# ATARI MICROSOFT BASIC II

Podręcznik języka programowania

*Bluki, 2013*

# Spis treści

# Alfabetyczny spis poleceń

# Wstęp

Atari Microsoft BASIC II (AMB) posiada w słowniku ponad 100 poleceń, nazywanych również słowami zastrzeżonymi lub słowami kluczowymi. Ten język programowania używa angielsko-podobnych słów jako poleceń. Aby słowo kluczowe zostało rozpoznane, musi być napisane poprawnie dużymi literami. Nie można też używać w programie poleceń jako nazw zmiennych, np. GOTO, ale RGOTO już tak. Alfabetyczny spis wszystkich poleceń znajduje się na poprzedniej stronie.

W zwykłym języku stosowane są znaki interpunkcyjne. AMB również ich używa, ale spełniają inną rolę.

Atari Microsoft BASIC II znajduje się na kartridżu, jednak część poleceń dostępna jest po wczytaniu rozszerzenia z dyskietki. Te polecenia opisane są jako „dostępne tylko z rozszerzeniem na dyskietce". Jeśli więc rozszerzenie nie zostało wczytane podczas uruchamiania systemu, nie można korzystać z tych poleceń.

Większość odmian BASIC-a pozostawia 6 stronę pamięci wolną. AMB używa jej na własne potrzeby, posiada jednak polecenie (OPTION RESERVE) umożliwiające zarezerwowanie wolnego obszaru pamięci RAM na potrzeby użytkownika.

Minimalne wymagania sprzętowe to komputer Atari z pamięcią 16kB i stacją dysków lub magnetofonem.

UWAGI.
Został przyjęty następujący sposób prezentacji poleceń:

- nazwa polecenia;
- format;
- przykład (przykłady);
- opis polecenia;
- ilustracja w formie listingu (przy niektórych poleceniach).

↦[polecenie] - patrz... (np.: ↦UNLOCK - patrz polecenie UNLOCK).

## CLOAD

CLOAD
Odczytuje program z kasety (w formacie stokenizowanym).

## CSAVE

CSAVE
Zapisuje program na kasecie (w formacie stokenizowanym).

## DOS

DOS
Powoduje opuszczenie BASIC-a i przejście do menu DOS-a. Daje to dostęp do wszystkich poleceń DOS-a. Powrót do Microsoft BASIC-a II opcją „B" DOS-a.

## KILL

KILL"D:nazwa"
KILL"D2:PROGRAM1.AMB"
Kasuje program o wskazanej *nazwie*, znajdujący się na dyskietce.

## LIST

LIST"urządzenie:nazwa" od_nr_linii-do_nr_linii
LIST"D:PROGRAM1.AMB"
LIST"C:"
LIST"P:" 100-500
Wysyła listing programu od *nr_linii* do *nr_linii* na wskazane *urządzenie* (stacja dysków, magnetofon, drukarka itp.). Numery linii mogą być pominięte, wówczas wysłany zostanie cały listing. W przypadku stacji dysków należy określić *nazwę* pod jaką listing zostanie zapisany.

LIST
LIST 100
LIST 100-500
LIST 100-
LIST -500
Wysyła listing programu na ekran, w przykładach kolejno: cały, tylko linia nr 100, od linii 100 do linii 500, od linii 100 do końca programu, od początku programu do linii 500.

## LOAD

LOAD"urządzenie:nazwa"
LOAD"D:PROGRAM.AMB"
LOAD"C:"
Wczytuje program do pamięci komputera ze wskazanego *urządzenia* o wskazanej *nazwie.* W przypadku magnetofonu *nazwa* jest zbędna.

## LOCK

LOCK"D:nazwa"
LOCK"D:PROGRAM.AMB"
Zabezpiecza plik znajdujący się dysku przed skasowaniem lub modyfikacjami (↦UNLOCK).

## MERGE

MERGE"urządzenie:nazwa"
MERGE"C:"
MERGE"D2:PROGRAM2.AMB"
Dołącza program z *urządzenia* (dyskietki - o wskazanej *nazwie* lub kasety) do programu znajdującego się w pamięci komputera. Jeśli dojdzie do zdublowania numerów linii, to linia znajdująca się w pamięci zostanie zastąpiona przez doczytywaną. Polecenia MERGE można użyć tylko wtedy, gdy program wcześniej został zapisany poleceniem LIST.

## NAME...TO (Dostępne tylko z rozszerzeniem na dyskietce).

NAME"D:stara"TO"nowa"
NAME"D:GRA.AMB"TO"NOWAGRA.AMB"
Zmienia nazwę pliku ze *starej* na *nową*.

## NOTE (Dostępne tylko z rozszerzeniem na dyskietce).

NOTE#iocb,sektor,bajt
NOTE#1,I,J
Odczytuje bieżący numer sektora i bajtu na dysku i podstawia do odpowiednich zmiennych. Należy pamiętać o wcześniejszym otworzeniu kanału IOCB (↦OPEN).

## RUN

RUN"urządzenie:nazwa"
RUN"D:GRA.AMB"
RUN"C:"
Wczytuje program o wskazanej *nazwie* z wybranego *urządzenia* i uruchamia go. Odczyt z magnetofonu nie wymaga *nazwy*.

RUN
RUN 100
Uruchamia program znajdujący się w pamięci komputera. Jeżeli zostanie podany numer linii, to start nastąpi od niej, w innym przypadku od pierwszej linii programu.

## SAVE

SAVE"urządzenie:nazwa"
SAVE"D2:GRA.AMB"
SAVE"C:"
Zapisuje program na wybranym *urządzeniu* o wskazanej *nazwie* w formacie stokenizowanym. W przypadku magnetofonu *nazwa* jest zbędna.

