntr = "256.512";
c_afp(pntr,nbr);
```

```
status = c_log(nbr,log);
c_fasc(log,answer) ;
printf("%d %s",status,answer);
```

DZIAŁANIE: Oblicza logarytm naturalny (przy podstawie e = 2.71828) liczby wskazanej przez pierwszy argument i umieszcza wynik w tablicy wskazanej drugim argumentem. Argumentem może być dowolna liczba większa od zera. Funkcją odwrotną do c_log jest c_alog. Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

Funkcja c_log jest dokładnym odpowiednikiem funkcji LOG w Atari Basic.

c_log10

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_log10(nbr,log10)

```
  char *nbr,*log10;
```

PRZYKŁAD:

```
char *pntr,nbr[6], log[6],answer[17];
int status;
pntr = "256.512";
c_afp(pntr,nbr);
status = c_log10(nbr,log);
c_fasc(log,answer);
printf ("%d %s",status,answer);
```

DZIAŁANIE: Oblicza logarytm dziesiętny (przy podstawie 10) liczby wskazanej przez pierwszy argument i umieszcza wynik w tablicy wskazanej drugim argumentem. Argumentem może być dowolna liczba większa od zera. Funkcją odwrotną do c_log10 jest c_alog10. Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

Funkcja c_log10 jest dokładnym odpowiednikiem funkcji CLOG w Atari Basic.

c_move

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_move(fpn1,fpn2)

```
  char *fpn1,*fpn2;
```

PRZYKŁAD:

```
char *pntr,fp1[6],fp2[6],output[17];
pntr = "321.'12";
c_afp(pntr,fp1);
c_move(fp1,fp2);
c_fasc(fp2,output);
printf ("%s",output) ;
```

DZIAŁANIE: Przepisuje liczbę rzeczywistą będącą pierwszym argumentem do tablicy wskazanej przez drugi argument. Funkcja ta nie zwraca statusu wykonanej operacji.

Funkcji c_move odpowiada w Atari Basic operacja przypisania wartości dwóch zmiennych.

c_rad

PLIK: TRIG.C
TYP: funkcja operacyjna
DEFINICJA: c_rad(i)
int i;
PRZYKŁADY:

```
c_rad(0);
c_rad(1);
c_rad(x);
```

DZIAŁANIE: Ustala jednostki (stopnie lub radiany), w których będą obliczane wszystkie następujące po niej funkcje trygonometryczne. Argument równy zero wybiera stopnie, a różny od zera radiany. Stan ten trwa, aż do następnego wywołania c_rad. Po wywołaniu funkcji c_itrig obliczenia są wykonywane w radianach.

Funkcja c_rad jest w przybliżeniu odpowiednikiem instrukcji DEG i RAD w Atari Basic.

c_rd

PLIK: TRIG.C
TYP: funkcja transformacji
DEFINICJA: c_rd(rad,degrees)
char *rad,*degrees;
PRZYKŁAD:

```
char *pntr,rad[6],deg[6],answer[17];
int status;
pntr = "0.78539816";
c_afp(pntr,rad);
status = c_rd(rad,deg);
c_fasc(deg,answer);
printf("%d %s",status,answer);
```

DZIAŁANIE: Zamienia wartość kąta wyrażoną w radianach i umieszczoną w tablicy wskazanej pierwszym argumentem na wyrażoną w stopniach i ich częściach ułamkowych oraz przypisuje ją zmiennej będącej drugim argumentem. Oba argumenty mogą być tą samą zmienną. Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

c_sifp

PLIK: MATHLIB.C
TYP: funkcja transformacji
DEFINICJA: c_sifp(sint,fp)
int sint;
char *fp;
PRZYKŁAD:

```
char fpn[6],output[17];
int integer;
integer = -5000;
c_sifp(integer,fpn);
c_fasc(fpn,output);
printf("%s",output);
```

DZIAŁANIE: Zamienia liczbę całkowitą ze znakiem (z zakresu od -32768 do 32767) będącą pierwszym argumentem na liczbą rzeczywistą umieszczoną w tablicy wskazanej przez drugi argument. Funkcja ta nie zwraca statusu wykonanej operacji.

c_sin

PLIK: TRIG.C
TYP: funkcja trygonometryczna
DEFINICJA: c_sin(angle,result)
char *angle,*result;
PRZYKŁAD:

```
char *pntr,nbr[6],sine[6],answer[17];
int status; rad(0);
pntr = "30";
c_afp(pntr,nbr);
status = c_sin(nbr,sine);
c_fasc(sine,answer);
printf ("%d %s",status,answer) ;
```

DZIAŁANIE: Oblicza wartość funkcji sinus pierwszego argumentu i umieszcza ją w tablicy wskazanej drugim argumentem. Pierwszy argument może mieć dowolną wartość dozwoloną w Atari. Wynikiem jest liczba z przedziału od -1 do +1, a jej wartość zależy od jednostek, w których wykonywane jest obliczenie (stopnie lub radiany - patrz opis funkcji c_rad). Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

Funkcja c_sin jest dokładnym odpowiednikiem funkcji SIN w Atari Basic.

c_sqrt

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_sqrt(nbr,sqr)
char *nbr,*sqr;
PRZYKŁAD:

```
char *pntr,nbr[6],sqr[6],answer[17];
int status;
pntr = "256.512";
c_afp(pntr,nbr);
status = c_sqrt(nbr,sqr);
c_fasc(sqr,answer);
printf ("%d %s" ,status,answer);
```

DZIAŁANIE: Oblicza pierwiastek kwadratowy liczby wskazanej przez pierwszy argument i umieszcza wynik w tablicy wskazanej drugim argumentem. Poprawny przebieg obliczenia powoduje zwrócenie zera. Gdy pierwiastkowana liczba jest ujemna, to funkcja zwraca wartość -2, a gdy wynik jest poza dopuszczalnym zakresem, wartość -1.

Funkcja c_sqrt jest dokładnym odpowiednikiem funkcji SQR w Atari Basic.

c_tan

PLIK: TRIG.C
TYP: funkcja trygonometryczna
DEFINICJA: c_tan(angle,result)
char *angle,*result;
PRZYKŁAD:

```
char *pntr,nbr[6],tangent[6],answer[17];
int status; rad(0);
pntr = "30";
c_afp(pntr,nbr);
status = c_tan(nbr,tangent);
c_fasc(tangent,answer);
printf("%d %s",status,answer);
```

DZIAŁANIE: Oblicza wartość funkcji tangens pierwszego argumentu i umieszcza ją w tablicy wskazanej drugim argumentem. Pierwszy argument może mieć dowolną wartość dozwoloną w Atari. Wynik zależy od jednostek, w których wykonywane jest obliczenie (stopnie lub radiany - patrz opis funkcji c_rad). Jeśli podczas obliczania funkcji wystąpi błąd, to zwracana jest wartość -1, a w przeciwnym przypadku zero.

c_zero

PLIK: MATHLIB.C
TYP: funkcja arytmetyczna
DEFINICJA: c_zero(fpn)
char *fpn;
PRZYKŁAD:

```
char fpn[6],answer[17];
c_zero(fpn);
c_fasc(fpn,answer);
printf ("%s",answer);
```

DZIAŁANIE: Wpisuje wartość zero do tablicy wskazanej przez argument, a zdefiniowanej jako liczba rzeczywista. Funkcja ta nie zwraca statusu wykonanej operacji.

Funkcji c_zero odpowiada w Atari Basic operacja przypisania zmiennej wartości zero.

2. 8. Meldunki błędów

Kompilator Deep Blue C sygnalizuje błędy składniowe wykryte podczas kompilacji przez wskazanie miejsca wystąpienia i podanie skróconego opisu. Brakujące definicje zmiennych i funkcji są wykrywane dopiero przez linker. Podczas scalania programu mogą także wystąpić błędy spowodowane zbyt dużym rozmiarem programu lub zastosowaniem konstrukcji składniowych, których linker nie potrafi zrozumieć.

W czasie wykonywania programu mogą pojawić się błędy spowodowane użyciem niewłaściwych wartości lub wykonaniem instrukcji, które są w aktualnej chwili niepoprawne. Błędy te, zwane błędami wykonania, są sygnalizowane dopiero podczas działania programu.

Poniżej przedstawione są meldunki błędów sygnalizowanych przez kompilator oraz ich krótki opis.

already defined
: Funkcja lub zmienna o podanej nazwie została już wcześniej zdefiniowana.

can't nest include file
: Pliki dołączane z zewnątrz nie mogą być zagnieżdżone, to znaczy, że w pliku dołączanym dyrektywą include niedozwolona jest taka dyrektywa.

can't subscript
: Podana wartość nie może być indeksem tablicy, zmienna nie może mieć indeksu lub użycie indeksu w tym miejscu jest niedopuszczalne.

'case' or 'default' expected
: W tym , miejscu instrukcji snitch powinna się znajdować instrukcja case lub default.

constant needed
: Brak stałej w instrukcji case lub podana wartość nie jest stałą.

do with no while
: Instrukcja do nie została zakończona przez odpowiednią instrukcję while.

expected comma
: W tym miejscu instrukcji powinien znajdować się przecinek.

expecting argument name
: W tym miejscu instrukcji powinna znajdować się nazwa argumentu.

global symbol table overflow
: Tablica zmiennych globalnych została całkowicie zapełniona (zbyt dużo zmiennych).

illegal address
   Niedozwolona wartość argumentu dla operatora adresowego (&).

illegal argument name
   Niedozwolona nazwa argumentu.

illegal function or declaration
   Niedozwolona funkcja lub deklaracja.

illegal symbol name
   Niedozwolona nazwa zmiennej, funkcji lub operatora.

invalid expression
   Nieprawidłowe wyrażenie - nadmiar lub brak nawiasu, operatora lub zmiennej.

line too long
   Zbyt długi wiersz programu źródłowego.

local symbol table overflow
   Tablica zmiennych lokalnych została całkowicie zapełniona (zbyt dużo zmiennych).

macro table full
   Tablica definicji funkcji została całkowicie zapełniona (zbyt dużo funkcji).

missing apostrophe
   Brak apostrofu ograniczającego znak (stałą znakową).

missing bracket
   Brak nawiasu "$(" rozpoczynającego definicję funkcji.

missing closing bracket
   Instrukcja złożona lub definicja funkcji nie została zakończona symbolem "$)".

missing colon
   Brak dwukropka wymaganego po instrukcjach case i default.

missing final end
   W instrukcji złożonej brak zamykającego nawiasu "$)".

missing open paren
   Brak nawiasu otwierającego po nazwie funkcji.

missing quote
   Brak cudzysłowu ograniczającego stałą tekstową (ciąg znaków).

missing quote or <
   Brak cudzysłowu lub znaku "<" rozpoczynającego specyfikacją pliku w dyrektywie include.

missing quote or >
   Brak cudzysłowu lub znaku ">" kończącego specyfikacją pliku w dyrektywie include.

missing semicolon
   Opuszczony został średnik kończący instrukcją.

must be constant
   W tym miejscu instrukcji wymagana jest stała.

must be value
   W tym miejscu instrukcji wymagana jest wartość.

needed address
   Instrukcja asm została użyta bez adresu procedury.

negative size illegal
   Niedozwolony jest ujemny rozmiar tablicy.

no active whiles
   W tym miejscu programu nie ma żadnej aktywnej pętli while.

open failure on include file
   Nie można otworzyć pliku wskazanego przez dyrektywą include.

string space exhausted
   Brak miejsca w obszarze przeznaczonym na tablice znakowe i liczbowe (zbyt dużo tablic).

too many active whiles
   Zostało równocześnie otwarte zbyt dużo pętli while.

wrong number args
   Funkcja lub operacja ma niewłaściwą liczbą argumentów.

Poniżej przedstawione są meldunki błądów sygnalizowanych przez linker oraz ich krótki opis.

bad byte code
   Błądny kod bajtu.

bad CCC code
   Błądny kod w pliku CCC.

bad CCC file
   Błądny format pliku CCC.

bad op code
   Błądny kod operatora.

can't find
   Brak wskazanego pliku.

can't read file
   Wskazanego pliku nie można odczytać.

can't write
   Wskazanego pliku nie można zapisać

CCC file too large
   Plik CCC jest zbyt duży.

never defined:
   Wymienione nazwy nie zostały nigdzie zdefiniowane.

no main ()
   W programie brak funkcji o nazwie *ain().

too many globals
   Użyto zbyt dużo zmiennych globalnych.

unknown file type
   Wskazany plik ma niewłaściwy format.

Błędy wykonania sygnalizowane są meldunkiem:

```
dbc 1 run—time—error ... type a
key to return to DOS.
```

Znajdujące się w tym meldunku literowe symbole błędów mają następujące znaczenie:

A - stack overflow RAMTOP
   Brak miejsca na stosie bieżącym programu, za dużo zmiennych lub zbyt wiele zagnieżdżeń pętli.

B - illegal op-code
   Program uszkodził własną treść i próbował wykonać niepoprawne operacje.

C - version error
   Użyte do tworzenia programu pliki CC.COM, CLINK.COM i DBC.OBJ mają różne numery wersji i nie stanowią zgodnej całości.

D - divide by zero
   W programie wystąpiło dzielenie przez zero lub obliczenie reszty z takiego dzielenia.

Rozdział 3
ACTION!

Język Action! powstał w roku 1983 w amerykańskiej firmie Optimized Systems Software. Jest to język programowania zaprojektowany specjalnie dla ośmiobitowych komputerów Atari i będący językiem pośrednim między Pascalem i C. System Action! umieszczony jest na cartridge'u, który zawiera monitor, edytor, kompilator i bibliotekę. Ponadto dostępna jest dyskietka "Action! Tool Kit" zawierająca dodatkowe procedury, które można dołączyć do własnego programu. Oferowane w Polsce wersje dyskowe i kasetowe Action! są pirackimi kopiami. Nie mają one pełnych możliwości i dają użytkownikowi o 8 KB pamięci mniej.

Action! nie ma swojego odpowiednika dla innych komputerów. Nie ma więc podręczników programowania w tym języku. Jedyną dostępną książką poświęconą tej tematyce jest dołączana do cartridge a instrukcja pod tytułem "The ACTION! System" wydana przez OSS. Ponadto kurs programowania w tym języku publikowany jest w "Bajtku — Tylko o Atari", a wiele ciekawych procedur można znaleźć w zwykłych numerach "Bajtka".

Action! uruchamia się automatycznie po włączeniu komputera. Jeśli w zestawie znajduje się stacja dysków, to ze znajdującej się w niej dyskietki odczytywany jest jeszcze DOS. Po uruchomieniu zgłasza się edytor Action!. Sposób przejścia z edytora do monitora i odwrotnie jest podany w rozdziałach opisujących te części systemu. Programy napisane w Action! mogą być uruchamiane (nawet po kompilacji) tylko wtedy, gdy cartridge znajduje się w gnieździe. Istnieje jednak możliwość uruchomienia tych programów samodzielne po dołączeniu do nich pliku runtime.

Cartridge Action! zajmuje obszar pamięci od adresu 40960 ($A000) do 49151 ($BFFF). Poniżej znajduje się pamięć obrazu. Cała pamięć operacyjna od szczytu DOS—u (lub od adresu 1792, gdy nie ma DOS—u) do początku palnięci obrazu jest przeznaczona dla systemu. Do dyspozycji użytkownika pozostaje więc tylko szósta strona pamięci RAM (1536-1791 = $0600-$06FF). Duża szybkość wykonywania programu czyni jednak zbędne stosowanie procedur w języku maszynowym, a konieczne, stosunkowo niewielkie fragmenty kodu maszynowego można włączać bezpośrednio w treść programu albo umieszczać w tablicach znakowych o dowolnym lub ustalonym położeniu w pamięci.

Sercem Action! jest monitor. Zarządza on pracą pozostałych części systemu: edytora, kompilatora i biblioteki. Dwa ostatnie elementy są bezpośrednio niedostępne dla użytkownika i korzysta się z nich poprzez monitor, natomiast edytor wykorzystuje się bezpośrednio po wywołaniu go z monitora.

Kompilator Action! normalnie nie rozróżnia małych i dużych liter, pozwala więc na dowolne wpisywanie treści programu. Można jednak zmienić tą. sytuację korzystając z menu wariantów monitora.

Po uruchomieniu Action! edytor zgłasza się automatycznie, a w ostatnim wierszu ekranu pojawia sie napis "ACTION! (c) 1983 ACS". Wiersz ten służy do wyświetlania komunikatów i wpisywania poleceń redakcyjnych dla edytora. W przypadku równoczesnego korzystania z dwóch okien wiersz komunikatów znajduje sie pomiędzy nimi.

Edytor Action! jest zwykłym edytorem tekstowym i jako taki posiada wiele specjalnych funkcji redakcyjnych. Większość z nich uzyskuje sie przez równoczesne naciskanie innych klawiszy oraz <CONTR0L> i <SHIFT>. Niektóre dostępne są przy naciśnięciu <CONTROL> z innymi klawiszami. Niemal wszystkie funkcje redakcyjne powodują wyświetlenie pomocniczych informacji w wierszu komunikatów, a także wymagają wprowadzenia odpowiednich danych. Poszczególne funkcje edytora są wywoływane przez następujące kombinacje klawiszy:

Ruch kursora

<CTRL><↑>	kursor o 1 wiersz w górę
<CTRL><↓>	kursor o 1 wiersz w dół
<CTRL><→>	kursor o 1 znak w prawo
<CTRL><←>	kursor o 1 znak w lewo
<CTRLXSHIFT><<>	kursor na początek wiersza
<CTRLXSHIFT><>>	kursor na koniec wiersza
<RETURN>	kursor do następnego wiersza
<TAB>	tabulae ja kursora
<SHIFTXTAB>	ustawienie tabulaeji
<CTRLXTAB>	skasowanie tabulae j i

Ruch okna ekranu

<CTRL><SHIFT><H>	head - kursor na początek pliku
<CTRL><SHIFT><↑>	okno o 1 ekran w górę
<CTRL><SHIFT><↓>	okno o 1 ekran w dół
<CTRL><SHIFT><]>	okno o 1 znak w lewo
<CTRL><SHIFT><[>	okno o 1 znak w prawo
<CTRL><SHIFT><2>	utworzenie drugiego okna
<CTRL><SHIFT><1>	przejście kursora do pierwszego okna
<CTRL><SHIFT><2>	przejście kursora do drugiego okna
<CTRL><SHIFT><D>	delete - kasowanie okna

Redagowanie tekstu

<CTRL><SHIFT><I>	insert - wstawianie/wymiana
<CTRL><SHIFT><U>	undone - odtworzenie zmienionego wiersza
<CTRL><SHIFT><P>	paste - odtworzenie usuniętego wiersza
<SHIFT><DELETE>	zapamiętanie bloku tekstu <CTRL><SHIFT><P> paste - wstawienie tekstu z bufora <CTRL><SHIFT><F> find - wyszukiwanie łańcucha znaków <CTRL><SHIFT><S> substitute - wymiana łańcucha znaków <CTRL><SHIFT><RETURN> podzielenie wiersza na dwa, <CTRL><SHIFT><DELETE> połączenie dwóch wierszy <CTRL><SHIFT><T> tag set - ustawienie etykiety
<CTRL><SHIFT><G>	go to tag - odszukanie etykiety
<SHIFT><CLEAR>	kasowanie całej zawartości edytora

Polecenia I/O

<CTRL><SHIFT><R> read - odczyt pliku
<CTRL><SHIFT><W> write - zapis pliku

<CTRL><SHIFT><M> monitor - przejście do monitora

Działanie klawiszy służących do przemieszczania kursora wewnątrz redagowanego tekstu jest oczywiste i nie wymaga specjalnego opisu. Dotyczy ta również ruchu okien edytora.

Kombinacja klawiszy <CONTROL><BHIFT><I> przełącza tryby pracy edytora. Po każdym jej naciśnięciu w wierszu komunikatów wyświetlany jest napis "INSERT" (wstawianie) lub "REPLACE" (wymiana).

Wiersz, w którym dokonywane są zmiany, jest zapamiętywany i można odtworzyć jego poprzednią treść przez naciśnięcie klawiszy <CONTROL><SHIFT><U>. Wykonanie tej operacji jest niemożliwe, jeśli w międzyczasie kursor opuścił zmieniany wiersz.

Klawisze <SHIFT> i <DELETE> powodują usunięcie wiersza, w którym znajduje się kursor i umieszczenie go w buforze. Poprzednia zawartość bufora jest przy tym kasowana, chyba że ta kombinacja klawiszy jest używana raz po razie. W ten sposób można więc skasować większy blok tekstu.

Znajdujący się w buforze tekst można wstawić na aktualnej pozycji kursora przy użyciu klawiszy <CONTROL><SHIFT><P>. Zawartość bufora pozostaje przy tym bez zmiany.

Klawisze <CONTROL><SHIFT><F> służą do przeszukiwania redagowanego tekstu. Poszukiwany ciąg znaków wpisuje się w wierszu komunikatów na polecenie "Find?". Przeszukiwanie przebiega od aktualnego położenia kursora do końca tekstu. Po znalezieniu ciągu kursor ustawiany jest na jego pierwszym znaku. Kolejne wystąpienie ciągu jest wskazywane po następnym naciśnięciu klawiszy. Brak poszukiwanego ciągu jest sygnalizowany komunikatem "not found".

Klawisze <CONTROL><SHIFT><S> pozwalają na wymianę fragmentu tekstu. Po ich naciśnięciu należy, na pytanie "Substitute?", podać nowy ciąg znaków, a następnie, na pytanie "For?", ciąg znaków, który ma być zastąpiony. Działanie funkcji jest zbliżone do funkcji wyszukiwania, czyli wykonywana jest tylko jedna wymiana, a dalsze wymagają ponownego użycia klawiszy.

Jeżeli zachodzi potrzeba podzielenia wiersza programu na dwa oddzielne, to realizuje się to przez ustawienie kursora na znaku, od którego ma się zaczynać drugi wiersz i naciśnięcie klawiszy <CQNTROL><SHlFT><RETURN>. Odwrotna operacja jest wykonywana po ustawieniu kursora na pierwszym znaku drugiego wiersza i naciśnięciu klawiszy <CONTROL><SHIFT><DELETE>.

Klawisze <CONTROLXSHIFTXT> powodują ustawienie etykiety w wierszu, w którym aktualnie znajduje się kursor. Symbolem tej etykiety jest litera wpisana na pytanie "tag id:". Skasowanie etykiety następuje po dokonaniu jakiejkolwiek zmiany w wierszu z etykietą oraz po ustawieniu innej etykiety oznaczonej tą samą 1 i terą.

Skok do etykiety następuje po naciśnięciu klawiszy <CONTROL><SHIFT><G> i podaniu symbolu wybranej etykiety na pytanie "tag id:". Jeśli wskazana etykieta nie istnieje, to wyświetlany jest komunikat "tag not set".

Całą zawartość edytora można usunąć naciskając klawisze <SHIFT> i <CLEAR>. Ola uniknięcia pomyłek funkcja ta wymaga potwierdzenia <Yes) na pytanie "CLEAR?". Jeżeli treść programu nie była zapisywana po wykonaniu ostatniej zmiany, to wyświetlane jest żądanie powtórnego potwierdzenia: "Not saved — Delete?".

Naciśnięcie klawiszy <CONTROL><SHIFT><R> umożliwia odczytanie pliku z urządzenia zewnętrznego. Zawartość tego pliku jest dołączana do tekstu znajdującego się w edytorze począwszy od aktualnej pozycji kursora. Można dzięki temu korzystać z gotowych podprogramów i procedur, co znacznie przyspiesza pracę. Przy podawaniu nazwy pliku można pominąć symbol urządzenia, przyjmowane jest wtedy urządzenie "D:". Wpisanie znaku zapytania zamiast nazwy urządzenia powoduje natomiast odczyt katalogu dyskietki. Na przykład, "?2:*.*" odczytuje katalog dyskietki ze stacji numer 2.

Zapis całej zawartości edytora lub tego z okien, w którym znajduje się kursor, uzyskuje się przez naciśnięcie klawiszy <CONTROL><SHIFT><W>. Zapis dokonywany jest w postaci pliku tekstowego zawierającego znaki ATASCII, czyli dokładnie w takiej postaci, jaka jest widoczna na ekranie. Podanie nazwy zapisywanego pliku "P:" powoduje wydrukowanie zawartości edytora na drukarce.

Opuszczenie edytora i przejście do monitora Action! jest możliwe tylko przez użycie klawiszy <CONTROL><SHIFT><M>. Zawartość edytora nie ulega przy tym zniszczeniu.

Parametry pracy edytora mogą być zmieniane przy użyciu funkcji "Options" monitora. Opis znaczenia tych parametrćw i ich wartości znajduje się w rozdziale 3.2. "Monitor".

3. 2. Monitor

Po przejściu do monitora Action! w górnym wierszu ekranu pojawia się znak ">". Wiersz ten służy do wprowadzania poleceń, a pozostała część ekranu stanowi obszar komunikatów. Monitor pozwala na sterowanie pozostałymi elementami systemu. Służące do tego celu polecenia wpisuje się jako jednoliterowe symbole z odpowiednimi parametrami i zatwierdza klawiszem <RETURN>.

Monitor Action! rozpoznaje następujące polecenia:

Boot	restart Action!
Dos	przejście do DOS-u
Editor	przejście do edytora
Options	menu wariantów
eXecute	wykonanie instrukcji
Compile	kompilacja programu
Write	zapis programu
Run	uruchomienie programu
Proceed	kontynuacja programu
Set	zmiana zawartości komórki
?	odczyt zawartości komórki
*	przegląd zawartości pamięci

Boot powoduje ponowne uruchomienie systemu Action! i skasowanie wszystkich zawartych w pamięci programów.

Dos przerywa pracę systemu, kasuje wszystkie programy znajdujące się w pamięci i powoduje przejście do DOS—u. Jeżeli w systemie nie ma stacji dysków, to następuje przejście do programu testującego.

Editor wywołuje edytor Action!. Gdy podczas kompilacji programu zawartego w edytorze wystąpił błąd, to po przejściu do edytora kursor ustawiony jest w miejscu napotkania tego błędu.

Options pozwala na ustalenie wariantów pracy systemu Action!. Kolejne parametry wyświetlane są teraz w wierszu poleceń. Użytkownik może ustalić żądane wartości parametrów lub pozostawić je bez zmian. Dostępne są następujące warianty (w nawiasie standardowe ustawienie):

Display? (Y) — wyświetlanie obrazu podczas kompilacji. Po ustawieniu na N obraz jest wygaszany na czas trwania kompilacji, co przyspiesza jej przebieg.

Bell? (Y) — sygnalizowanie błędów i niektórych poleceń dźwiękiem buczka. W wyłącza buczek.

Case sensitive? (N) - rozróżnianie przez kompilator dużych i małych liter. Po Y nazwy instrukcji muszą być pisane dużymi literami.

Trace? (N) - śledzenie wykonywanego programu. Podczas realizacji programu po ustawieniu Y nazwa każdej wywoływanej procedury oraz jej parametry wyświetlane są w polu komunikatów monitora.

List? (N) - listowanie kompilowanego programu. Podczas kompilacji po Y w polu komunikatów wyświetlany jest każdy kompilowany wiersz programu.

Window 1 size? (18) - liczba wierszy, w pierwszym oknie edytora podczas pracy w dwóch oknach. Dozwolone są wartości z zakresu od S do 18. Oba okna mają razem zawsze 23 wiersze.

Line size? (120) - liczba znaków w wierszu podczas druku na drukarce (oprócz marginesu). Przekroczenie ustalanej długości wiersza jest sygnalizowane buczkiem. Maksymalna długość wiersza w edytorze wynosi 240 znaków.

Left margin? (2) — liczba znaków tworzących lewy margines wiersza na ekranie lub drukarce. Dozwolone są wartości z zakresu od p do 39.

EOL character? (spacja) — symbol wyświetlany jako znak końca wiersza. Normalnie jest on niewidoczny, a zalecane jest stosowanie serca (CONTROL-.) lub kółka (CONTROL-T).

eXecute nakazuje wykonanie instrukcji, procedury lub dyrektywy kompilatora podanej po poleceniu.

Compile wywołuje kompilator Action! i wykonuje kompilację programu zawartego w pliku, którego nazwa jest podana po poleceniu. Jeśli nie została podana żadna nazwa, to kompilowany jest program znajdujący się w pamięci. Skompilowany program umieszczany jest w pamięci począwszy od ostatniego adresu zajmowanego przez zawartość edytora. Adres ten można jednak zmienić - patrz opis słowa "SET".

Write zapisuje w pliku binarnym o podanej nazwie skompilowany program znajdujący się w pamięci komputera. Program taki można uruchomić bezpośrednio z poziomu OOS-u (jeśli cartridge jest dołączony).

Run powoduje uruchomienie skompilowanego programu. Możliwe są tu cztery warianty:

—odczytanie i uruchomienie programu znajdującego się w pliku a podanej nazwie;

—uruchomienie programu znajdującego się w pamięci od podanego adresu;

—wywołanie i wykonanie znajdującej się w pamięci procedury o podanej nazwie;

— uruchomienie całego programu znajdującego się w pamięci (polecenie bez parametru).

Proceed nakazuje kontynuację programu przerwanego przez wywołanie procedury Break lub przez naciśnięcie klawisza <BREAK>.

Set powoduje wpisanie wartości będącej drugim parametrem polecenia do komórki pamięci, której adres jest pierwszym parametrem. Parametry muszą być oddzielone od siebie przecinkiem.

? wyświetla w polu komunikatów zawartość komórki pamięci wskazanej przez parametr polecenia. Parametrem może być zarówno adres komórki, jak i nazwa lub wskaźnik zmiennej. W skład odpowiedzi wchodzą kolejno: adres dziesiętny i szesnastkowy, znak ATASCII, dwubajtowa wartość szesnastkowa oraz 'jedno- i dwubajtowa wartość dziesiętna.

* powoduje przegląd zawartości komórek pamięci od miejsca wskazanego parametrem. Parametr polecenia i format wyświetlanego wyniku jest taki, jak w poleceniu ?. Przerwanie przeglądania pamięci następuje po naciśnięciu klawisza spacji.

3 . 3. Technika programowania

Struktura programu w języku Action! zbliżona jest do programu w Pascalu, lecz ma większą elastyczność. Pod tym względem Action! jest znacznie bliższy językowi C.

Każdy program w Action! składa się z definicji zmiennych, procedur i funkcji. Poszczególne wiersze programu nie są numerowane. Instrukcje są oddzielane od siebie spacjami. Liczba instrukcji w wierszu jest dowolna, lecz długość wiersza nie może przekraczać 240 znaków. Podział programu na wiersze nie ma znaczenia dla kompilatora, lecz służy wyłącznie do poprawienia czytelności i zrozumiałości programu.

Kolejność umieszczenia w programie deklaracji jego elementów musi być taka, aby każda zmienna, funkcja i procedura była zdefiniowana przed użyciem. Definicje te nie mogą być zagnieżdżane (nie mogą znajdować się wewnątrz innych definicji). Wykonywanie programu rozpoczyna się zawsze od ostatniej zawartej w programie procedury lub funkcji.

Nazwy wszystkich procedur i funkcji są dowolne. Po nazwie należy umieścić w nawiasie okrągłym nazwy przekazywanych parametrów wraz z ich deklaracjami. Jeśli nie są wymagane żadne parametry, to po nazwie procedury lub funkcji umieszcza się tylko pusty nawias "()". Ponadto nazwa funkcji musi być poprzedzona deklaracją typu wartości zwracanej przez tą funkcję. Definicja procedury lub funkcji kończy się słowem "RETURN".

W Action! mogą występować tylko instrukcje proste, czyli polecenia wykonania pojedynczych operacji, których opisy znajdują się w słowniku. Gdy zachodzi konieczność użycia instrukcji złażonej (zawierającej kilka instrukcji prostych), to trzeba ją zdefiniować w postaci procedury.

Komentarze mogą być umieszczane w dowolnym miejscu programu i są oznaczane średnikiem. Wszystko, co znajduje się w wierszu programu między średnikiem i końcem wiersza, jest ignorowane przez kompilator. Komentarze mogą więc zawierać dowolne znaki.

Action! umożliwia łatwe korzystanie z procedur systemu operacyjnego oraz proste dołączanie do programów krótkich procedur w języku maszynowym. Dostęp do procedur systemowych i innych procedur maszynowych uzyskuje się przez określenie adresu w definicji procedury (patrz opis słowa "PROC"). Natomiast krótkie fragmenty kodu maszynowego umieszcza się w dowolnych miejscach programu w postaci bloków kodu.

Blok kodu jest to zamknięty w nawiasach kwadratowych ciąg wartości, które mogą być dowolnymi stałymi, zmiennymi i stałymi kompilatora. Kompilator po napotkaniu bloku kodu przenosi zawarte w nim wartości bezpośrednio do tworzonego programu (kodu) wynikowego.

3. 4. Wartości

W Action! występują stałe i zmienne różnych typów. Muszą one być: zawsze zadeklarowane przed użyciem. Powinno się więc umieszczać deklaracje wartości globalnych na początku programu, procedury lub funkcji. W dalszej części tego rozdziału opisane są wszystkie rodzaje wartości dostępne w Action! oraz niektóre zależności między nimi.

Stałe

Stałe są wartościami definiowanymi przez programistę i występującymi najczęściej w postaci jawnej. Możliwe jest także użycie stałych w postaci nazw symbolicznych (stałych symbolicznych), które symbolizują wartości tych stałych w całym programie i są zamieniane na wartości stałe w procesie kompilacji. Dzięki temu znacznie łatwiejsze jest modyfikowanie programu w przypadku, gdy konieczna jest zmiana tych wartości. Ponadto użycie stałych symbolicznych znacznie zwiększa czytelność programu.

W Action! możliwe jest użycie kilku rodzajów stałych. Dpis typów tych stałych znajduje się poniżej.

Liczby dziesiętne są to liczby całkowite bez znaku z zakresu od 0 do 65535 (typ CARD) lub liczby całkowite ze znakiem z zakresu od -32768 do 32767 (typ INT). Liczby dziesiętne nie wymagają specjalnego oznaczania.

Liczby szesnastkowe są oznaczone przez umieszczenie na początku znaku "$" i mają także wartości całkowite bez znaku lub ze znakiem. Zakres ich wartości wynosi odpowiednio od $0000 do $FFFF lub od -$8000 do $7FFF.

Stałe znakowe są poprzedzonymi apostrofem znakami kodu ASCII (np.: 'a, '3, '%) i reprezentują wartości liczbowe tego kodu (od 0 do 255). Stałe znakowe mogą być więc użyte w obliczeniach jako liczby.

Stałe tekstowe są ciągami znaków kodu ATASCII ujętymi w cudzysłowy. Ich użycie jest dozwolone jedynie w dyrektywach DEFINE, procedurach Print oraz przy nadawaniu wartości tablicom znakowym.

Ponadto w Action! występują wartości stosowane podczas kompilacji i zwane stałymi kompilatora. Są one używane do ustawiania atrybutów zmiennych, procedur i funkcji. Dozwolone są cztery formy stałych kompilatora: stałe liczbowe, identyfikatory, wskaźniki oraz sumy poprzednio wymienionych.

Identyfikator użyty jako stała kompilatora ma wartość odpowiadającą wartości adresu tego identyfikatora w pamięci. Wskaźnik natomiast ma wartość adresu wskazywanej zmiennej.

Wartości zmienne, które są używane w programie, muszą być przed użyciem zadeklarowane. Deklaracja zmiennej określa jej typ

i nazwę (identyfikator) oraz ewentualnie wartość początkową lub adres, pod którym zmienna będzie umieszczona w pamięci. W Action! dozwolone są zmienne typu dwubajtowego bez znaku {CARD), dwubajtowego ze znakiem {INT), jednobajtowego {BYTE) i znakowego {CHAR) oraz wskaźniki {POINTER), tablice {ARRAY) i rekordy {TYPE). Zmienne typów BYTE i CHAR są sobie równoważne i można je stosować zamiennie.

Nazwa zmiennej może składać się z liter, cyfr i znaku podkreślenia (_). Pierwszym znakiem nazwy musi być litera. Action! może rozróżniać duże i małe litery (zależnie od ustawienia), zwyczajowo przyjęte jest pisanie nazw zmiennych małymi literami, nazw procedur i funkcji małymi i dużymi, a słów kluczowych dużymi. Nazwa zmiennej może mieć dowolną długość i wszystkie jej znaki są rozróżniane przez kompilator.

W języku Action! przestrzegana jest lokalność zmiennych. Każda zmienna jest określona w tej procedurze lub funkcji, w której zastała zdefiniowana oraz w procedurach przez nią wywołanych. Zmiennymi globalnymi są zmienne zdefiniowane przed wszystkimi definicjami procedur i funkcji oraz po dyrektywie "MODULE". Zmienne zdefiniowane wewnątrz procedur i funkcji są lokalne.

Tablice

Tablicą jest zmienna zawierająca kilka elementów danego typu. Dozwolone są tablice złożone z elementów czterech typów podstawowych (CHAR, BYTE, CARD i INT). Wielkość tablicy jest określana przez wartości indeksowe, które mogą mieć dowolną wartość całkowitą. Rzeczywista wielkość tablic zależy jednak od wielkości pozostałego do dyspozycji obszaru pamięci.

Definicja tablicy jest bardzo zbliżona do definicji zmiennej prostej: składa się z nazwy typu, słowa "ARRAY" i nazwy tablicy oraz ewentualnie jej adresu, rozmiaru lub wartości początkowych. Elementy te mogą wystąpić w dowolnej kombinacji, lecz — w odróżnieniu od innych języków programowania — w Action! nie jest wymagane określenie rozmiaru deklarowanej tablicy.

Elementy tablicy są zawsze numerowane od O, a rozmiar tablicy określa liczbę elementów. W celu wskazania elementu tablicy należy podać nazwę tej tablicy, a po niej wartość indeksu ujętą w nawias okrągły. Na przykład tablica zdefiniowana jako "tab(10)" zawiera elementy od "tab(0)" do "tab(9)".

Wskaźniki

Wskaźnik jest zmienną, która zawiera adres innej zmiennej typu podstawowego (BYTE, CHAR, CARD lub INT). Wskaźniki nie są odrębnym typem, gdyż przyjmują wartości odpowiadające liczbom całkowitym bez znaku (od 0 do 65535), a więc typu CARD. Trzeba dla nich natomiast określić typ wskazywany.

Definitja wskaźnika składa się z nazwy typu wskazywanego, słowa "POINTER" i nazwy wskaźnika oraz ewentualnie jego wartości (wskazywanego adresu lub zmiennej). Nadanie wartości wskaźnikowi

lub jej zmiana mogą być ponadto zrealizowane w dowolnym miejscu programu. Ponadto wskaźnik zmiennej może być używany zamiast nazwy zmiennej. W tym przypadku do nazwy wskaźnika trzeba dodać na końcu znak "^".

Rekordy

Rekord jest zbiorem różnych wartości, dla których zostały ustalone ich typy podstawowe i pola w rekordzie. Przez zdefiniowanie typu rekordowego można więc zdefiniować nowy typ zmiennej.

Deklaracja typu rekordowego składa się ze słowa "TYPE", nazwy typu i definicji pól rekordu. Każdą definicja pola określa jego typ podstawowy i nazwę. Definicja zmiennej rekordowej jest zbliżona do definicji zwykłej zmiennej i składa się z nazwy typu rekordowego i nazwy zmiennej oraz ewentualnie adresu tej zmiennej.

Dostąp do poszczególnych pól zmiennej rekordowej uzyskuje się przez identyfikator złożony z identyfikatora zmiennej, kropki (.) i identyfikatora pola. Na przykład:

```
 TYPE date=[BYTE day,month CARD year]
date term
term.year = 1989
term.month = 8
term.day = 1
```

Wyrażenia

Wyrażenie jest konstrukcją Action! oznaczającą wartość pewnego typu. Określenie tej wartości odbywa się przez wykonanie operacji wskazanych przez znajdujące się w wyrażeniu operatory. Wyrażenia są zbudowane ze stałych, zmiennych, operatorów, nazw funkcji i nawiasów okrągłych.

Przy konstruowaniu wyrażeń arytmetycznych w Action! można dowolnie mieszać typy BYTE, GARD i INT. Uzyskany wynik będzie w takim przypadku typu nadrzędnego, przy czym starszeństwo typów jest następujące: BYTE, INT, CARD. Jednakże operator jednoargumentowy "-" daje zawsze wynik typu INT, zaś operator "@" - typu CARD.

Proste wyrażenia logiczne mogą zawierać tylko jeden operator logiczny. Jeśli konieczne jest zbadanie złożonego warunku, to wyrażenie logiczne można rozbudować łącząc kilka wyrażeń prostych. Do połączenia można użyć wyłącznie operatorów AND i OR - zastosowanie w tym celu innego operatora spowoduje błąd podczas kompilacji programu. Wyrażenia logiczne mogą być używane wyłącznie w instrukcjach warunkowych i dają zawsze wynik "fałsz" lub "prawda".

W przypadku użycia jako wyrażenia warunkowego (w instrukcji warunkowej) wyrażenia, które daje wynik liczbowy, konieczna jest logiczna interpretacja uzyskanego wyniku. Wartość takiego wyrażenia równa zero jest zawsze traktowana jako "fałsz", a każda wartość różna od zera jest przyjmowana jako "prawda".

Operatory.

W Action! występują operatory arytmetyczne, bitowe i logiczne. Znaczna ich część ma odpowiedniki w innych językach, lecz są też takie, które różnią się zastosowanym symbolem lub realizowaną operacją. Wszystkie operatory dostępne w Action! są zestawione w poniższej tabeli. Gwiazdka <♦) przy nazwie operacji oznacza, że działanie operatora jest opisane w słowniku języka.

Operatory arytmetyczne

operator	operacja
—	znak
+	dodawanie
-	odejmowanie
*	mnożenie
/	dzielenie całkowite
MOD	reszta z dzielenia *

Operatory bitowe

operator	operacja
& lub AND	koniunkcja bitowa *
% lub OR	alternatywa bitowa *
! lub XOR	bitowa różnica symetryczna *
LSH	przesunięcie w lewo *
RSH	przesunięcie, w prawa *
@	adres zmiennej *

Operatory logiczne

operator	operacja
=	równość
() lub #	nierówność
<	mniejszość
>	większość
<=	nie większość
>=	nie mniejszość
AND	iloczyn logiczny *
OR	suma logiczna *

Priorytet operatorów, czyli kolejność wykonywania określonych przez nie operacji, jest podany w poniższej tabeli. Operatory znajdujące się wyżej są wykonywane przed operatorami umieszczonymi niżej. Operatory o jednakowym priorytecie znajdują się w jednym wierszu, a w wyrażeniu są realizowane od lewej do prawej.

( ) -
(znak) @
* / MOD LSH RSH
+ —
& AND
%
OR !
XOR

W instrukcjach przypisania wartości, w których po obu stronach występuje ta sama zmienna, możliwe jest zastosowanie dodatkowego operatora. Zamiast instrukcji w postaci:

<zmienna>=<zm ienna><operator><wyrazenie>

można użyć formy:

<zmienna>=<operator><wyrażenie>

Takie instrukcje przypisania są często wykorzystywane w programach i użycie skróconej formy zmniejsza liczbę błędów popełnianych podczas pisania programu, a także daje po skompilowaniu prostszy i krótszy kod wynikowy. Oto kilka przykładów zastosowania skróconej formy tej instrukcji:

```
   x=x+1          =>        x==+1
   x=x-y          =>        x==-y
   x=x&$0F        =>       x==&$0F
x=x LSH (2+i)     =>     x==LSH (2+i)
```

3. 5. Słownik

Ten rozdział zawiera kompletny słownik słów kluczowych jeżyka Action! w kolejności alfabetycznej. Podana jest tu składnia każdego słowa, sposoby jego wykorzystania oraz liczne przykłady. Ponadto na początku zamieszczony jest spis tych słów.

W słowniku przyjęta została następująca kolejność opisu: nazwa, typ, składnia, przykłady i działanie. Typ określa rodzaj słowa kluczowego: instrukcja, definicja typu lub dyrektywa. Składnia opisuje przy użyciu symboli podanych we wprowadzeniu dozwolone sposoby zapisu słowa kluczowego. Pozostałe punkty nie wymagają objaśnienia.

SŁOWA KLUCZOWE ACTION!

AND	IF	STEP
ARRAY	INCLUDE	THEN
BYTE	INT	TO
CARD	LSH	TYPE
CHAR	MOD	UNTIL
DEFINE	MODULE	WHILE
DO	OD	XOR
ELSE	OR	!
ELSE IF	POINTER	%
EXIT	PROC	&
FI	RETURN	@
FOR	RSH	;
FUNC	SET	

AND

TYP: operator logiczny i bitowy
FORMAT: <wyraż_log> AND <wyraż_log>
lub <wyraż_arytm> AND <wyraż_arytm>
PRZYKŁADY:

```
x = a AND b
PrintCE(a+2 AND x/2)
IF (a>0) AND (x<=0) THEN Print("prawda") FI
```

DZIAŁANIE: Wykonuje operację iloczynu logicznego dwóch argumentów logicznych lub operacją koniunkcji poszczególnych bitów dwóch argumentów liczbowych. Operator AND w wersji logicznej może być użyty tylko w złożonych wyrażeniach w instrukcjach warunkowych (IF, WHILE i UNTIL). W wersji bitowej operator AND jest równoważny operatorowi &. Gdy oba argumenty operacji logicznej są prawdziwe, to wynikiem jest także "prawda", a w przeciwnym przypadku "fałsz". Gdy oba bity argumentów operacji bitowej mają wartość 1, to wynikiem jest także 1, a w przeciwnym przypadku zero.

ARRAY

TYP: definicja typu
FORMAT: <typ> ARRAY <nazwa>[<<wyraż>)]
[= [<wyraż>] | <wyraż>][,<nazwa>...]
PRZYKŁADY:

```
BYTE ARRAY text(20),string
BYTE ARRAY color(5)=$2C4
CARD ARRAY num=[10 100 1000]
BYTE ARRAY firm=["SOETO"]
```

DZIAŁANIE: Definiuje podaną nazwę jako nazwę tablicy wskazanego typu. Nazwa może być dowolnym, poprawnym identyfikatorem, a typ tablicy musi być typem podstawowym. Nie trzeba określać rozmiaru definiowanej tablicy, można natomiast nadać wartość jej elementom lub ustalić położenie tablicy w pamięci. Wartość umieszczona,w nawiasie okrągłym jest rozmiarem tablicy, bez nawiasu — adresem tablicy, zaś w nawiasie kwadratowym podaje się początkowe wartości elementów. Elementy tablicy są zawsze numerowane od zera, a rozmiar tablicy określa ich liczbę.

BYTE

TYP: definicja typu
FORMAT: BYTE <nazwa>[=[<wyraż>]|<wyraż>][,<nazwa>...]
PRZYKŁADY:

```
BYTE znak BYTE x1,x2,x3
BYTE alfa=[$10],beta=1536
BYTE eol=[155],consol=$D01F
```

DZIAŁANIE: Definiuje podane nazwy jako nazwy jednobajtowych zmiennych liczbowych (z zakresu od 0 do 255). Nazwa może być dowolnym, poprawnym identyfikatorem. Definiowanej zmiennej można jednocześnie nadać wartość lub ustalić jej położenie w pamięci.

Umieszczona po nazwie wartość w nawiasie kwadratowym jest początkową wartością zmiennej, natomiast wartość bez nawiasu określa adres zmiennej. Typ BYTE jest równoważny z typem CHAR i można je stosować zamiennie.

Ponadto słowo BYTE służy do definiowania typu tablicy, wskaźnika i rekordu (patrz opis słów ARRAY, POINTER i TYPE) oraz do deklararowania typu parametrów procedur i funkcji (patrz opis PROC i FUNC) .

CARD

TYP: definicja typu
FORMAT: CARD <nazwa>[=]<wyraż>]|<wyraż>][,<nazwa>...]
PRZYKŁADY:

```
CARD liczba CARD x1,x2,x3
CARD alfa=[$100],beta=1536
CARD size=[2000],dliv=$200
```

DZIAŁANIE: Definiuje podane nazwy jako nazwy dwubajtowych zmiennych całkowitych bez znaku (z zakresu od 0 do 65S35). Nazwa może być dowolnym, poprawnym identyfikatorem. Definiowanej zmiennej można jednocześnie nadać wartość lub ustalić jej położenie w pamięci. Umieszczona po nazwie wartość w nawiasie kwadratowym jest początkową wartością zmiennej, natomiast wartość bez nawiasu określa adres zmiennej.

Ponadto słowo CARD służy do definiowania typu tablicy, wskaźnika i rekordu (patrz opis słów ARRAY, POINTER i TYPE) oraz do deklararowania typu parametrów procedur i funkcji (patrz opis PROC i FUNC).

CHAR

TYP: definicja typu
FORMAT: CHAR <nazwa>[=]<wyraż>]|<wyraż>][,<nazwa>...]
PRZYKŁADY:

```
CHAR znak CHAR x1,x2,x3
CHAR alfa=[$10],beta=1536
CHAR eol=[155],consol=*D01F
```

DZIAŁANIE: Definiuje podane nazwy jako nazwy jednobajtowych zmiennych znakowych (o wartościach z zakresu od 0 do 255). Nazwa może być dowolnym, poprawnym identyfikatorem. Definiowanej zmiennej można jednocześnie nadać wartość lub ustalić jej położenie w pamięci. Umieszczona po nazwie wartość w nawiasie kwadratowym jest początkową wartością zmiennej, natomiast wartość bez nawiasu określa adres zmiennej. Typ CHAR jest równoważny z typem BYTE i można je stosować zamiennie.

Ponadto słowo CHAR służy do definiowania typu tablicy, wskaźnika i rekordu (patrz opis słów ARRAY, POINTER i TYPE) oraz do deklararowania typu parametrów procedur i funkcji (patrz opis PROC i FUNC).

DEFINE

TYP: dyrektywa kompilatora
FORMAT: DEFINE <identyfikator>=<stała_tekst>[,...]
PRZYKŁADY:

```
DEFINE eol = "155"
DEFINE ramtop = "$6A"
DEFINE begin = "DO", end = "OD"
DEFINE list_on = "SET $49A=1"
```

DZIAŁANIE: Powoduje zastąpienie podczas kompilacji programu wskazanego identyfikatora przez określoną dla niego wartość stałą. Identyfikator jest zastępowany we wszystkich miejscach, w których występuje samodzielnie (nie jako część innego identyfikatora). Nie są zamieniane jedynie identyfikatory znajdujące się w stałych tekstowych. Zastosowanie tej dyrektywy znacznie ułatwia późniejsze modyfikowanie programów dzięki wyeliminowaniu konieczności przeszukiwania całego programu w celu zmiany wprowadzonej wartości.

DO

TYP: instrukcja
FORMAT: DO <instrukcja>... OD
PRZYKŁADY:

```
CARD num=[0]
DO
  PrintCE(num)
  num==+1
OD

CARD num=[0]
DO
  PrintCE(num)
  IF num>100 THEN EXIT FI
  num==+1
OD
```

DZIAŁANIE: Instrukcja DO rozpoczyna pętlę DO/OD. Wszystkie instrukcje zawarte pomiędzy słowami DO i OD są wykonywane, dopóki nie nastąpi przerwanie pętli. Pętla DO/OD bez instrukcji sterujących może być przerwana tylko przez instrukcję EXIT (zwykle umieszczaną w instrukcji IF), jest to więc pętla bez końca. Do modyfikowania pętli DO/OD służą instrukcje FOR, WHILE i UNTIL.

ELSE

TYP: instrukcja
FORMAT: patrz opis instrukcji IF
DZIAŁANIE: Słowo ELSE stanowi integralną część instrukcji IF. Następują po nim instrukcje, wykonywane gdy warunek po IF jest fałszywy. Część instrukcji IF rozpoczynająca się od słowa ELSE może zostać pominięta (patrz też opis IF).

ELSEIF

TYP: instrukcja
FORMAT: patrz opis instrukcji IF
DZIAŁANIE: Słowo ELSEIF stanowi integralną część instrukcji IF. Następuje po nim kolejny warunek sprawdzany przez tą instrukcję, gdy poprzednie warunki są fałszywe. Odpowiada do sekwencji ELSE IF... . Część instrukcji IF rozpoczynająca się od słowa ELSEIF może zostać pominięta (patrz też opis IF).

EXIT

TYP: instrukcja
FORMAT: EXIT
PRZYKŁAD:

```
CARD num=[0]
DO
  PrintCE(num)
  IF num>100 THEN EXIT FI
  num==+1
OD
```

DZIAŁANIE: Powoduje przerwanie wykonywania pętli DO/OD i jej opuszczenie. Następną instrukcją wykonywaną po EXIT jest instrukcja znajdująca się po słowie OD.

FI

TYP: instrukcja
FORMAT: patrz opis instrukcji IF
DZIAŁANIE: Słowo FI oznacza koniec bloku instrukcji warunkowej IF. Odnosi się ono do ostatniej jeszcze niezamkniętej takiej instrukcji. Patrz też opis instrukcji IF.

FOR

TYP: instrukcja
FORMAT: FOR <zmienna>:=<wart_pocz> TO <wart_końc>
[STEP <krok>] <pętla DO/OD>
PRZYKŁADY:

```
FOR i = 1 TO 10 DO silnia(i) OD
FOR i = 10 TO 1 STEP -1 DO silnia(i) OD
FOR 1 = 2*x TO 8*x DO oblicz(1) OD
FOR k=1 TO 20 DO PrintBE(k) k==+1 OD
```

DZIAŁANIE: Tworzy z pętli DO/OD bez końca pętlę, w której instrukcje znajdujące się między słowami DO i OD wykonywane są określoną ilość razy. Liczba przejść pętli oraz wartości licznika są ustalane przez parametry instrukcji FOR. Kolejno oznaczają one:
<zmienna> - nazwa zmiennej typu podstawowego wykorzystywanej jako licznik pętli (tzw. zmienna sterująca);
<wart_pocz> - początkowa wartość licznika pętli, przypisywana zmiennej sterującej przed rozpoczęciem pętli; <wart_końc> - wartość graniczna licznika - pętla jest wykonywana

tak długo, dopóki wartość zmiennej sterującej nie przekracza wartości granicznej;
<krok> — wartość, o którą zwiększana jest zmienna sterująca po każdym przejściu pętli, w przypadku pominięcia słowa STEP przyjmowana jest wartość kroku równa 1;
<pętla DO/OD> - zwykła pętla DO/OD bez końca, która teraz jest wykonywana ograniczoną liczbę razy.

FUNC

TYP: definicja funkcji
FORMAT: <typ> FUNC <nazwa>([<lista_param>])
lub <typ> FUNC = <adres>
PRZYKŁADY:

```
INT FUNC czas()
CHAR FUNC dzień(BYTE d)
CARD FUNC oblicz(INT a, CARD b,c,d)
BYTE FUNC getch=$F180
```

DZIAŁANIE: Rozpoczyna deklarację funkcji o nazwie podanej jako pierwszy parametr. Po nazwie może być umieszczona lista parametrów, która wylicza wszystkie parametry danej funkcji oraz ich typ. Dozwolone jest użycie do ośmiu parametrów. Typ wyniku zwracanego przez funkcję jest podany w postaci identyfikatora typu umieszczonego przed słowem FUNC.

Opisane wyżej elementy stanowią nagłówek funkcji. Dalsza część deklaracji może się składać z definicji zmiennych lokalnych oraz instrukcji języka. Definicja funkcji powinna zawierać co najmniej jedną instrukcję RETURN, która kończy funkcję.

Wywołanie funkcji następuje przez podanie jej nazwy oraz zamkniętej w nawiasy okrągłe listy aktualnych parametrów przekazywanych do funkcji. Nie wolno użyć wywołania funkcji wewnątrz wywołania procedury lub innej funkcji. Nie wolno także przekazać do funkcji większej liczby parametrów, niż określona w definicji, dozwolone jest natomiast użycie mniejszej liczby parametrów. Dopuszczalne jest również takie wywołanie funkcji, które nie wykorzystuje zwracanej przez nią wartości.

Możliwe jest także zdefiniowanie funkcji wyłącznie przez jej adres (patrz drugi format), co pozwala na wykorzystanie procedur systemu operacyjnego. Tak zdefiniowana funkcja nie zawiera innych elementów (nagłówek stanowi całość definicji). Przy wywoływaniu takiej funkcji kolejne parametry są umieszczane w akumulatorze, rejestrze X, rejestrze Y oraz od adresu $A3 do $AF, a następnie wykonywany jest skok do procedury maszynowej, której adres początkowy jest podany w definicji funkcji. Jeśli po realizacji tej procedury ma nastąpić powrót do Action!, to musi się ona kończyć rozkazem RTS.

IF

TYP: instrukcja
FORMAT: IF <wyraż_log> THEN <instrukcje>
[ELSEIF <wyraż_log> THEN <instrukcje>] [...]
[ELSE <instrukcje>] FI

PRZYKŁADY:

```
IF x THEN oblicz(x) FI
IF y>0 THEN x=y ELSE x=-y FI
IF a=7 THEN PrintE("niedziela")
ELSEIF a=6 THEN PrintE("sobota")
ELSE PrintE("dzien roboczy") FI
```

DZIAŁANIE: Pozwala na wybranie sposobu postępowania w zależności od wartości wyrażenia warunkowego. Jeżeli warunek ma wartość prawdziwą (różną od zera), to wykonywane są instrukcje umieszczone między słowami THEN i ELSE lub ELSEIF. W przeciwnym przypadku wykonywane są instrukcje znajdujące się między ELSE i FI. Umieszczenie słowa ELSEIF powoduje sprawdzenie kolejnego warunku oraz ewentualne wykonanie instrukcji znajdujących się między THEN i następnym ELSEIF lub ELSE. Po wykonaniu wybranej grupy instrukcji oraz gdy takiej grupy nie ma, to realizacja programu jest kontynuowana od instrukcji następującej po FI.

INCLUDE

TYP: dyrektywa kompilatora
FORMAT: INCLUDE <"specyfikacja_pliku">
PRZYKŁADY:

```
INCLUDE "DsPMG.ACT"
INCLUDE "SOUND.ACT"
```

DZIAŁANIE: Dołącza do kompilowanego programu procedury zawarte w pliku dyskowym wskazanym w dyrektywie. Umożliwia to stworzenie biblioteki procedur i uproszczenie pisanych później programów. Ponadto dzięki tej dyrektywie możliwe jest kompilowanie programów, które w całości nie mogą być skompilowane ze względu na znaczną objętość. Dyrektywy INCLUDE mogą być zagnieżdżane, to znaczy w pliku dołączanym przez INCLUDE można również umieścić takie dyrektywy. Action! zezwala na 6 poziomów zagnieżdżenia, lecz dodatkowe ograniczenia wprowadzają urządzenia zewnętrzne. Przy pracy ze stacją dysków dopuszczalne są 3,poziomy zagnieżdżenia, a dla magnetofonu tylko jeden. Jeżeli w buforze edytora Action! nie ma żadnego programu, to dozwolona liczba zagnieżdżeń zmniejsza się o jeden.

INT

TYP: definicja typu
FORMAT: INT <nazwa>[=[<wyraż>]|<wyraż>][,<nazwa>...]
PRZYKŁADY:

```
INT liczba
INT x1,x2,x3
INT alfa=[$100],beta=1536
INT value=[-2000],number=$600
```

DZIAŁANIE: Definiuje podane nazwy jako nazwy dwubajtowych zmiennych całkowitych ze znakiem (z zakresu od —32767 do 32768). Nazwa może być dowolnym, poprawnym identyfikatorem. Definiowanej zmiennej można jednocześnie nadać wartość lub ustalić jej położenie w pamięci. Umieszczona po nazwie wartość w nawiasie kwadratowym jest początkową wartością zmiennej, natomiast wartość bez nawiasu określa adres zmiennej.

Ponadto słowo INT służy do definiowania typu tablicy, wskaźnika i rekordu (patrz opis słów ARRAY, POINTER i TYPE) oraz do deklararowania typu parametrów procedur i funkcji (patrz opis PROC i FUNC.

LSH

TYP: operator bitowy
FORMAT: <wyrażenie> LSH <wyrażenie>
PRZYKŁADY:

```
x = y LSH 2
num = 3*a LSH b+2
PrintC($200 LSH 3)
```

DZIAŁANIE: Powoduje przesunięcie w lewo bitów lewego argumentu o liczbą miejsc określoną przez prawy argument. Opróżnione miejsca są wypełniane zerami. Wykonanie przesunięcia o jeden bit odpowiada operacji mnożenia przez 2, lecz jest wykonywane znacznie szybciej.

MOD

TYP: operator arytmetyczny
FORMAT: <wyraż_liczb> MOD <wyraż_liczb>
PRZYKŁADY:

```
x = 20 MOD 3
PrintCE(x MOD y)
IF a MOD 2 = 0 THEN Print("liczba parzysta") FI
```

DZIAŁANIE: Operator MOD realizuje dzielenie modulo, czyli daje w rezultacie resztę z dzielenia pierwszego argumentu przez drugi. Argumenty operacji mogą być dowolnego typu.

MODULE

TYP: dyrektywa kompilatora
FORMAT: MODULE
DZIAŁANIE: Wskazuje kompilatorowi, że następujące po niej definicje zmiennych określają zmienne globalne. Definicje te muszą się znajdować pomiędzy dyrektywą MODULE i pierwszą po niej definicją procedury lub funkcji. Użycie tej dyrektywy nie jest konieczne, gdyż kompilator automatycznie zakłada jej istnienie na początku programu. Najczęściej dyrektywa MODULE jest stosowana przy pisaniu programów w kilku częściach.

OD

TYP: instrukcja
FORMAT: patrz opis instrukcji DO
DZIAŁANIE: Słowo OD oznacza koniec bloku instrukcji zawartych w pętli DO/OD. Odnosi się ono do ostatniej jeszcze niezamkniętej takiej pętli.

OR

TYP: operator logiczny i bitowy
FORMAT: <wyraż_log> OR <wyraż_log>
  lub <wyraż_arytm> OR <wyraż_arytm>
PRZYKŁADY:

```
x = a OR b
PrintCE(a+2 OR x/2)
IF (a>0) OR (x<=0) THEN Print("prawda") FI
```

DZIAŁANIE: Wykonuje operację sumy logicznej dwóch argumentów logicznych lub operację alternatywy poszczególnych bitów dwóch argumentów liczbowych. Operator OR w wersji logicznej może być użyty tylko w złożonych wyrażeniach w instrukcjach warunkowych (IF, WHILE i UNTIL). W wersji bitowej operator OR jest równoważny operatorowi %. Gdy oba argumenty operacji logicznej są fałszywe, to wynikiem jest także "fałsz", a w przeciwnym przypadku "prawda". Gdy oba bity argumentów operacji bitowej mają wartość 0, to wynikiem jest także 0, a w przeciwnym przypadku jeden.

POINTER

TYP: definicja typu
FORMAT: <typ> POINTER <nazwa>[=<wyraż>][,<nazwa>...]
PRZYKŁADY:

```
BYTE POINTER ptr
CARD POINTER adr_1,adr_2,adr_3
INT POINTER wsk=$8000
```

DZIAŁANIE: Definiuje podaną nazwę jako nazwę wskaźnika, który wskazuje zmienne typu podstawowego wymienionego w definicji. Nazwa może być dowolnym, poprawnym identyfikatorem. W definicji można wskaźnikowi nadać wartość (zawsze typu CARD).

PROC

TYP: definicja procedury
FORMAT: PROC <nazwa>([<lista_param>])
  lub PROC = <adres>
PRZYKŁADY:

```
PROC czas() PROC dzien(BYTE d)
PROC oblicz(INT a, CARD b,c,d)
PROC cio=$E456
```

DZIAŁANIE: Rozpoczyna deklarację procedury o nazwie podanej jako pierwszy parametr. Po nazwie może być umieszczona lista parametrów, która wylicza wszystkie parametry danej procedury oraz ich typ. Dozwolone jest użycie do ośmiu parametrów. Procedura nie zwraca żadnego wyniku.

Opisane wyżej elementy stanowią nagłówek procedury. Dalsza część deklaracji może się składać z definicji zmiennych lokalnych oraz instrukcji języka. Definicja procedury powinna zawierać co najmniej jedną instrukcję RETURN, która kończy procedurę.

Wywołanie procedury następuje przez podanie jej nazwy oraz

zamkniętej w nawiasy okrągłe listy aktualnych parametrów przekazywanych do procedury. Nie wolno użyć wywołania procedury wewnątrz wywołania funkcji lub innej procedury. Nie wolno także przekazać do procedury większej liczby parametrów, niż określona w definicji, dozwolone jest natomiast użycie mniejszej liczby parametrów.

Możliwe jest także zdefiniowanie procedury wyłącznie przez jej adres (patrz drugi format), co pozwala na wykorzystanie procedur systemu operacyjnego. Tak zdefiniowana procedura nie zawiera innych elementów (nagłówek stanowi całość definicji). Przy wywoływaniu takiej procedury kolejne parametry są umieszczane w akumulatorze, rejestrze X, rejestrze Y oraz od adresu $A3 do $AF, a następnie wykonywany jest skok do procedury maszynowej, której adres początkowy jest podany w definicji procedury. Jeśli po realizacji tej procedury ma nastąpić powrót do Action!, to musi się ona kończyć rozkazem RTS.

RETURN

TYP: instrukcja
FORMAT: RETURN [(<wyrażenie>)]
PRZYKŁADY:

```
RETURN
RETURN (i)
RETURN (x*y+2)
```

DZIAŁANIE: Przerywa wykonywanie procedury lub funkcji, w której się znajduje, i powoduje powrót do miejsca wywołania. Wartość wyrażenia znajdującego się w instrukcji RETURN jest przekazywana do miejsca wywołania. Jeśli w instrukcji RETURN nie ma żadnego wyrażenia, to powrót do miejsca wywołania zwraca wartość nieokreśloną. W każdej procedurze lub funkcji można umieścić dowolną liczbę instrukcji RETURN, jednak już pierwsza z nich napotkana podczas realizacji programu powoduje opuszczenie procedury lub funkcji.

RSH

TYP: operator bitowy
FORMAT: <wyrażenie> RSH <wyrażenie>
PRZYKŁADY:

```
x = y RSH 2
nun = 3*a RSH b+2
PrintC($200 RSH 3)
```

DZIAŁANIE: Powoduje przesunięcie w prawo bitów lewego argumentu o liczbę miejsc określoną przez prawy argument. Opróżnione miejsca są wypełniane zerami. Wykonanie przesunięcia o jeden bit odpowiada operacji dzielenia przez 2, lecz jest wykonywane znacznie szybciej.

SET

TYP: dyrektywa kompilatora
FORMAT: SET <adres> = <wartość>

PRZYKŁADY:

```
SET $600 = 64
SET max =16
SET 10OOO = $FFFF
SET $CFOO = value
```

DZIAŁANIE: Służy do zmiany zawartości komórek pamięci RAM podczas kompilacji. Pozwala to na zmianę wariantów pracy edytora i kompilatora oraz modyfikację zmiennych systemowych przez program użytkownika. Wartości użyte w dyrektywie SET mogą być dowolnymi stałymi kompilatora.

Najważniejszym dla programisty zastosowaniem dyrektywy SET jest ustalanie początkowego adresu kompilacji. Normalnie skompilowany program jest umieszczany w pamięci poza końcem obszaru zajętego przez program źródłowy znajdujący się w edytorze. Programista może zmienić adres początkowy kompilacji przy pomocy dyrektyw:

```
SET $0E = <adres> SET
$491 = <adres>
```

gdzie <adres> jest początkowym adresem kompilacji. W celu zabezpieczenia prawidłowej pracy systemu Action! adres ten nie powinien być mniejszy od wartości MEMLO+$200. Aktualną wartość MEMLO odczytuje się z komórek $2E7 i $2E8.

UWAGA: Dyrektywa SET działa jedynie podczas kompilacji, w odróżnieniu od procedur Poke i PokeC, które działają podczas pracy programu.

STEP

TYP: instrukcja
FORMAT: patrz opis.instrukcji FOR
DZIAŁANIE: Słowo STEP stanowi integralną część instrukcji FOR. Następujące po nim wyrażenie określa krok pętli czyli wartość, o którą należy zmienić zmienną sterującą. Patrz też opis instrukcji FOR.

THEN

TYP: instrukcja
FORMAT: patrz opis instrukcji IF
DZIAŁANIE: Słowo THEN stanowi integralną część instrukcji IF. Następują po nim instrukcje, wykonywane gdy warunek przed THEN jest prawdziwy. Patrz też opis instrukcji IF.

TO

TYF: instrukcja
FORMAT: patrz opis instrukcji FOR
DZIAŁANIE: Słowo TO stanowi integralną część instrukcji FOR. Następuje po nim wyrażenie określające wartość graniczną zmiennej sterującej pętli. Przekroczenie tej wartości powoduje przerwanie działania pętli. Patrz też opis instrukcji FOR.

TYPE

TYP: deklaracja typu
FORMAT: TYPE <nazwa>=[<deklaracje_pól>]
PRZYKŁADY:

```
TYPE rec=[BYTE b1,b2 INT i CARD c1,c2 BYTE b3]
TYPE info=[BYTE level CARD idn,value]
TYPE date=[BYTE day,month CARD year]
```

DZIAŁANIE: Definiuje podaną nazwę jako nazwą typu rekordowego. Nazwa może być dowolnym, poprawnym identyfikatorem. Opis formatu typu wskazuje kolejność i typ pól zawartych w rekordzie. Zmienna typu rekordowego musi być zdefiniowana po definicji typu, a w jej definicji jako nazwy typu należy użyć nazwy typu rekordowego. Nie wolno jednak w definicji nadawać wartości takim zmiennym. Na przykład (definicje typów jak wyżej):

```
rec arec,brec=$8000,crec
info entry_1,entry_2
date start,end
```

Dostęp do poszczególnych pól zmiennej rekordowej (w celu przypisania lub odczytania ich wartości) uzyskuje się przez użycie identyfikatora złożonego z nazw zmiennej i pola oddzielanych od siebie kropką. Na przykład:

```
arec.b1 = (x LSH 2) + $10
PrintC(entry_1.idn)
FOR i=start.day TO end.day DO ...
```

UNTIL

TYP: instrukcja
FORMAT: DO <instrukcje> UNTIL <wyraż_log> OD
PRZYKŁADY:

```
DO oblicz(a) UNTIL stan OD
DO x=2*a y=0.5*a UNTIL x+y>100 OD
DO x=x+1 UNTIL x>100 OD
```

DZIAŁANIE: Tworzy z pętli DO/OD bez końca pętlą warunkową. Jest ona wykonywana tak długo, dopóki warunek pętli znajdujący sią po słowie UNTIL jest fałszywy (równy zero). Warunek prawdziwy (niezerowy) powoduje przerwanie pętli i wykonanie instrukcji następującej po słowie OD. Warunek ten jest sprawdzany na końcu pętli, wiąc zawarte w niej instrukcje muszą być wykonane co najmniej jeden raz.

WHILE

TYP: instrukcja
FORMAT: WHILE <wyrażenie> <pętla DO/OD>
PRZYKŁADY:

```
WHILE a<100 DO oblicz(a) OD
WHILE x<100 DO x+=1 OD
WHILE lim==-1>0 AND c#155 DO s(i)=c i==+1 OD
```

DZIAŁANIE: Tworzy z pętli DO/OD bez końca pętlą warunkową. Jest ona wykonywana tak długo, dopóki warunek pętli znajdujący się po słowie WHILE jest prawdziwy (różny od zera). Warunek fałszywy (równy zero) powoduje przerwanie pętli i wykonanie instrukcji następującej po słowie OD. Warunek ten jest sprawdzany na początku pętli, więc zawarte w niej instrukcje mogą wcale nie być wykonane.

XOR

TYP: operator bitowy
FORMAT: <wyraż_arytm> XOR <wyraż_arytm>
PRZYKŁADY:

```
x = a XOR b
PrintCE(a+2 XOR x/2)
```

DZIAŁANIE: Wykonuje operację różnicy symetrycznej na poszczególnych bitach dwóch argumentów liczbowych. Operator ten jest równoważny operatorowi !. Gdy odpowiadające sobie bity obu argumentów mają równe wartości, to wynikiem jest 0, a w przeciwnym przypadku jeden.

!

TYP: operator bitowy
FORMAT: <wyraż_arytm> ! <wyraż_arytm>
PRZYKŁADY:

```
x = a ! b
PrintCE(a+2 ! x/2)
```

DZIAŁANIE: Wykonuje operację różnicy symetrycznej na poszczególnych bitach dwóch argumentów liczbowych. Operator ten jest równoważny operatorowi XOR. Gdy odpowiadające sobie bity obu argumentów mają równe wartości, to wynikiem jest 0, a w przeciwnym przypadku jeden.

%

TYP: operator bitowy
FORMAT: <wyraż_arytm> % <wyraż_arytm>
PRZYKŁADY:

```
x = a % b
PrintCE(a+2 % x/2)
IF c LSH 2 % $20 THEN Print("prawda") FI
```

DZIAŁANIE: Wykonuje operację alternatywy poszczególnych bitów dwóch argumentów liczbowych. Operator ten jest równoważny operatorowi OR, lecz realizuje tylko operacje bitowe. Gdy oba odpowiadające sobie bity argumentów mają wartość O, to wynikiem jest również O, a w przeciwnym przypadku jeden.

&

TYP: operator bitowy
FORMAT: <wyraż_arytm> & <wyraż_arytm>

PRZYKŁADY:

```
x = a & b
PrintCE(a+2 & x/2)
IF c LSH 2 & $20 THEN Print("prawda") FI
```

DZIAŁANIE: Wykonuje operację koniunkcji poszczególnych bitów dwóch argumentów liczbowych. Operator ten jest równoważny operatorowi AND, lecz realizuje tylko operacje bitowe. Gdy oba odpowiadające sobie bity argumentów mają wartość 1, to wynikiem jest również 1, a w przeciwnym przypadku zero.

@

TYP: operator arytmetyczny
FORMAT: @ <zmienna>
PRZYKŁADY:

```
ptr = @value
PrintCE(@color)
x = @n x^ = 222 PrintB(n)
```

DZIAŁANIE: Zwraca adres zmiennej, która jest argumentem. Nieozwolone jest użycie stałych liczbowych, jako argumentów operacji. Operator @ jest najczęściej stosowany w działaniach na wskaźnikach.

;

TYP: instrukcja
FORMAT: ;[<dowolny ciąg znaków>]
PRZYKŁADY:

```
;*** TO JEST KOMENTARZ *** oblicz (x)
;obliczenie silni ;** dozwolone wszystkie
znaki ; IF Poke
```

DZIAŁANIE: Znak ten rozpoczyna komentarz, który jest podczas kompilacji programu całkowicie ignorowany. Po jego napotkaniu dalsza kompilacja programu jest wykonywana od następnego wiersza programu. W komentarzu dozwolone są wszystkie znaki dostępne w Atari.

3. 6. Biblioteka

Język Action! - podobnie jak C - posiada bibliotekę, która zawiera wiele procedur umożliwiających pełniejsze wykorzystanie możliwości komputera. Procedury te zawarte są na cartridge'u i dla użytkownika stanowią po prostu nierozłączną część języka. Jednak ze względu na strukturę języka i jego składnię są one zawsze opisywane oddzielnie. Większość z procedur bibliotecznych ma swoje odpowiedniki w instrukcjach Atari Basic.

Ten rozdział zawiera kompletny słownik procedur bibliotecznych Action! opisanych w kolejności alfabetycznej. Podana jest tu nazwa każdej funkcji, jej definicja, sposoby wykorzystania oraz liczne przykłady. Ponadto na początku zamieszczony jest spis tych procedur.

W słowniku przyjęta została następująca kolejność opisu: nazwa, typ, definicja, przykłady i działanie. Typ określa rodzaj procedury. Definicja opisuje sposób wywołania i deklaracje parametrów wywołania. Pozostałe punkty nie wymagają objaśnienia.

UWAGA: Użytkownik może zdefiniować własne procedury o takich samych nazwach. Podczas kompilacji programu procedury biblioteczne zostaną w takim przypadku zastąpione przez procedury użytkownika.

PROCEDURY BIBLIOTECZNE ACTION!

Break	LIST	PrintCD	SCompare
Close	Locate	PrintCDE	SCopy
color	MoveBlock	PrintCE	SCopyS
device	Note	PrintD	SetBlock
DrawTo	Open	PrintDE	SetColor
EOF	Paddle	PrintE	SndRst
Error	Peek	PrintF	Bound
Fill	PeekC	PrintI	Stick
GetD	Plot	PrintID	StrB
Graphics	Point	PrintIDE	StrC
InputB	Poke	PrintIE	StrI
InputBD	PokeC	Ptrig	Strig
InputC	Position	Put	TRACE
InputCD	Print	PutD	ValB
InputI	PrintB	PutDE	ValC
InputID	PrintBD	PutE	ValI
InputMD	PrintBDE	Rand	XIO
InputS	PrintBE	SAssign	Zero
InputSD	PrintC		

Break

TYP: procedura operacyjna
DEFINICJA: PROC Break()
PRZYKŁADY:

```
Break()
IF x=0 THEN Break() FI
```

DZIAŁANIE: Zatrzymuje działanie programu i sygnalizuje błąd numer 128. Po zatrzymaniu programu możliwe jest sprawdzenie wartości zmiennych i funkcji. Do wznowienia realizacji programu służy polecenie monitora Proceed. Procedura Break() jest zwykle wykorzystywana podczas uruchamiania programów.

Close

TYP: procedura I/O
DEFINICJA: PROC Close(BYTE channel)
PRZYKŁADY:

```
Close(2)
Close(iocb)
```

DZIAŁANIE: Zamyka kanał IOCB o podanym numerze i kończy w ten sposób transmisję przeprowadzaną tym kanałem. Przed zamknięciem kanału aktualna zawartość jego bufora jest jeszcze przesyłana i nie zastaje utracona. Zamknięcie zamkniętego kanału nie powoduje błędu. Dozwolone są numery kanałów od 0 do 7, jednak nie należy zamykać kanału 7 (patrz opis procedury Open).

color

TYP: zmienna biblioteczna
DEFINICJA: BYTE color=$2FD color=0
PRZYKŁADY:

```
color = 3
color = x
color = '#
```

DZIAŁANIE: Wybiera kolor lub znak, który będzie umieszczany ,na ekranie przez procedury Plot, DrawTo i Fill. Wartość zmiennej przekraczająca dostępną liczbę kolorów jest przez nią dzielona. Reszta z tego dzielenia wybiera rejestr koloru. Użyte tu liczby są INNE niż w procedurze SetColor: zero oznacza zawsze kolor tła, a dalsze wartości — kolejne kolory.

device

TYP: zmienna biblioteczna
DEFINICJA: BYTE device=$B7 device=0
PRZYKŁADY:

```
device = 5
device = x
```

DZIAŁANIE: Określa numer standardowego kanału IOCB, który jest wykorzystywany przez operacje wejścia/wyjścia. Zmiana wartości tej zmiennej pozwala na zmianę urządzenia I/O bez zmiany użytych procedur.

DrawTo

TYP: procedura graficzna
DEFINICJA: PROC DrawTo(CARD col, BYTE row)
PRZYKŁADY:

```
DrawTo(10,10)
DrawTo(x,y)
DrawTo(i*2,j+10)
```

DZIAŁANIE: Rysuje na ekranie linię prostą od aktualnego położenia kursora do punktu o podanych współrzędnych. Punkt 0,0 znajduje się w lewym, górnym rogu ekranu. pierwsza liczba określa współrzędną poziomą, a druga pionową. Przekroczenie dopuszczalnej wartości współrzędnych powoduje błąd numer 141.

Wybór koloru (w trybach bitowych) lub znaku (w trybach znakowych), którym linia zostanie narysowana, jest wykonywany przy użyciu zmiennej color.

EOF

TYP: tablica biblioteczna
DEFINICJA: BYTE ARRAY EOF(8)
PRZYKŁADY:

```
x = EOF(1)
IF EOF(x) THEN Close(x) FI
```

DZIAŁANIE: Zawiera wartości wskazujące stan otwartych kanałów IOCB. Jeżeli w kanale został napotkany koniec pliku, to odpowiadający mu element tablicy ma wartość 1, a w przeciwnym przypadku zero. Jeśli kanał IOCB nie został otwarty, to odpowiedni element tablicy EOF ma wartość przypadkową.

Error

TYP: procedura operacyjna
DEFINICJA: PROC Error
DZIAŁANIE: Procedura ta jest wywoływana przez Action! w przypadku wystąpienia w programie błędu. Użytkownik nie może jej sam wywołać, lecz może ją zastąpić własną procedurą obsługi błędu, która spełni taką rolę, jak instrukcja TRAP w Atari Basic. W tym celu należy zdefiniować nową procedurę i jej adres przypisać procedurze Error. Pierwotny adres trzeba przechować i odtworzyć po zakończeniu programu, aby nie spowodować zawieszenia się komputera. Ilustruje to poniższy przykład:

```
PROC NewError(BYTE errcode)
 .... ;tu treść procedury
RETURN

PROC Program()
 CARD tmperr
 tmperr = Error
 Error = NewError
 .... ;tu treść programu
 Error = tmperr
RETURN
```

Fill

TYP: procedura graficzna
DEFINICJA: PROC Fill(CARD col, BYTE row)
PRZYKŁADY:

```
Fill(20,10)
Fill(hor,vert)
Fill(i*2,j+10)
```

DZIAŁANIE: Wypełnia tło obrazu w prawo od każdego narysowanego punktu. Wypełnianie danej linii kończy się po napotkaniu punktu o kolorze innym niż kolor tła. Znaczenie parametrów jest takie samo, jak w procedurze DrawTo. Numer koloru, który będzie użyty do wypełniania jest ustalany przez wartość zmiennej color. Poniższy przykład jest przerobioną wersją przykładu ilustrującego użycie XIO 18, a zamieszczonego w "Poradniku programisty Atari".