SAVE"D:nazwa"LOCK
SAVE"D:GRA.AMB"LOCK
SAVE"C:"LOCK
Zapisuje program i zabezpiecza go przed listowaniem i modyfikacjami kodu. Tak zabezpieczony program można wczytać i uruchomić poleceniem RUN"D:nazwa" (RUN"C:"), ewentualnie LOAD"D:nazwa" i następnie wykonać RUN. Uwaga. Te polecenie nie jest tożsame z LOCK"D:nazwa". Należy pamiętać o zachowaniu niezabezpieczonej kopii programu w razie potrzeby dokonania późniejszych poprawek lub do celów archiwalnych.

## UNLOCK

UNLOCK"D:nazwa"
UNLOCK"D:TEST.AMB"
Odbezpiecza plik na dysku, który wcześniej został zabezpieczony przed zapisem poleceniem LOCK. Użycie UNLOCK w stosunku do programu niezabezpieczonego nie powoduje błędu.

## VERIFY (Dostępne tylko z rozszerzeniem na dyskietce).

```
VERIFY"urządzenie:nazwa"
VERIFY"D:GRA.AMB
VERIFY"C:"
```

Sprawdza poprawność zapisanego programu na wybranym nośniku przez porównanie go z tym w pamięci. Jeśli pojawi się niezgodność otrzymasz komunikat „TYPE MISMATCH ERROR”.

# Operacje wejścia/wyjścia (I/O)

## CLOSE

```
CLOSE #iocb
CLOSE #5
```

Zamyka operacje na plikach przeprowadzane przez wybrany blok IOCB (1 ÷ 7), ↦OPEN.

## EOF

```
EOF(nr_iocb)
X=EOF(1)
```

Wykrywa koniec pliku (end-of-file), czyli sprawdza, czy zostały już pobrane wszystkie dane z określonego pliku poprzez blok IOCB (*nr_iocb*). Jeśli tak, zwraca wartość -1, jeśli nie – 0.

```
100 OPEN#1,"D:TEST.DAT"INPUT
120 GET#1,X:PRINT CHR$(X);
140 IF EOF(1)=0 THEN 120
```

## GET

```
GET#iocb,nazwa_zmiennej...
GET#iocb,AT(sektor,bajt);nazwa_zmiennej...
GET#1,B
GET#5,AT(144,1);A,B,LICZBA,X
```

Z kanału IOCB pobiera jeden bajt i przypisuje go zmiennej (lub kolejne bajty kolejnym zmiennym). Przy pobieraniu danych z dysku, po słowie kluczowym TO, można określić sektor i bajt od którego nastąpi pobieranie danych. Należy pamiętać, aby wcześniej otworzyć kanał do transmisji (↦OPEN), patrz też niżej PUT.

```
10 OPEN#5,"K:"INPUT
20 GET#5,K:PRINT K
30 IF CHR$(K)="E" THEN END
40 GOTO 20
```

## OPEN

```
OPEN#iocb,"urządzenie:nazwa_programu"rodzaj_dostępu
OPEN#1,"K:"INPUT
OPEN#7,"P:"OUTPUT
OPEN#2,"D:PRG.DAT"UPDATE
OPEN#4,"D2:"PRG.SCR"APPEND
```

Otwiera blok kontroli wejścia/wyjścia (IOCB) umożliwiający transmisję danych z wybranymi urządzeniami.

**#** – obowiązkowy znak wprowadzany przez użytkownika;
**iocb** – blok wejścia/wyjścia. Należy tu wpisać numer z zakresu 1 ÷ 7;
**"urządzenie:nazwa_programu"** – *urządzenie* to: D: (dysk), P: (drukarka), E: (edytor ekranowy), S: (ekran), C: (kaseta/magnetofon), R: (RS232).

Korzystanie z D: wymaga podania po dwukropku *nazwy_programu* składającej się z jednego do ośmiu znaków - liter lub cyfr - oraz (opcjonalnie) po kropce rozszerzenia składającego się z maksimum trzech znaków. Pierwszym znakiem *nazwy_programu* nie może być cyfra.

Więcej na temat korzystania z urządzeń zewnętrznych znajdziesz w instrukcji DOS-a, którego używasz;

**rodzaj_dostępu** - dopuszczalne są następujące operacje:
INPUT - możliwy tylko odczyt (wejście);
OUTPUT - możliwy tylko zapis (wyjście);
UPDATE - możliwy zarówno odczyt i zapis;
APPEND - dołączanie. Możliwe jest dodawanie nowych danych na końcu już istniejącego pliku.

## PUT

```
PUT#iocb,wyrażenie_liczbowe
PUT#iocb,AT(sektor,bajt);wyrażenie_liczbowe
PUT#1,X
PUT#1,255,A,B,C
```

Wysyła na urządzenie zewnętrzne, poprzez blok IOCB (*#iocb*), pojedynczy bajt. Poniższy program tworzy na dysku plik o nazwie „DANE" i zapisuje w nim kolejno dwa bajty. Do odczytu tak zapisanych informacji służy np. polecenie GET.

```
10 OPEN#1,"D:DANE"OUTPUT
20 PUT#1,47,56
30 CLOSE#1
```

## STATUS

```
STATUS(nr_iocb)
A=STATUS("urządzenie:nazwa")
S=STATUS(6)
A=STATUS("D:MIKRO.BAS")
```

Zwraca wartość czwartego bajtu bloku IOCB (bajt statusu). Najbardziej znaczący bit jest ustawiony gdy wystąpił błąd, pozostałe bity reprezentują numer błędu. Zero oznacza brak błędu.
Kody funkcji STATUS znajdują się w aneksie II.

# Edycja programu

## AUTO

```
AUTO od_nr_linii,skok
AUTO 500,10
AUTO 500,
AUTO ,20
AUTO
```

Automatyczna numeracja linii programu począwszy *od_nr_linii* i ze wskazanym *skokiem*. W pierwszym przykładzie automatyczna numeracja rozpocznie się od numeru linii 500 ze skokiem 10, czyli następne numery linii to 510, 520, 530 itd. Pominięcie *od_nr_linii* spowoduje rozpoczęcie autonumeracji od numeru 100. W przypadku pominięcia *skoku* przyjęta domyślnie wartość to 10. Jeżeli linia o wybranym numerze już istnieje, zostanie wyświetlona jej zawartość.

## CONT

```
CONT
```

Wznawia program zatrzymany poleceniem ↦STOP

## DEL (Dostępne tylko z rozszerzeniem na dyskietce).

```
DEL od_nr_linii,do_nr_linii
DEL 100
DEL 100-500
DEL 100-
DEL -500
```

Kasuje wybrane linie programu. W przykładzie kolejno: linię nr 100, linie w przedziale od 100 do 500, linie od nr 100 do końca programu i linie od początku programu do linii 500.

## ERL

```
PRINT ERL
```

Zwraca numer linii, w której nastąpiło przerwanie programu w wyniku błędu lub naciśnięcia klawisza BREAK. Użycie w programie polecenia STOP nie wywołuje błędu.

## ERR

```
PRINT ERR
```

Zwraca numer ostatniego błędu, który się pojawił (↦ Aneks I). Jeśli błędu nie było, to ERR=0. Polecenie STOP nie powoduje błędu, a użycie klawisza BREAK błąd nr 128.

## ERROR

```
ERROR kod_błędu
ERROR 6
ERROR 165
```

Wymusza na interpreterze błąd zgodny z listą komunikatów błędów (Aneks I). Może być to pomocne w fazie testowania programu, gdy chcemy przekonać się jak zareaguje program na wystąpienie określonego błędu.

## FRE(0)

```
PRINT FRE(0)
```

Podaje, w bajtach, ilość wolnej pamięci RAM. Wartość ta zmienia się na różnych etapach wykonywania programu.

## LIST

Patrz Operacje dyskowe.

## NEW

```
NEW
```

Usuwa znajdujący się w pamięci program.

## REM

```
REM
REM dowolny_ciąg_znaków
REM PODPROGRAM RYSUJACY POLE GRY
```

Umożliwia wstawianie komentarzy, zwykle w celu łatwiejszego zrozumienia kodu programu. Są one ignorowane w trakcie realizacji programu (nie wpływają na wykonywany program). Ponieważ wszystko, co znajdzie się po poleceniu REM traktowane jest jako komentarz (również dwukropek), nie mogą za nim znajdować żadne inne polecenia. Zamiast słowa REM można użyć wykrzyknika lub apostrofu – jeśli znajduje się za innym poleceniem nie trzeba wtedy używać rozdzielającego dwukropka.

```
30 REM PODPROGRAM RYSUJACY POLE GRY
210 A=10:B=2:XC=12!INICJACJA ZMIENNYCH
570 'POZIOM GRY
```

## RENUM (Dostępne tylko z rozszerzeniem na dyskietce).

```
RENUM nowa_linia,stara_linia,skok
RENUM 1000,250,10
RENUM 1000,250
RENUM
```

Przenumerowuje program. Pierwszy parametr to numer nowej linii (*nowa_linia*), drugi parametr to stary numer linii od której ma rozpocząć się renumeracja (*stara_linia*) i trzeci parametr to *skok*, czyli o ile kolejna linia od poprzedniej będzie miała większy numer. Jeśli *skok* zostanie pominięty, to jego wartość domyślnie zostanie ustawiona na 10. Użycie RENUM bez parametrów przenumerowuje cały program ze skokiem 10, nadając nowy numer pierwszej linii programu równy 10. RENUM zmienia też adresy skoku po GOTO, GOSUB itp.
Uwaga. Przed użyciem tego polecenia należy program zapisać na dysku lub kasecie na wypadek niepowodzenia operacji.

## RUN

Patrz Operacje dyskowe.

## STACK

```
PRINT STACK
```

Polecenie STACK podaje liczbę wolnych wpisów dostępnych na stosie. Stos, mogący pomieścić maksymalnie 20 chwilowych wpisów, wykorzystywany jest do chwilowego przechowywania parametrów SOUND i AFTER.

```
1020 PRINT STACK !Drukuje liczbe wolnych wpisow na stosie
1030 IF STACK=0 THEN PRINT "STOS PELNY"
```

## STOP

```
STOP
```

Przerywa działanie programu i wyświetla komunikat w której linii programu to nastąpiło: „BREAK IN [nr_linii]”. Zatrzymanie programu może być pomocne np. przy wyszukiwaniu błędów, gdyż program można edytować, wyświetlać wartości zmiennych (PRINT), po czym ewentualnie wznowić jego działanie poleceniem CONT.

## TROFF (Dostępne tylko z rozszerzeniem na dyskietce).

```
TROFF
```

Wyłącza śledzenie programu (↦TRON).

## TRON (Dostępne tylko z rozszerzeniem na dyskietce).

```
TRON
```

Włącza śledzenie programu. Numer każdej napotkanej linii zostanie wyświetlony na ekranie zanim linia zostanie wykonana. Polecenia TRON można użyć zarówno w trybie bezpośrednim jak i wewnątrz programu (↦TROFF).