```
PROC example() BYTE
 y1,y2,c,consol=53279 CARD
 x1,x2

 Graphics(31)
 DO
   x1  =  Rand(120)+40
   y1  =  Rand(162)+30
   DO x2 = Rand(140)
     UNTIL x2<x1 AND x2>=x.1-60 OD
   DO y2 = Rand(180)
     UNTIL y2<Y1 AND y2>=y1-60 OD
   c==+1
   IF c>3 THEN c=1 FI
   color = c
   Plot(x1,y1)
   DrawTo(x1,y2)
   DrawTo(x2,y2)
   Fill(x2,y1)
   DO
    UNTIL consol=6 OD
 OD
RETURN
```

GetD

TYP: funkcja I/O
DEFINICJA: CHAR FUNC GetD(BYTE channel)
PRZYKŁADY:

```
x = GetD(1)
key = GetD(chn)
```

DZIAŁANIE: Zwraca wartość bajtu danych pobranego z kanału IOCB a numerze wskazanym przez parametr. Kanał transmisji musi być uprzednio otwarty. Dozwolone są numery kanałów od 0 do 7. Funkcja GetD musi pobrać bajt, więc gdy go nie ma, to zawsze czeka na jego dostarczenie. Przy odczycie z klawiatury zatrzymuje to pracę programu.

Graphics

TYP: procedura graficzna
DEFINICJA: PROC Graphics(BYTE mode)
PRZYKŁADY:

```
Graphics(1)
Graphics(17)
Graphics(1+32)
Graphics(x)
```

DZIAŁANIE: Wybiera tryb wyświetlania obrazu według podanego parametru. System operacyjny Atari umożliwia uzyskanie szesnastu różnych trybów graficznych: pięciu znakowych i jedenastu bitowych. Wybierane są one przez wartości parametru z zakresu od 0 do 15. Ponadto zwiększenie numeru trybu o 16 ustala tryb graficzny bez okna tekstowego, które ma zawsze tryb Graphics(0). Numer trybu zwiększony o 32 powoduje zachowanie niezmienionej zawartości obszaru pamięci obrazu. Oba te warianty mogą być zastosowane łącznie - numer trybu musi być wtedy zwiększony o 48. Wykaz dostępnych trybów graficznych oraz ich szczegółowy opis znajduje się w "Poradniku programisty Atari".

InputB
InputBD
InputC
InputCD
InputI
InputID

TYP: funkcje I/O
DEFINICJE:

```
BYTE FUNC InputB()
BYTE FUNC InputBD(BYTE channel)
CARD FUNC InputC()
CARD FUNC InputCD(BYTE channel)
INT FUNC InputI()
INT FUNC Input ID(BYTE channel)
```

PRZYKŁADY:

```
a = InputB()
x = InputC()
y = InputI()
k = InputBD(4)
ą = InputCD(chn)
b = InputID(iocb)
```

DZIAŁANIE: Zwraca wartość liczbową typu zgodnego z typem funkcji odczytaną z urządzenia zewnętrznego. Jeśli funkcja ma w nazwie literę D, to odczyt wykonywany jest przez kanał IOCB określony przez parametr funkcji (dozwolone są przy tym wartości od 0 do 7) . W przeciwnym przypadku wartość jest odczytywana z kanału standardowego, ustalonego przez zmienną device.

Przy odczycie danej z edytora użytkownik wpisuje informację poprzez klawiaturę, a komputer przyjmuje ją dopiero po naciśnięciu klawisza <RETURN>. Przy odczycie z innego urządzenia (wcześniej otwartego procedurą Open) system sam pobiera daną i zwraca ją jako wartość funkcji. Na ekranie nie pojawia się żaden ślad tej operacji.

InputS
InputSD
InputMD

TYP: procedura I/O
DEFINICJE: PROC Inputs(<string>)
PROC InputSD(BYTE channel, <string>> PROC InputMD(BYTE channel, <string>, BYTE max)
PRZYKŁADY:

```
InputMD(2,text,20)
InputMD(iocb,chr,len)
Inputs(name)
InputSD(5,text)
```

DZIAŁANIE: Odczytuje ciąg znaków z urządzenia zewnętrznego i przypisuje go tablicy o nazwie <string>. Tablica ta musi być zdefiniowana jako BYTE ARRAY. Procedury S przepisują cały ciąg, a procedura M tylko taką liczbę znaków ciągu, która nie przekracza ustalonej przez trzeci parametr długości maksymalnej. Jeśli procedura ma w nazwie literę D, to odczyt wykonywany jest przez kanał IOCB określony przez parametr procedury (dozwolone są przy tym wartości od 0 do 7). U przeciwnym przypadku ciąg jest odczytywany ' z kanału standardowego, ustalonego przez zmienną device.

Przy odczycie danej z edytora użytkownik wpisuje informację poprzez klawiaturę, a komputer przyjmuje ją dopiero po naciśnięciu klawisza <RETURN>. Przy odczycie z innego urządzenia (wcześniej otwartego procedurą Open) system sam pobiera daną i zwraca ją jako wartość funkcji. Na ekranie nie pojawia się żaden ślad tej operacji.

LIST

TYP: zmienna biblioteczna
DEFINICJA: BYTE LIST
PRZYKŁADY:

```
SET LIST=0
SET LIST=1
```

DZIAŁANIE: Steruje wariantem "LIST" kompilatora. Musi być użyta w dyrektywie SET i musi się znajdować na początku programu. Przypisanie zmiennej LIST wartości 1 włącza listowanie, a wartości zero - wyłącza.

Locate

TYP: funkcja graficzna
DEFINICJA: BYTE FUNC Locate(CARD col, BYTE row)
PRZYKŁADY:

```
a = Locate(5,10)
code = Locate(x*2,y)
```

DZIAŁANIE: Umieszcza kursor w punkcie ekranu o podanych współrzędnych i zwraca kod znajdującego się tam znaku lub koloru. W trybie znakowym jest to kod ASCII znaku, zaś w trybie bitowym - kod koloru. Po wykonaniu funkcji kursor przesuwany jest o jedno miejsce w prawo.

moveblock

TYP: procedura operacyjna
DEFINICJA: PROC MoveBlock(BYTE POINTER dest,source,
CARD size)
PRZYKŁADY:

```
MoveBlock(chrom,chb,1024)
MoveBlock(pm,pm+2,256)
MoveBlock(adr_1,adr_2,count)
```

DZIAŁANIE: Przemieszcza blok bajtów o długości określonej przez trzeci parametr z obszaru rozpoczynającego się od adresu wskazanego drugim parametrem do obszaru wskazanego pierwszym. Jeśli wybrane obszary pamięci się pokrywają i adres źródłowy jest mniejszy od docelowego, to część przepisywanego bloku ulegnie zniszczeniu. W takim przypadku należy użyć pomocniczego obszaru i procedurę MoveBlock wywołać dwukrotnie.

Note

TYP: procedura I/O
DEFINICJA: PROC Note(BYTE channel, CARD POINTER sector, BYTE POINTER offset)
PRZYKŁADY:

```
Note(1,sect,byte)
Note(iocb, x,y)
```

DZIAŁANIE: Odczytuje aktualne wartości wskaźnika głowicy stacji dysków w pliku otwartym przez t podany kanał IOCB i przypisuje je wskazanym zmiennym. Dozwolone są numery kanałów od 0 do 7. Pierwsza zmienna otrzymuje wartość określającą numer sektora, a druga numer bajtu w sektorze. Uzyskane wartości zależą od gęstości zapisu dyskietki. Numer sektora może mieć wartość od 1 do 1040, numer bajtu od 0 do 252.

Procedura Note działa tylko ze stacją dysków i wymaga uprzedniego otwarcia pliku przez procedurę Open.

Open

TYP: procedura I/O
DEFINICJA: PROC Open(BYTE channel,<file>,BYTE mode,aux2)
PRZYKŁADY:

```
Open(1,"K:",4,0)
Open(iocb,"P:",8,0)
Open(2,"D:NAZWA.DAT",x,0)
Open(k,fname,a,b)
Open(1,"D:*.*",6,0)
Open(5,"S:",12,7)
```

DZIAŁANIE: Otwiera kanał IOCB do komunikacji z urządzeniem zewnętrznym. Kolejne parametry procedury określają zadane parametry transmisji, przy czym <file> może być stałą tekstową lub nazwą tablicy zawierającej specyfikację pliku.

Procedura Open jest dokładnym odpowiednikiem instrukcji OPEN w Atari Basic. Znaczenie wszystkich parametrów Open jest więc identyczne, jak w OPEN i jest opisane w "Poradniku programisty Atari".

UWAGA: Nie należy otwierać kanału IOCB 7, ponieważ jest on używany przez system Action! do odczytu klawiatury. Podczas inicjowania systemu kanał ten jest otwierany dla urządzenia "K:", można więc wykorzystać go w programie bez otwierania.

Paddle

TYP: funkcja manipulatorów
DEFINICJA: BYTE FUNC Paddle(BYTE port)
PRZYKŁADY:

```
state = Paddle(2)
x = Paddle(y)
IF Paddle (a)>100 THEN RETURN FI
```

DZIAŁANIE: Zwraca wartość określającą położenie suwaka potencjometru przyłączonego do portu o numerze wskazywanym przez parametr. Wynik ten może przyjmować wartości z zakresu od 1 do 228. Dopuszczalne są wartości argumentu od 0 do 7.

Peek
PeekC

TYP: funkcja operacyjna
DEFINICJE: BYTE FUNC Peek(CARD address)
CARD FUNC PeekC(CARD address)
PRZYKŁADY:

```
a = Peek(x+5)
sc = Peek(88)+256*Peek(89)
sc = PeekC(88)
vec = PeekC(adr)
DO UNTIL Peek(53279)#7 OD
```

DZIAŁANIE: Zwracają liczbę typu BYTE lub CARD stanowiącą zawartość komórki lub komórek pamięci, których adres jest parametrem. Dozwolone są wartości parametru od 0 do 65535. Funkcja Peek odczytuje wartość jednobajtową z komórki <address>, zaś funkcja PeekC wartość dwubajtową z komórek <address> i <address>+1 (w kolejności LSB-MSB).

Ponieważ Action! umożliwia ustalenie adresu zmiennych i bezpośrednie odczytywanie ich wartości, to funkcje Peek i PeekC są wykorzystywane stosunkowo rzadka — zwykle do jednokrotnie odczytywanych rejestrów.

Plot

TYP: procedura graficzna
DEFINICJA: PROC Plot(CARD col, BYTE row)
PRZYKŁADY:

```
Plot(10,10)
Plot(h,v)
Plot(i*2,j+10)
```

DZIAŁANIE: Umieszcza na ekranie punkt (w trybach bitowych) lub znak (w trybach znakowych) w miejscu o podanych współrzędnych. Punkt 0,0 znajduje się w lewym, górnym rogu ekranu. Pierwsza liczba określa współrzędną poziomą, a druga

pionową. Przekroczenie dopuszczalnych wartości współrzędnych powoduje błąd. Wybór koloru <w trybach bitowych) lub znaku (w trybach znakowych), który zostanie narysowany, jest dokonywany przez zmienną color.

Point

TYP: procedura I/O
DEFINICJA: PROC Point(BYTE chan, CARD sector, BYTE offset)
PRZYKŁADY:

```
Point(2,100,12)
Point(1,x,1/125)
Point(iocb,sect,byte)
```

DZIAŁANIE: Ustawia wskaźnik położenia głowicy stacji dysków w kanale IOCB wskazanym przez pierwszy parametr według wartości podanych jako następne parametry. Pierwsza z nich określa numer sektora, a druga numer bajtu w sektorze. Dozwolone wartości zależą od gęstości zapisu dyskietki. Numer sektora może mieć wartość od 1 do 1040, a numer bajtu od 0 do 252.

Procedura Point działa tylko ze stacją dysków i wymaga uprzedniego otwarcia pliku do wymiany danych (tryb 12) przez procedurę Open.

Poke
PokeC

TYP: procedura operacyjna
DEFINICJE: PROC Poke(CARD address, BYTE value)
PROC PokeC(CARD address,value)
PRZYKŁADY:

```
Poke(708,2)
Poke(addr,nun)
PokeC(712,$1A6)
PokeC(dli,vec)
```

DZIAŁANIE: Umieszcza w komórce pamięci o adresie podanym jako pierwszy parametr wartość, która jest drugim parametrem. Pierwszy parametr jest z zakresu od 0 do 65535, a zakres drugiego określa jego typ. Procedura Poke zapisuje wartość jednobajtową w komórce <address>, zaś PokeC wartość dwubajtową w komórkach <address> i <address>+1 w kolejności LSB-MSB.

Ponieważ Action! umożliwia ustalenie adresu zmiennych i bezpośrednie odczytywanie ich wartości, to procedury Poke i PokeC są wykorzystywane stosunkowo rzadko - zwykle do jednokrotnie zapisywanych rejestrów.

Position

TYP: procedura graficzna
DEFINICJA: PROC Position(CARD col, BYTE row)
PRZYKŁADY:

```
Position(10,10)
Position(h,v)
Position(i*2,j+10)
```

DZIAŁANIE: Umieszcza kursor na ekranie w miejscu o podanych współrzędnych. Punkt 0,0 znajduje się w lewym, górnym rogu ekranu. Pierwszy argument określa współrzędną poziomą, a drugi pionową. Po wykonaniu funkcji położenie kursora pozostaje bez zmian, aż do następnej procedury wyjścia na ekran.

PrintD **PrintC**
PrintDE **PrintCD**
PrintE **PrintCDE**
PrintB **PrintI**
PrintBD **PrintID**
PrintBDE **PrintIDE**
PrintBE **PrintIE**

TYP: procedura I/O
DEFINICJE:
```
PROC Print(<string>)
PROC PrintE(<string>)
PROC PrintD(BYTE channel,<string>)
PROC PrintDE(BYTE channel,<string>)
PROC PrintB(BYTE number)
PROC PrintBE(BYTE number)
PROC PrintBD(BYTE channel,number)
PROC PrintBDE(BYTE channel,number)
PROC PrintC(CARD number)
PROC PrintCE(CARD number)
PROC PrintCD(BYTE channel, CARD number)
PROC PrintCDE(BYTE channel, CARD number)
PROC PrintI(INT number)
PROC PrintlE(INT number)
PROC Print ID(BYTE channel, INT number)
PROC Print IDE(BYTE channel, INT number)
```
PRZYKŁADY:
```
Print(text)
PrintC(1000)
PrintDE(6,"KOMPUTER ATARI")
PrintBD(2,22)
Print IDE(iocb,2+3*4-10/2)
```
DZIAŁANIE: Wysyła do urządzenia zewnętrznego podaną informację. Poszczególne procedury służą do zapisu wartości różnych typów, przez różne kanały oraz z lub bez znaku końca wiersza (EOL) na końcu zapisywanej informacji. Przeznaczenie i działanie procedur jest określone przez litery znajdujące się na końcu nazwy. Są to:

B - zapis liczb typu BYTE
C - zapis liczb typu CARD
D — zapis na wskazany IOCB
E — zapis znaku EOL na końcu
I - zapis liczb typu INT
brak — zapis tekstów

Procedury bez litery D wykonują zapis przez kanał ustalony zmienną device. W pozostałych procedurach konieczne jest podanie numeru kanału IOCB (z zakresu od 0 do 7).

PrintF

TYP: procedura I/O
DEFINICJA: PROC PrintF(<string>,<data>[,<data>...])
PRZYKŁADY:

```
PrintF("atari")
PrintF ("%S%E",text)
PrintF("%H = %I + %C%E",x,y,z)
PrintF("%C %U %H",65,65,65)
```

DZIAŁANIE: Wyświetla na ekranie w formacie określonym przez pierwszy parametr dane będące dalszymi parametrami. Jest to wiec funkcja wykonująca formatowany zapis na urządzeniu określonym przez zmienną device. W ciągu formatującym specjalne znaczenie ma znak "%". Oznacza on, że znajdujący się po nim znak będzie definiował format. Możliwe są następujące określenia formatu:

```
I — liczba dziesiętna ze znakiem
U - liczba dziesiętna bez znaku
H — liczba szesnastkowa bez znaku
C - znak
S - ciąg znaków
E - znak końca wiersza (EOL)
% - znak "%"
```

Liczba wyświetlanych argumentów nie może być większa niż liczba formatujących znaków "%", a tych z kolei nie może być więcej niż 5 w jednej procedurze. Wszystkie pozostałe znaki znajdujące się w ciągu określającym format są wyświetlane normalnie.

Poniżej podane są przykłady ilustrujące działanie procedury PrintF

procedura	rezultat
PrintF("atari") ;	atari
PrintF("=%S=","atari");	=atari=
PrintF("*%D*",77);	*77*
PrintF("*%I*",-77);	*-77*
PrintF("%C %U %H" ,65,65,65);	A 65 41

Ptrig

TYP: funkcja manipulatorów
DEFINICJA: BYTE FUNC Ptrig(BYTE port)
PRZYKŁADY:

```
x = Ptrig(2)
a = Ptrig(y)
IF Ptrig(a) THEN fire(a) FI
```

DZIAŁANIE: Zwraca wartość określającą stan przycisku potencjometru wskazanego przez parametr. Gdy przycisk jest wciśnięty, to wynikiem jest wartość* zero, a w przeciwnym przypadku jeden. Dopuszczalne są wartości argumentu od 0 do 7.

Put PutD
PutDE
PutE

TYP: procedura I/O
DEFINICJE: PROC Put(CHAR character)
PROC PutE()
PROC PutD(BYTE channel, CHAR character)
PROC PutDECBYTE channel, CHAR character)
PRZYKŁADY:

```
PutE() Put(byte)
PutD(iocb,128)
PutDE(2,'A)
```

DZIAŁANIE: Zapisuje jeden bajt informacji na urządzeniu zewnętrznym. Kanał do zapisu jest określony przez zmienną device lub pierwszy parametr procedury (dla procedur z literą D). Zapisywana wartość musi być z zakresu od 0 do 255. Procedury zawierające na końcu nazwy literę £ zapisują dodatkowo znak końca wiersza (EOL) - w przypadku PutE jest to jedyny znak przesyłany do urządzenia zewnętrznego.

Rand

TYP: funkcja operacyjna
DEFINICJA: BYTE FUNC Rand(BYTE range)
PRZYKŁADY:

```
los = Rand(100)
x = Rand(y)
IF Rand (100)<50 THEN PrintE("50 % szansy")
```

DZIAŁANIE: Zwraca wartość losową, która jest liczbą całkowitą mniejszą od parametru i większą lub równą zero, czyli z przedziału od 0 do range—1. Dla parametru równego 0 rezultat jest z zakresu od 0 do 255.

SAssign

TYP: procedura tekstowa
DEFINICJA: PROC SAssign(<dest>,<source>, BYTE start,stop)
PRZYKŁADY:

```
SAssign(dest,source,begin,end)
SAssign(name,"Atari",10,14)
SAssign(file;"OBRAZ",3,i)
```

DZIAŁANIE: Kopiuje zawartość ciągu znaków lub tablicy znakowej wskazanej drugim parametrem do fragmentu tablicy wskazanej przez pierwszy parametr. Kopiowany fragment jest umieszczany w ciągu docelowym od znaku wskazanego trzecim parametrem do znaku określonego przez czwarty. W przypadku niezgodnej długości ciągów procedura SAssign działa jak SCopy.

SCompare

TYP: funkcja tekstowa
DEFINICJA: INT FUNC SCompare(<stringlXstring2>>
PRZYKŁADY:

```
x = SCompare("Atari XL","Atari XE")
typ = SCompare(computer,"Atari")
IF SCompare(text1,text2)=0 THEN PrintE("rowne")
```

DZIAŁANIE: Zwraca wartość określającą rezultat porównania dwóch ciągów będących parametrami. Mogą to być same ciągi lub nazwy zawierających je tablic znakowych. Jeśli ciągi są równe, to wynikiem jest zero. Gdy pierwszy ciąg jest mniejszy, to funkcja zwraca wartość mniejszą od zera, a gdy większy, to większą od zera.

SCopy

TYP: procedura tekstowa
DEFINICJA: PROC SCopy<<dest>,<source>)
PRZYKŁADY:

```
SCopy(dest,source)
SCopy(name,"Atari")
SCopy(file,"D:DBRAZ.DAT")
```

DZIAŁANIE: Kopiuje zawartość ciągu znaków lub tablicy znakowej wskazanej drugim parametrem do tablicy wskazanej przez pierwszy parametr. Jeśli ciąg <dest> jest krótszy niż <source>, to kopiowana jest tylko cześć ciągu <source>. Jeśli <dest> jest dłuższy niż <source>, to kopiowany jest cały ciąg <source>, a reszta ciągu <dest> pozostaje bez zmian.

SCopyS

TYP: procedura tekstowa
DEFINICJA: PROC SCopyS(<dest>,<source>, BYTE start,stop)
PRZYKŁADY:

```
SCopyS(dest,source,begin,end)
SCopyS(name,"Komputer Atari XL",10,14)
SCopyS(file,"D:OBRAZ.DAT",3,i)
```

DZIAŁANIE: Kopiuje cześć zawartości ciągu znaków IUD tablicy znakowej wskazanej drugim parametrem do tablicy wskazanej przez pierwszy parametr. Kopiowany fragment rozpoczyna się od znaku wskazanego trzecim parametrem, a kończy na znaku określonym przez czwarty. W przypadku niezgodnej długości ciągów procedura SCopyS działa jak SCopy.

SetBlock

TYP: procedura operacyjna
DEFINICJA: PROC SetBlock(BYTE POINTER address, CARD size
BYTE value)
PRZYKŁADY:

```
SetBlock(*8600,512,32)
SetBlock(*600,256,'A)
```

```
SetBlock(screen,len,pattern)
SetBlock(chset,1024,code)
```

DZIAŁANIE: Wpisuje do tablicy wskazanej pierwszym parametrem od bajtu 0 do bajtu size—1 wartość określoną przez trzeci parametr. Procedura ta służy zwykle do wypełniania jedną wartością znacznych obszarów pamięci lub dużych tablic.

SetColor

TYP: procedura graficzna
DEFINICJA: PROC SetColor(BYTE register,hue,luminance)
PRZYKŁADYA

```
SetColor(2,0,0)
SetColor(i,j,k)
```

DZIAŁANIE: Ustala barwę i stopień jasności wskazanego koloru. Numer rejestru koloru jest określany przez pierwszy parametr, który może przyjmować wartości od 0 do 4. Drugi parametr oznacza barwą, zaś trzeci stopień jasności koloru. Jasność może mieć wartości parzyste od 0 do 14.

Ponieważ rejestry koloru są zwykłymi komórkami pamięci RAM, to zamiast SetColor(i,j,k) można zastosować Poke(708+i,16*j+k). W Action! jednak najczęściej stosuje się definiowanie tablicy BYTE ARRAY colr(5)=708, co umożliwia bezpośredni dostęp do rejestrów koloru.

SndRst

TYP: procedura dźwiękowa
DEFINICJA: PROC SndRst()
PRZYKŁAD:

```
SndRst()
```

DZIAŁANIE: Wyłącza wszystkie generatory dźwięku. Procedura ta nie wymaga żadnych parametrów.

Sound

TYP: procedura dźwiękowa
DEFINICJA: PROC Sound(BYTE voice,pitch,distortion,volume)
PRZYKŁADY:

```
Sound(0,200,10,10)
Sound(k,10*1,m,15-n)
Sound(k,0,0,0)
```

DZIAŁANIE: Ustawia generator dźwięku, którego numer jest pierwszym parametrem procedury (z zakresu od 0 do 3). Jeśli wszystkie pozostałe parametry są zerami, to następuje wyłączenie generatora. Drugi parametr procedury (od 0 do 255) określa okres dźwięku, a więc pośrednio jego częstotliwość. Parametr trzeci wybiera rodzaj zniekształceń tworzonego dźwięku (wartości parzyste od 0 do 14 — nieparzyste, wyłączają generator). Czysty ton uzyskuje się dla wartości 10 i 14. Ostatnim parametrem jest liczba z zakresu od 0 do 15, która ustala głośność dźwięku.

Stick

TYP: funkcja manipulatorów DEFINICJA:
BYTE FUNC Stick(BYTE port) PRZYKŁADY:

```
dir = Stick(1)
x = Stick(y)
IF Stick(a)=13 THEN h==-1 FI
```

DZIAŁANIE: Zwraca wartość określającą położenie joysticka przyłączonego do portu o numerze wskazywanym przez argument, który może być z zaksesu od 0 do 3. Wynik ten może przyjmować wartości z zakresu od 5 do 7, od 9 do 11 oraz od 13 do 15 według zamieszczonego poniżej schematu.

```
10 14  6
  \ | /
11- 15 -7
  / | \
 9 13  5
```

StrB
StrC
StrI

TYP: procedura operacyjna
DEFINICJE: PROC StrB<BYTE number,<string>)

```
PROC StrC(CARD number,<string>)
PROC StrI(INT number,<string>)
```

PRZYKŁADY:

```
StrB(x,val)
StrC(100,text)
z=ValI("-12") StrI(z,a)
```

DZIAŁANIE: Zamienia pierwszy parametr (liczbowy) na reprezentujący go ciąg cyfr, który jest umieszczany w tablicy znakowej wskazanej drugim parametrem. Postać ciągu wynikowego jest identyczna z uzyskiwaną na ekranie przy użyciu procedury Print. Wynik procedury Strl poza cyframi może zawierać znak "-". Odwrotnością procedury Str jest funkcja Val.

Strig

TYP: funkcja manipulatorów
DEFINICJA: BYTE FUNC Strig(BYTE port)
PRZYKŁADY:

```
fire = Strig(0)
x = Strig(y)
IF Strig(a)=0 THEN fire(a) FI
```

DZIAŁANIE: Zwraca wartość określającą stan przycisku joysticka przyłączonego do gniazda wskazanego przez argument. Sdy przycisk jest wciśnięty, to wynikiem jest zero, a w przeciwnym przypadku jeden. Argument może mieć wartości z zakresu od 0 do 3.

TRACE

TYP: zmienna biblioteczna
DEFINICJA: BYTE TRACE
PRZYKŁADY:

```
BET TRACE=0
SET TRACE=1
```

DZIAŁANIE: Steruje wariantem "TRACE" kompilatora. Musi być użyta w dyrektywie SET i musi się znajdować na początku programu. Przypisanie zmiennej TRACE wartości 1 włącza siedzenie programu, a wartości zero — wyłącza.

ValB
ValC
ValI

TYP: funkcja operacyjna
DEFINICJE: BYTE FUNC ValB(<string>)
CARD FUNC ValC(<string>) INT
FUNC ValI(<string>)
PRZYKŁADY:

```
x = ValB("5")
a = ValC(num)+ValI("-656")
z=ValI("-12") Strl(z,a)
```

DZIAkANIE: Zwraca wynik zamiany parametru tekstowego na reprezentowaną przez niego liczbę typu zgodnego z typem funkcji. Zamiana jest dokonywana znak po znaku, od lewej do prawej, aż do napotkania niedozwolonego znaku. Jeżeli argument funkcji nie zawiera żadnego poprawnego znaku, to sygnalizowany jest błąd. Dopuszczalne są tylko cyfry oraz znak "-" w funkcji ValI. Odwrotnością funkcji Val jest procedura Str.

XIO

TYP: procedura I/O
DEFINICJA: PROC XIO(BYTE chan,0,end,aux1,aux2,
<filestring>)
PRZYKŁADY:

```
XI0(1,0,254,0,0,"Ds")
XI0(iocb,0,3S,0,0,file)
XI0(io,0,com,al,a2,name)
```

DZIAŁANIE: Procedura XI0 służy do wykonania poprzez wskazany kanał IOCB operacji I/O, której kod jest podany jako parametr cmd. Większość operacji, które dotyczą standardowych urządzeń systemu Atari - ekran, drukarka, magnetofon i stacja dysków, może być zrealizowane przez inne procedury, więc XIO jest w programach pisanych w Action! wykorzystywana zwykle tylko do uzyskania funkcji DOS-u.

Procedura XI0 jest dokładnym odpowiednikiem instrukcji XI0 w Atari Basic. Szczegółowy opis jej działania oraz znaczenia poszczególnych parametrów znajduje się w książce "Poradnik programisty Atari".

Zero

TYP: procedura operacyjna
DEFINICJA: PROC Zero(BYTE POINTER address, CARD size)
PRZYKŁADY:

```
Zero(array,length)
Zero(table,20)
Zero(string,x*$80)
```

DZIAŁANIE: Zeruje tablice wskazaną pierwszym argumentem od bajtu 0 do bajtu size-1. Procedura ta jest najczęściej wykorzystywana do kasowania znacznych obszarów pamięci i inicjowania dużych tablic. Przy inicjowaniu tablic typu CARD i INT jako <size> należy podać podwojony rozmiar tej tablicy, gdyż każda taka liczba zajmuje dwa bajty.

Dodatkowe procedury uzupełniające standardową bibliotekę Action! są zamieszczane w "Bajtku". Dotychczas opublikowano następujące rodzaje procedur:

	Bajtek
Liczby rzeczywiste	11/88
Procedury graficzne	12/88
Funkcje trygonometryczne	2/89
Procedury wejścia/wyjścia	4/89
Grafika graczy i pocisków	7/89

3. 7. Kody błędów

Kompilator Action! sygnalizuje błędy składniowe wykryte podczas kompilacji przez wskazanie miejsca wystąpienia i podanie liczbowego kodu błędu. U czasie wykonywania programu mogą pojawić się błędy spowodowane użyciem niewłaściwych wartości lub wykonaniem instrukcji, które są w aktualnej chwili niepoprawne. Błędy te, zwane błędami wykonania, są sygnalizowane dopiero podczas działania programu.

Poniżej przedstawione są meldunki błędów sygnalizowanych przez kompilator oraz ich krótki opis.

0 — Out of system memory
Zbyt mały obszar dostępnej pamięci. Jeśli błąd wystąpił podczas kompilacji, to konieczne jest kompilowanie programu z dyskietki lub kasety.

1 — Missing "
Brak cudzysłowu (") w ciągu tekstowym.

2 - Nested DEFINE
Zagnieżdżona dyrektywa DEFINE. Dyrektywy DEFINE nie mogą występować w innych dyrektywach DEFINE.

3 - Global variable symbol table full
Brak miejsca w tablicy symboli zmiennych globalnych.

4 - Local variable symbol table full
Brak miejsca w tablicy symboli zmiennych lokalnych.

5 - SET directive syntax error
Błąd składniowy w dyrektywie SET.

6 — Declaration error
Nieprawidłowy format deklaracji zmiennej, procedury lub funkcji.

7 — Invalid argument list
Zbyt dużo parametrów zostało umieszczone w instrukcji albo wywołaniu procedury lub funkcji.

8 — Variable not declared
Kompilator napotkał identyfikator, który nie został uprzednia zadeklarowany.

9 - Not a constant
Użyta zmiennej w miejscu, w którym wymagane jest zastosowanie stałej.

10 - Illegal assignment
Niedozwolona forma instrukcji przypisania, na przykład przypisanie zmiennej wartości logicznej <x=5<7).

11 — Unknown error
Wystąpił błąd, lecz system Action! nie może rozpoznać jego rodzaju.

12 - Missing THEN
Brak słowa THEN po warunku w instrukcji IF.

13 - Missing FI
Brak słowa FI kończącego instrukcję IF.

14 - Out of code space
Zbyt mały obszar pamięci dla programu wynikowego. Konieczna jest kompilacja z dyskietki lub kasety.

15 - Missing DO
Brak instrukcji DO rozpoczynającej pętlę instrukcji FOR, WHILE lub UNTIL.

16 — Missing TO
Brak słowa TO w instrukcji FOR.

17 - Bad expression
Nieprawidłowy format wyrażenia arytmetycznego lub warunkowego.

18 - Unmatched parentheses
Niezamknięty nawias, czyli brak nawiasu z lewej lub prawej strony.

19 - Missing OD
Brak słowa OD kończącego pętlę DO/OD.

20 - Can't allocate memory
System Action! nie może przemieścić większego obszaru swojej pamięci operacyjnej.

21 — Illegal array reference
Forma odwołania do tablicy lub jej elementu jest nieprawidłowa.

22 — Input file too large
Plik źródłowy do kompilacji jest zbyt duży i musi być podzielony na mniejsze części.

23 — Illegal conditional expression
Niedozwolony format wyrażenia warunkowego lub wyrażenie to zostało użyte w niedozwolonym miejscu.

24- Illegal FOR syntax
Nieprawidłowa składnia instrukcji FOR.

25- Illegal EXIT
Instrukcja EXIT została użyta w niewłaściwym miejscu lub w niedozwolony sposób.

26 - Nesting too deep
Nastąpiło zbyt głębokie zagnieżdżenie programu (dozwolone jest maksymalnie 16 poziomów).

27 - Illegal TYPE syntax
Niedozwolona składnia definicji rekordu TYPE.

28 - Illegal RETURN
Instrukcja RETURN została użyta w niewłaściwym miejscu lub w niedozwolony sposób.

61 - Out of symbol table space
Brak miejsca w tablicy symboli kompilatora.

DODATKI

A. Kody błędów I/O

Błędy powstające w systemie operacyjnym Atari są sygnalizowane w większości języków programowania tylko przez podanie kodu tego błędu (w Deep Blue C z minusem). Prawie wszystkie kody błędów systemu są związane z operacjami wejścia/wyjścia (Input/Output), a wszystkie mają kody o wartości przekraczającej 127.

Poniżej przedstawione są kody błędów systemu operacyjnego wraz z krótkim objaśnieniem.

128 - BREAK ABORT
Podczas wykonywania operacji I/O został naciśnięty klawisz <BREAK>, co spowodowało przerwanie operacji.

129 - PREVIOUS OPEN
Podjęto próbę otwarcia kanału IOCB, który był już uprzednio otwarty - ponowne użycie OPEN dla tego samego kanału.

130 - NON EXISTANT DEVICE
Podjęto próbę komunikacji z urządzeniem, które nie jest zainstalowane w systemie operacyjnym.

131 - WRITE ONLY
Podjęta została próba odczytu z pliku otwartego tylko do zapisu.

132 - INVALID COMMAND
M uniwersalnej procedurze I/O (XIO) został użyty niedozwolony kod rozkazu.

133 - IOCB NOT OPEN
Podjęta została próba wykonania operacji wejścia/wyjścia przez blok IOCB, który nie został otwarty - brak OPEN.

134 - BAD IOCB NUMBER
Użyty został numer kanału IOCB, który nie jest liczbą z przedziału od 0 do 7.

135 - READ ONLY
Podjęta została próba zapisu do pliku otwartego tylko do odczytu.

136 - END OF FILE
Podjęta została próba odczytu z pliku, w którym osiągnięto koniec, a więc próba odczytu spoza pliku.

137 - TRUNCATED RECORD
Odczytany został rekord, który miał długość większą od długości przeznaczonego dla niego bufora.

138 - TIMEOUT ERROR
Upłynął określony przez system operacyjny czas, w jakim urządzenie powinno wykonać żądaną operację.

139 - DEVICE NACK ERROR
Urządzenie nie potrafi wykonać żądanej operacji pomimo, iż komunikacja przebiega prawidłowo.

140 - FRAMING ERROR
Nieprawidłowy przebieg transmisji - zła forma lub nierówne odstępy miedzy przesyłanymi bitami.

141 - CURSOR OVERRAN6E
W procedurze graficznej (POSITION, PLOT, DRAWTO, LOCATE lub FILL) użyte zostały parametry przekraczające zakres dopuszczalny dla aktualnego trybu graficznego.

142 - SIO OVERRUN
Nieprawidłowy przebieg transmisji do lub z urządzenia.

143 - SIO CHECKSUM
Suma kontrolna obliczona dla przesyłanego bloku informacji jest różna od przesłanej razem z tym blokiem.

144 - DEVICE DONE
Urządzenie nie jest w stanie wykonać żądanej operacji pomimo, iż jest to poprawna operacja.

145 - BAD SCREEN MODE
Procedura graficzna nie może być wykonana w aktualnie wykorzystywanym trybie pracy ekranu.

146 - FUNCTION NOT IMPLEMENTED
Procedury obsługi urządzenia nie przewidują wykonania żądanej operacji I/O.

147 - INSUFFICIENT SCREEN MEMORY
Brak wolnej pamięci operacyjnej na utworzenie obrazu we wskazanym trybie graficznym.

150 - SERIAL PORT OPEN
Podjęto próbę otwarcia portu RS, który był już uprzednio otwarty poprzez inny kanał IOCB (dotyczy tylko współpracy z interfejsem RS-232).

151 - CONCURRENT MODE NOT ENABLED
Podjęto próbę rozpoczęcia transmisji współbieżnej przez port RS, który został otwarty bez zezwolenia na taką transmisję (dotyczy tylko współpracy z interfejsem RS-232).

152 - BUFFER ERROR
Podjęto próbę uruchomienia transmisji współbieżnej z ustalonym przez użytkownika buforem o niewłaściwych parametrach (dotyczy tylko współpracy z interfejsem RS-232).

153 - CONCURRENT MODE ACTIVE
Podjęto próbę komunikacji z urządzeniem podczas, gdy aktywny jest współbieżny tryb pracy portu RS (dotyczy tylko współpracy z interfejsem RS—232).

154 - CONCURRENT MODE NOT ACTIVE
Podjęto próbę odczytu poprzez port RS bez uruchomienia współbieżnego trybu pracy (dotyczy tylko współpracy z interfejsem RS-232).

160 - BAD DRIVE NUMBER
Podany został numer stacji dysków niedopuszczalny w aktualnej konfiguracji użytego DOS-u.

161 - TOO MANY FILES
Równocześnie próbowano otworzyć liczbę plików dyskowych, która przekracza dozwoloną w aktualnej konfiguracji DOS—u.

162 - DISK FULL
Wszystkie sektory dyskietki dostępne dla użytkownika zostały już zapisane i nie ma miejsca na więcej informacji.

163 - I/O ERROR
Błędne wykonanie operacji przez Dyskowy System Operacyjny.

164 - BAD FILE NUMBER
Odczytany sektor dyskietki należy do innego pliku niż aktualnie używany - dyskietka jest uszkodzona lub nieprawidłowo zapisana.

165 - BAD FILENAME
Użyta została nazwa pliku zawierająca niedozwolone znaki.

166 - POINT ERROR
Błąd wykonania procedury POINT spowodowany nieprawidłowym parametrem.

167 - FILE PROTECTED
Podjęta została próba zapisu, skasowania lub zmiany nazwy pliku, który jest zabezpieczony przed zapisem.

168 - BAD COMMAND
W uniwersalnej procedurze I/O (XIO) został użyty niedozwolony kod rozkazu dla stacji dysków.

169 - DIRECTORY FULL
Nie można zapisać na dyskietce nowego pliku, gdyż przekroczona zostałaby dopuszczalna ich liczba.

170 - FILE NOT FOUND
Na dyskietce we wskazanej stacji dysków nie ma pliku o podanej nazwie.

171 - BAD POINT
Błąd wykonania procedury POINT spowodowany niewłaściwymi parametrami.

172 - PROHIBITED APPEND
Niemożliwe jest odczytanie dyskietki — prawdopodobnie użyty niewłaściwy DOS.

173 - BAD FORMAT
Odczytywana dyskietka ma uszkodzony sektor lub nie jest sformatowana albo została sformatowana przy użyciu innego DOS-u.

174 - DUPLICATE FILENAME
Podjęto próbę zapisu pliku o nazwie, która została użyta dla pliku istnijacego na dyskietce (błąd sygnalizowany tylko przez niektóre DOS—y).

175 - BAD LOAD FILE
Podjęto próbę odczytu poprzez DOS pliku, który nie jest zapisany w standardzie DOS-u (brak nagłówka).

176 - PROHIBITED APPEND
Podjęta została próba uzyskania dostępu do pliku, który ma niewłaściwy format lub znajduje się na dyskietce o innym formacie.

177 - BAD DISK
Odczytywana dyskietka ma uszkodzony sektor lub nie jest sformatowana albo została sformatowana przy użyciu innego DOS-u.

B. Tabela instrukcji

Dodatek ten zawiera wykaz instrukcji, procedur i funkcji dostępnych w opisanych w książce językach. Ponadto dla porównania zamieszczone zostały również instrukcje Atari Basic. Poniższa tabela może być bardzo pomocna przy tłumaczeniu programów pomiędzy różnymi językami.

Atari Basic	Kyan Pascal	Deep Blue C	Ac t i on!
ABS	ABS	c abs	-
ADR	—	@	@
AND	AND	& / &&	& / AND
ASC	-	-	-
ATN	ARCTAN	c_atan	-
—	BEGIN	$(	—
-	BOOLEAN	-	-
-	-	break	EXIT
BYE	—	—	—
—	CHAR	char	BYTE/CHAR
-	-	chget	-
—	—	choget	—
-	-	chput	-
CHR$	CHR	—	—
—		clear	Zero
CLOAD	-	_	-
CL0G	-	c_loq10	-
CLOSE	—	close/cclose	Close
CLR	-	-	-
COLOR	-	color	color
—	CONCAT	—	—
—	CONST	—	—
CONT	-	-	P (monitor)
-	-	continue	-
COS	COS	c cos	—
CSAVE	-	-	-
-	-	c_alog10	-
—	—	C.chs	—
-	-	c_cmp	-
—	—	c ddms	—
—	—	c dmsd	—
—	—	c dr	—
-	-	c_fpi	-
—	—	c fpsi	—
-	-	c_ifp	-
—	—	c iral	—
-	-	c_itrig	-
—		c_rd	
-	-	c sifp	-
		c tan	
DATA	-	-	-
DEG	—	c rad	—
-	-	-	device
DIM/COM	ARRAY		ARRAY

Atari Basic	Kyan Pascal	Deep Blue C	Action!
-	DISPOSE	-	-
-	DIV	/	/
—	—	—	DO...OD
DOS	-	-	-
-	-	dpeek	PeekC
—	ASSIGN	dpoke	PokeC
DRAWTO	DRAWTO	drawto	DrawTo
END	—	-	
—	END	$)	
ENTER			
-	EOF	-	EOF
—	EOLN	-	
EXP	EXP	c_alog	-
—	—	extern	-
—	FALSE	—	
-	FILE	-	-
-	-	fill	Fill
—	—	find	
FOR TO STEP ... NEXT	FOR TO/DOWNTO DO ...	for (...) ...	FOR TO STEP DO ... OD
-	FORWARD	-	-
-	-	fprintf	-
FRE	—	—	-
-	FUNCTION	*2)	FUNC
GET	GET	cgetc/getchar	GetD
GOSUB			
GOTO	GOTO	-	-
GRAPHICS	GRAPHICS	graphics	Graphics
-	—	hitclear	—
—	-	hitp2pf	-
-	—	hitpγpl	—
—	—	hstick	—
-	—	hval	
IF ... THEN	IF ... THEN ... ELSE ...	if ... ... else ...	IF ... THEN ... ELSEIF ... THEN ... ELSE ... FI
—	IN	-	
-	INDEX	-	_
INPUT	READ/READLN	gets	Input
INPUT #	READ/READLN	—	InputD
—	INPUT	-	-
INT	ROUND	c int	—
—	INTEGER	int	CARD/INT
—	LABEL	—	
LEN	LENGTH	strlen	-
LET / =	: =	= / c move	
LIST	—		
LOAD			
LOCATE	LOCATE	locate	Locate
LOG	LN	c_log	-
LPRINT	PRON PROFF	-	

Atari Basic	Kyan Pascal	Deep Blue C	Action!
	_	<<	LSH
-	MAXINT	-	-
-	MOD	%	MOD
-	-	-	MODULE
—	—	move	MoveBlock
NEW	-	-	-
-	NEW	—	—
-	NIL	-	-
—	—	normalize	—
NOT	NOT	! / $-	-
NOTE	—	—	Note
-	ODD	-	-
ON ... GOTO/GOSUB	CASE ... OF ...	switch (...) case ... default	-
OPEN	RESET/REWRITE	copen/open	Open
OR	OR	| / ||	X / OR
-	OUTPUT	_	-
-	PACKED	-	-
PADDLE	-	paddle	Paddle
PEEK	—	peek	Peek
-	-	pladdr	-
—	—	pimove	—
PLOT	PLOT	plot	Plo
-	-	pmcflush	-
-	-	pmcinit	-
-	-	pmclear	-
—	—	pmcolor	—
-	-	pmgraphics	-
—	—	pmwidth	—
POINT	SEEK	—	Point
—	—	—	POINTER
POKE	ASSIGN	poke	Poke
POP	—	—	—
POSITION	POSITION	position	Position
-	PRED	-	-
PRINT	WRITE/WRITELN	cprints	Print
PRINT #	WRITE/WRITELN	cputs	PrintD
-	-	printf	PrintF
—	PROCEDURE	*٢)	PROC
*٣)	PROGRAM	*٣)	*٣)
PTRIG	-	ptrig	Ptrig
PUT	PUT	cputc/putchar	Put/PutD
RAD	—	c rad	—
READ	—	—	—
—	REAL	—	—
-	RECORD	-	TYPE
REM ...	(* ... *)	/* ... */	;
—	REPEAT ... UNTIL ...	do ... while ...	DO ... UNTIL ... OD
RESTORE	-	-	-
RETURN	-	return	RETURN
RND	RANDOM	rnd	Rand
-	-	>>	RSH

Atari Basic	Kyan Pascal	Deep Blue C	Action!
RUN	CHAIN	-	R (monitor)
-	-	-	SAssign
SAVE	—	—	—
-	-	-	SCompare
-	SET	—	—
-	-	-	SetBlock
SETCOLOR	SETCOLOR	setcolor	SetColor
SIN	-	-	-
SIN	SIN	c_sin	-
-	-	-	SndRst
SOUND	SOUND	sound	Sound
-	SQR	-	-
SQR	SQRT	c_sqrt	-
STATUS	—	—	—
STICK	-	stick	Stick
-	-	strcpy	SCopy
STRIG	-	strig	Strig
STOP	—	—	Break
STR$	—	c fasc	Str
-	SUBSTRING	-	SCopyS
—	SUCC	—	—
—	TEXT	—	CHAR ARRAY
-	-	tolower	-
-	-	toupper	-
TRAP	—	—	Error
-	TRUE	-	-
—	TRUNC	c f rac	—
-	TYPE	-	-
USR	—	asm/usr	—
VAL	ORD	val/c_afp	Val
—	VAR	—	—
-	-	vstick	-
—	WHILE ... DO ...	while	WHILE ... DO ... OD
-	WITH	-	-
XIO	-	ciov	XIO
—	—	—	! / XOR
-	-	#define	DEFINE
-	#I	#include	INCLUDE
+	+	+ / c_fadd	+
—	—	- / c fsub	—
*	*	* / c fmul	*
/	/	c_fdiv	-
^	^	c exp	—

*1) zamiast GOSUB trzeba wywołać procedure lub funkcje,
*2) nie ma słowa kluczowego przed deklaracją,
*3) nie ma słowa kluczowego na początku programu.