## !

(Wykrzyknik) ↦REM

## '

(Apostrof) ↦REM

## AFTER

AFTER(n)GOTO nr_linii
AFTER(n)nr_linii
AFTER(50)300

Przerywa działanie programu po *n* czasie i wykonuje skok do *nr_linii*. Jednostką czasu jest 1/50 sekundy. Jeśli *n*=50, to czas oczekiwania na przerwanie wyniesie 1 sekundę. Maksymalny odliczany czas to 24 godziny. Po RUN, STOP lub END licznik jest kasowany. AFTER współpracuje z poleceniem ↦RESUME i ↦CLEAR STACK.
Uwaga. Jeżeli zostanie podany tylko *nr_linii*, bez GOTO, to polecenie RENUM nie zmieni tego numeru.

```
10 AFTER(75)1000
20 PRINT "*";
30 GOTO 20
40 PRINT :PRINT"KONIEC TESTU"
50 END
1000 PRINT :PRINT"PRZERWANIE!"
1010 RESUME 40
```

## CLEAR STACK

CLEAR STACK

Anuluje polecenie AFTER. W pewnych warunkach może zaistnieć potrzeba zniesienia działania polecenia AFTER. CLEAR STACK to umożliwia.

```
10 X=5
20 AFTER(10)GOTO 70
30 IF X=0 THEN CLEAR STACK
40 PRINT "*";
50 IF PEEK(53279)=7 THEN 40
60 END
70 PRINT " X>0"
80 X=X-1
90 RESUME 20
```

Ten program demonstruje efekt CLEAR STACK. Gdy X osiągnie zero, program wejdzie w „pętlę bez końca". Aby go zatrzymać naciśnij START, SELECT, OPTION lub BREAK.

## FOR...TO...STEP/NEXT

FOR zmienna_początkowa=wartość TO wartość_końcowa STEP skok
FOR X=1 TO 500 STEP 10
FOR X=1 TO 10
FOR X=A TO B STEP C/5

Zestaw instrukcji znajdujących się między FOR a NEXT jest wykonywany dopóki zmienna nie osiągnie pewnej wartości. *Zmienna_początkowa* przy każdym obiegu pętli po napotkaniu NEXT zwiększa swoją wartość o wartość reprezentowaną przez *skok*. Gdy zostanie przekroczona *wartość_końcowa*, pętla zostanie zakończona i program przejdzie do następnego polecenia po NEXT. Parametry pętli mogą mieć wartości zarówno dodatnie jak i ujemne. Jeśli polecenie STEP zostanie pominięte, domyślna wartość *skoku* wyniesie 1.
W przeciwieństwie do Atari BASIC-a nie ma potrzeby (choć można) umieszczania nazwy *zmiennej_początkowej* po NEXT w przypadku pojedynczej pętli. Gdy w programie występują pętle zagnieżdżone (jedna w drugiej), to do ich zamknięcia wystarczy jedno NEXT z wpisanymi *zmiennymi_początkowymi* występującymi w zagnieżdżonych pętlach, oddzielone przecinkami. Należy pamiętać, że pętle zamykamy w odwrotnej kolejności do ich otwierania (patrz pierwszy przykład niżej).

Przy okazji tego przykładu, po jego uruchomieniu, widać błędy procedur matematycznych Microsoft BASIC-a, które są szybkie, ale niezbyt dokładne. Nie ma to jednak wpływu na działanie programu.

W drugim przykładzie, jeśli przed wejściem w pętlę *wartość_początkowa* będzie większa od *wartości_końcowej* to pętla nie zostanie wykonana ani razu.

```
100 FOR X=-5  TO 5  STEP 0.1
110 FOR Y=0 TO 1
120 FOR Z=1 TO 9 STEP 2
130 PRINT "X=";X;" Y=";Y;" Z=";Z
140 NEXT Z,Y,X
150 END
```

```
10 A=2:B=0:C=1
20 FOR X=A TO B STEP C
30 PRINT X
40 NEXT
```

## GOSUB/RETURN

GOSUB nr_linii
GOSUB 2250

GOSUB wywołuje podprogram, którego początkiem jest *nr_linii*, a końcem polecenie RETURN. Po napotkaniu polecenia RETURN następuje powrót z podprogramu do polecenia następnego po GOSUB *nr_linii*.
Uwaga. *Nr_linii* musi być stałą liczbową z przedziału 0 ÷ 63999. Konstrukcja typu GOSUB 490+J zostanie potraktowana jak GOSUB 490, a GOSUB J+490 lub GOSUB J zakończy się komunikatem błędu „UNDEF'D LINE ERROR IN...”.

```
10 GOSUB 62000
20 PRINT "KONIEC"
30 END
62000 PRINT "TO JEST PODPROGRAM"
62010 RETURN
```

## GOTO

GOTO nr_linii
GOTO 800

Normalnie linie programu wykonywane są w kolejności od najniższego numeru. GOTO może ten porządek zmienić. Gdy w programie pojawi się GOTO, nastąpi skok do linii wskazanej przez *nr_linii*.
Uwaga. *Nr_linii* musi być stałą liczbową z przedziału 0 ÷ 63999. Konstrukcja typu GOTO 490+J zostanie potraktowana jak GOTO 490, a GOTO J+490 lub GOTO J zakończy się komunikatem błędu „UNDEF'D LINE ERROR IN...”.

```
10 GOTO 980
20 PRINT "KONIEC"
30 END
980 PRINT "TO JEST LINIA 980"
990 GOTO 20
```

## IF...THEN

IF warunek THEN instrukcje
IF Z=AL+EF THEN A$="STOP"
IF A$="ABC" THEN X=0:Y=17:GOTO 4580
IF R$="N" THEN GOSUB 29000
IF PEEK(53279)<7 THEN 500

Jeżeli *warunek jest* spełniony, czyli wyrażenie między IF a THEN jest prawdziwe, to wykonywane są wszystkie *instrukcje* po THEN do końca linii. Gdy *warunek* nie jest spełniony następuje przejście do kolejnej linii programu. Jeśli po THEN następuje polecenie skoku: GOTO nr_linii, to słowo kluczowe GOTO może zostać pominięte (patrz ostatni przykład). Uwaga. Polecenie skoku musi być stałą liczbową (↦GOTO, ↦GOSUB).

## IF...THEN...ELSE

```
IF warunek THEN instrukcje_1 ELSE instrukcje_2
IF WYNIK>0 THEN PRINT"TWOJA WYGRANA"ELSE PRINT"MOJA WYGRANA"
IF A$="ABC" THEN X=0:Y=17:GOSUB 4580 ELSE X=5:Y=18:GOSUB 4000
```

Jeżeli *warunek jest* spełniony, czyli wyrażenie między IF a THEN jest prawdziwe, to wykonywane są wszystkie instrukcje znajdujące się między THEN i ELSE (*instrukcje_1*), a pomijane po ELSE (*instrukcje_2*).
Gdy *warunek* nie jest spełniony, czyli wyrażenie między IF a THEN nie jest prawdziwe, wówczas wykonywane są polecenia po ELSE do końca linii (*instrukcje_2*), natomiast pomijane między THEN i ELSE (*instrukcje_1*).
Separatorem dla ELSE jest zwykle spacja, ale może też być cudzysłów, jeśli kończy tekst przeznaczony do wyświetlenia na ekranie, i dwukropek.
Uwaga. Jeśli wewnątrz IF...THEN...ELSE występuje polecenie skoku, to adres skoku musi być stałą liczbową (↦GOTO, ↦GOSUB).

## NEXT

↦FOR...TO...STEP/NEXT, ↦RESUME.

## ON ERROR

```
ON ERROR GOTO numer_linii
ON ERROR numer_linii
ON ERROR GOTO 700
ON ERROR 500
```

Gdy nastąpi błąd, wykonywanie programu zostaje przerwane i pojawia się komunikat błędu. ON ERROR przechwytuje błąd i wykonuje skok do wskazanej linii. Aby to zadziałało, polecenie ON ERROR musi być umieszczone przed miejscem, w którym nastąpił błąd. Powrót do normalnej pracy programu następuje po poleceniu RESUME. RUN, STOP i END wyłącza ON ERROR. Numer linii musi być stałą liczbową.

## ON...GOSUB/RETURN

```
ON wyrażenie_liczbowe GOSUB nr_linii_1,nr_linii_2,nr_linii_3...
ON X GOSUB 480,490,500,510,880
```

Polecenie to określa który podprogram zostanie wywołany. Kolejność wyboru zależy od wartości *wyrażenia_liczbowego*. W przykładzie: dla X=1 nastąpi skok do podprogramu zaczynającego się od linii 480, dla X=2 - 490, dla X=3 - 500 itd.
Jeżeli wyrażenie_liczbowe będzie równe 0 lub większe niż liczba odwołań po GOSUB (w przykładzie >5) otrzymamy komunikat o błędzie „FUNCTION CALL ERROR...".
Numer linii musi być stałą liczbową.

## ON...GOTO

```
ON wyrażenie_liczbowe GOTO nr_linii_1,nr_linii_2,nr_linii_3...
ON X GOTO 480,490,500,510,880
```

Polecenie to określa do której linii programu nastąpi skok. Kolejność wyboru zależy od wartości *wyrażenia_liczbowego*. W przykładzie: dla X=1 nastąpi skok do linii 480, dla X=2 - 490, dla X=3 - 500 itd.
Jeżeli *wyrażenie_liczbowe* będzie równe 0 lub większe niż liczba odwołań po GOTO (w przykładzie >5) otrzymamy komunikat o błędzie „FUNCTION CALL ERROR...".
Numer linii musi być stałą liczbową.

## RESUME

Polecenie kończące procedurę wywołaną przez przerwanie programu poleceniem ON...ERROR lub AFTER...GOTO. Efektem wykonania RESUME jest skok:

RESUME
- do polecenia, którego realizacja została przerwana;

RESUME NEXT
- do następnego polecenia, po tym które zostało przerwane;

RESUME nr_linii
- do linii o wskazanym numerze (*nr_linii*).

```
10 AFTER(90)GOTO 500
20 PRINT "LINIA NR 20
30 L=L+1:IF L=200 THEN 200 ELSE 30
100 PRINT "LINIA NR 100
200 STOP
300 PRINT "LINIA NR 300":GOTO 200
500 PRINT "LINIA NR 500
510 RESUME 300
```

A oto co wyświetli ten program przy różnych opcjach RESUME w linii 510:

| RESUME | RESUME NEXT | RESUME 300 |
|---|---|---|
| LINIA NR 20 | LINIA NR 20 | LINIA NR 20 |
| LINIA NR 500 | LINIA NR 500 | LINIA NR 500 |
| | LINIA NR 100 | LINIA NR 300 |
| BREAK IN 200 | | |
| | BREAK IN 200 | BREAK IN 200 |

## RETURN

↦GOSUB/RETURN

## WAIT

WAIT adres,bajt_1
WAIT adres,bajt_1,bajt_2
WAIT 54283,255,110 !CZEKAJ NA VBLANK

Zatrzymuje program do momentu spełnienia określonych warunków. Następuje porównanie (na zasadzie iloczynu logicznego - AND) wartości bajtu w lokalizacji pamięci wskazanej przez *adres* z wartością zapisaną w zmiennej *bajt_1*. Natomiast *bajt_2* zostanie porównany z wartością bajtu spod *adresu* na zasadzie alternatywy wykluczającej (XOR). Gdy *bajt_2* zostanie pominięty, jego domyślna wartość wyniesie zero.

WAIT doskonale nadaje się do zatrzymania programu do momentu wystąpienia VBLANK. Umożliwia to uzyskanie płynnej animacji obiektów ekranowych. Opis przerwania VBLANK znajdziesz m.in. w książce „De Re Atari".

Uwaga. Program zatrzymany poleceniem WAIT można przerwać tylko klawiszem RESET.

```
100 WAIT 644,1
110 PRINT "FIRE
200 POKE 544,100
210 WAIT 544,255
220 PRINT "KONIEC
```

W tym przykładzie WAIT najpierw zostało użyte do oczekiwania na naciśnięcie „fire" w dżojstiku, a potem do zatrzymania programu na ok. 2 sekundy.

## MOVE

```
MOVE adres_źródłowy,adres_docelowy,ile_bajtów
MOVE 1536,1664,48
MOVE PEEK(560)+256*PEEK(561),DL,B
```

Kopiuje blok pamięci o długości określonej przez parametr *ile_bajtów* począwszy od *adresu_źródłowego* do obszaru zaczynającego się od *adresu_docelowego*.
Dane przenoszone są zawsze w taki sposób, aby wszystkie znalazły się w nowym obszarze zanim zostaną zamazane (np. gdy *adres_docelowy* jest mniejszy niż *adres_źródłowy* + *ile_bajtów*). Istotnym zastosowaniem dla MOVE może być obsługa grafiki player-missile.

## OPTION CHR

Rezerwuje obszar pamięci na potrzeby generatora znaków:
OPTION CHR1 - 1024 bajty;
OPTION CHR2 - 512 bajtów;
OPTION CHR0 - zwalnia (usuwa) zarezerwowany obszar.
Adres początku zarezerwowanego obszaru można odczytać odpowiednio poleceniem VARPTR(CHR1) i VARPTR(CHR2).

## OPTION PLM

Rezerwuje obszar pamięci na potrzeby grafiki player-missile (graczy i pocisków):
OPTION PLM1 - rozdzielczość jednoliniowa - 1280 bajtów;
OPTION PLM2 - rozdzielczość dwuliniowa - 640 bajtów;
OPTION PLM0 - zwalnia (usuwa) zarezerwowany obszar.
Polecenie to musi być zawsze poprzedzone instrukcją GRAPHICS, a to dlatego, że komputer najpierw musi znać obszar pamięci zajmowany przez określony tryb graficzny, by mógł zarezerwować miejsce. To, gdzie znajduje się zarezerwowany obszar pamięci wskaże ci polecenie VARPTR(PLM1) i VARPTR(PLM2).
Więcej na temat korzystania z grafiki player-missile znajdziesz m.in. w podręczniku „De Re Atari".

## OPTION RESERVE

```
OPTION RESERVE ile_bajtów
OPTION RESERVE 48
```

Rezerwuje obszar pamięci na własne procedury użytkownika np. programy w języku maszynowym (w przykładzie jest to 48 bajtów). Adres początku zarezerwowanego obszaru można odczytać poleceniem VARPTR(RESERVE).

## PEEK

```
PEEK(adres)
A=PEEK(1536)
A=PEEK(X)
```

Odczytuje wartość bajtu (0 ÷ 255) spod *adresu* pamięci (0 ÷ 65535).

## POKE

```
POKE adres,bajt
POKE 1536,255
POKE X,Y
```

Zapisuje *bajt* (0 ÷ 255) pod wskazany *adres* pamięci (0 ÷ 65535).

# VARPTR

VARPTR(nazwa_zmiennej)
Umożliwia właściwe umieszczenie w pamięci danych (procedury maszynowe, PMG, generator znaków i in.) oraz modyfikacje zmiennych, nawet nie znając ich fizycznych adresów.

VARPTR(PLM1)
VARPTR(PLM2)
Zwraca adres początku obszaru pamięci zarezerwowanego dla grafiki graczy i pocisków (PMG), a dokładniej początku pamięci zajmowanej przez pociski. PLM1 - rozdzielczość jednoliniowa, PLM2 - dwuliniowa. Do rezerwacji obszaru służy polecenie ↦OPTION PLM. Więcej na temat grafiki graczy i pocisków znajdziesz np. w podręczniku „De Re Atari".

VARPTR(CHR1)
VARPTR(CHR2)
Zwraca adres początku obszaru pamięci zarezerwowanego dla nowego generatora znaków. CHR1 dla 1024-bajtowego obszaru, a CHR2 dla 512-bajtowego. Do rezerwacji obszaru służy polecenie ↦OPTION CHR.

VARPTR(RESERVE)
Zwraca początkowy adres zarezerwowanego przez użytkownika obszaru pamięci. Uwaga. Ten obszar jest ruchomy, ulega przesunięciu wraz ze zmianą objętości programu. Do rezerwacji obszaru służy polecenie ↦OPTION RESERVE.

VARPTR(zmienna_tekstowa)
VARPTR(A$)
Zwraca początkowy adres w tablicy symboli. Pierwszy bajt określa długość zmiennej. Aby znaleźć ciąg znaków trzeba odczytać wartość drugiego (młodszego) i trzeciego (starszego) bajtu.

```
10 A$="ABCDE"
20 X=VARPTR(A$)
30 D=PEEK(X):Y=PEEK(X+1)+256*PEEK(X+2)
40 PRINT D:PRINT X:PRINT Y:PRINT:PRINT A$
50 POKE Y+1,33
60 PRINT A$
```

Oto efekt działania programu (niektóre uzyskane wartości mogą być inne - zależy to od lokalizacji programu w pamięci):

```
 5
 11238
 31761

ABCDE
A!CDE
```

VARPTR(zmienna_liczbowa)
VARPTR(LPX)
Zwraca 2-bajtowy adres początkowy zmiennej w pamięci (młodszy - starszy bajt).

```
10 A=7
20 X=VARPTR(A)
30 PRINT X:PRINT A
40 POKE X,129
50 PRINT A
```

Oto efekt działania programu (niektóre uzyskane wartości mogą być inne - zależy to od lokalizacji programu w pamięci):

```
 11185
 7
 1.75
```

## CLS

CLS
CLS kolor_tła
CLS 160

Czyści ekran w trybie tekstowym 0 i ustala (opcjonalnie) *kolor_tła*. Wartość tego parametru obliczana jest według wzoru: *kolor_tła* = *barwa**16+*jaskrawość* - patrz niżej polecenie SETCOLOR.

## COLOR

COLOR wyrażenie_liczbowe
COLOR 1
COLOR A

Wybiera rejestr koloru, który będzie używany do rysowania na ekranie. Polecenie COLOR nie określa zawartości rejestru koloru, do tego służy polecenie SETCOLOR. Różne tryby graficzne używają różnych rejestrów koloru. Podstawowe informacje znajdują się w tabeli.

| Tryb | SETCOLOR (rejestr koloru) | COLOR | Uwagi |
|---|---|---|---|
| 0 (tryb tekstowy) | 5<br>6<br>8 | -<br>-<br>- | tekst<br>tło<br>ramka |
| 1, 2 (tryby tekstowe) | 4<br>5<br>6<br>7<br>8 | -<br>-<br>-<br>-<br>- | ATASCII 32÷95<br>ATASCII 0÷31 i 96÷127<br>ATASCII 160÷223<br>ATASCII 128÷159 i 224÷255<br>tło |
| 3, 5, 7, 15 (tryby 4-kolorowe) | 4<br>5<br>6<br>8 | 1<br>2<br>3<br>0 | <br><br><br>tło |
| 4, 6, 14 (tryby 2-kolorowe) | 4<br>8 | 1<br>0 | <br>tło |
| 8 (tryb 1½ koloru) | 5<br>6<br>8 | 1<br>0<br>- | <br>tło<br>ramka |
| 9 (tryb GTIA) | 8 | 0÷15 | COLOR wybiera jeden z 16 dostępnych poziomów jaskrawości. |
| 10 (tryb GTIA) | 0÷7<br>8 | 0÷7<br>8 | kolor punktu<br>tło |
| 11 (tryb GTIA) | 8 | 0÷15 | 16 kolorów o tej samej jaskrawości |
| 12, 13 (tryby tekstowe) | 4÷8 | - | pięciokolorowe tryby tekstowe |

Uwaga. Każdorazowo po wykonaniu polecenia GRAPHICS w rejestrach kolorów umieszczane są wartości standardowe (nie dotyczy to rejestrów dla grafiki player-missile). Aby ustawić pożądane przez użytkownika wartości należy użyć polecenia SETCOLOR.

## FILL

FILL x1,y1 TO x2,y2
FILL 10,25 TO 8,8

Wypełnia ekran kolorem określonym przez polecenie COLOR (patrz też SETCOLOR). Wypełnianie następuje od lewej do prawej strony ekranu i trwa do momentu napotkania prawego skraju ekranu lub punktu, który nie jest tłem. Początkowy punkt od którego rozpoczyna się wypełnianie określają parametry *x1*,*y1*, a końcowy - *x2*,*y2*.

```
10 GRAPHICS 7
20 COLOR 1:PLOT 10,10 TO 50,10 TO 50,25
30 COLOR 2:FILL 10,10 TO 18,25
50 COLOR 1:PLOT 20,40 TO 50,40 TO 50,70 TO 20,70 TO 20,40
60 COLOR 3:FILL 21,40 TO 21,69
```

## GRAPHICS

GRAPHICS wyrażenie_liczbowe
GRAPHICS 0
GRAPHICS 18
GRAPHICS 7+16+32
GRAPHICS TG

Wybiera jeden z 16 trybów graficznych. W tabeli znajdują się podstawowe parametry dostępnych trybów (bez okienka tekstowego).

| Tryb | Rodzaj | Rozdzielczość (pozioma x pionowa) | Kolory - poziomy jasności | Wymagana pamięć (bajty) ekran - ekran z DL |
|---|---|---|---|---|
| 0 | tekstowy | 40 x 24 | 1½ - 8 | 960 - 992 |
| 1 | tekstowy | 20 x 24 | 5 - 8 | 640 - 672 |
| 2 | tekstowy | 20 x 12 | 5 - 8 | 400 - 420 |
| 3 | graficzny | 40 x 24 | 4 - 8 | 400 - 432 |
| 4 | graficzny | 80 x 48 | 2 - 8 | 640 - 696 |
| 5 | graficzny | 80 x 48 | 4 - 8 | 1120 - 1176 |
| 6 | graficzny | 160 x 96 | 2 - 8 | 2080 - 2184 |
| 7 | graficzny | 160 x 96 | 4 - 8 | 4000 - 4200 |
| 8 | graficzny | 320 x 192 | 1½ - 8 | 7856 - 8138 |
| 9 | graficzny/GTIA | 80 x 192 | 1 - 16 | 7856 - 8138 |
| 10 | graficzny/GTIA | 80 x 192 | 9 - 8 | 7856 - 8138 |
| 11 | graficzny/GTIA | 80 x 192 | 16 - 1 | 7856 - 8138 |
| 12 | tekstowy | 40 x 24 | 5 - 8 | 1120 - 1152 |
| 13 | tekstowy | 40 x 12 | 5 - 8 | 640 - 660 |
| 14 | graficzny | 160 x 192 | 2 - 8 | 4000 - 4296 |
| 15 | graficzny | 160 x 192 | 4 - 8 | 7856 - 8138 |

- Tryby 1 do 8 i 12 do 15 standardowo zawierają na dole okienko tekstowe. Jeśli nie jest ono potrzebne, należy do numeru trybu dodać 16.
- Dodanie liczby 32 do numeru trybu otwiera go bez kasowania zawartości ekranu.
- Drukowanie tekstu w okienku tekstowym oraz w trybie 0 wymaga polecenia PRINT, natomiast w trybie 1, 2, 12 i 13 PRINT#6.
- Tryby 12 i 13 są specjalnymi trybami kolorowymi. Aby uzyskać czytelny wygląd tekstu należy odpowiednio przedefiniować generator znaków.
- Polecenia PRINT#6 można też używać do rysowania w trybach graficznych (3 do 11 oraz 14 i 15).

- W trybach 0 i 8 określenie koloru „1½” oznacza, że można ustawić dowolny kolor tła i tylko poziom jasności dla tekstu lub rysunku (kolor będzie taki sam jak tła).
- Adres początku pamięci ekranu zawierają komórki 88, 89, a display list 560, 561 (młodszy i starszy bajt). Więcej na temat trybów graficznych znajdziesz m.in. w „De Re Atari”.

## PLOT/PLOT...TO

```
PLOT x1,y1
PLOT x1,y1 TO x2,y2
PLOT 17,9
PLOT 0,8 TO 20,8 TO A,B/2
```

PLOT x1,y1 rysuje punkt na ekranie graficznym. Współrzędna pozioma na ekranie to *x1*, a współrzędna pionowa to *y1*. Aby uzyskać linię, należy podać po poleceniu PLOT współrzędne punktu od którego rozpocznie się rysowanie linii (*x1*,*y1*), a następnie po słowie kluczowym TO współrzędne końca linii (*x2*,*y2*). W celu uzyskania bardziej złożonych rysunków można wielokrotnie powtórzyć polecenie TO x,y.

Należy pamiętać o wcześniejszym wybraniu poleceniem COLOR rejestru koloru, którego będziemy używać do rysowania.

```
10 GRAPHICS 7:COLOR 1
20 PLOT 10,10 TO 50,10 TO 50,25 TO 10,25 TO 10,11
```

Ten program rysuje prostokąt.

## SETCOLOR

```
SETCOLOR rejestr_koloru,barwa,jaskrawość
SETCOLOR 6,12,4
SETCOLOR R,C*2,X
```

Ustala w wybranym *rejestrze_koloru barwę* i jej *jaskrawość* (patrz też wyżej COLOR).

***Rejestry_koloru*** o wartości:
0 ÷ 3 przeznaczone są odpowiednio dla graczy i pocisków (grafika player-missile);
4 ÷ 7 określają pole gry (playfield);
8 – określa kolor tła.
Różne tryby graficzne w różnym zakresie korzystają z tych rejestrów – patrz tabela przy poleceniu COLOR.

***Barwa*** stanowi kod koloru i przybiera wartości 0 ÷ 15. Wykaz kolorów i ich kodów zawiera tabela. Należy pamiętać, że realnie uzyskiwane kolory mogą się nieco różnić, zależnie od egzemplarza komputera i monitora.

| Kod | Kolor | Kod | Kolor |
|---|---|---|---|
| 0 | szary | 8 | niebieski |
| 1 | jasnopomarańczowy | 9 | jasnoniebieski |
| 2 | pomarańczowy | 10 | turkusowy |
| 3 | czerwonopomarańczowy | 11 | niebieskozielony |
| 4 | różowy | 12 | zielony |
| 5 | purpurowy | 13 | żółtozielony |
| 6 | fioletowoniebieski | 14 | złocisty |
| 7 | lazurowy | 15 | jasnopomarańczowy |

***Jaskrawość*** to parzyste liczby z zakresu 0 ÷ 14, przy czym 0 to jaskrawość bliska czerni, a 14 bliska bieli, czyli im większa wartość, tym jaśniejszy kolor.

## SOUND

SOUND głos,częstotliwość,zniekształcenia,głośność,trwanie
SOUND 0,114,14,6,220
SOUND 1,230,2,10

Steruje generatorami dźwięku:

- ***głos*** to numer generatora dźwięku, dostępne są cztery o numerach 0 ÷ 3;
- ***częstotliwość*** generowanego dźwięku jest liczbą z przedziału 0 ÷ 255. Im większa wartość, tym niższa częstotliwość (patrz tabela niżej);
- ***zniekształcenia*** to liczba parzysta z przedziału 0 ÷ 14. Wpisując tu wartość 10 lub 14 uzyskiwany jest czysty dźwięk, inne wartości wprowadzają różnego rodzaju zniekształcenia. Wpisanie wartości nieparzystej wyłącza generator;
- ***głośność*** to parametr z przedziału 0 ÷ 15. 1 to dźwięk ledwo słyszalny, 15 to maksymalna głośność, a 0 to dźwięk całkowicie wyciszony. Wartość 8 uznać można za normalną. Suma tego parametru dla wszystkich równocześnie używanych głosów nie powinna przekroczyć 32, inaczej wywoła to nieprzyjemny „brum";
- ***trwanie*** określa jak długo będzie trwał dźwięk. Jednostką czasu jest 1/50 sekundy, inaczej mówiąc, wpisanie tu liczby 50 wywoła dźwięk przez jedną sekundę. Parametr ten jest nieobowiązkowy. W przypadku jego pominięcia dźwięk będzie generowany dopóki nie zostanie ponownie użyte polecenie SOUND dla tego głosu. Dźwięk jest wyłączany również po zakończeniu programu, naciśnięciu BREAK oraz RETURN - jeśli kończy wprowadzanie danych przez polecenie INPUT.

| Tony niskie | | Tony średnie | | Tony wysokie | |
|---|---|---|---|---|---|
| Dźwięk | Okres (p) | Dźwięk | Okres (p) | Dźwięk | Okres (p) |
| C | 243 | C | 121 | C | 60 |
| C# | 230 | C# | 114 | C# | 57 |
| D | 217 | D | 108 | D | 53 |
| D# | 204 | D# | 102 | D# | 50 |
| E | 193 | E | 96 | E | 47 |
| F | 182 | F | 91 | F | 45 |
| F# | 173 | F# | 85 | F# | 42 |
| G | 162 | G | 81 | G | 40 |
| G# | 153 | G# | 76 | G# | 37 |
| A | 144 | A | 72 | A | 35 |
| A# | 136 | A# | 68 | A# | 33 |
| H | 128 | H | 64 | H | 31 |
| | | | | C | 29 |

*Źródło: Atari Basic. „Język programowania i obsługa mikrokomputera Atari", NOT 1987*

Przykład użycia SOUND w programie (na podstawie programu „Night Launch" zawartego w „Reference Manual ATARI MICROSOFT BASIC II"). Efekt dźwiękowy jest podobny do startującej rakiety w kosmos.

```
10 GRAPHICS 2+16
20 PA=1000:GOSUB 300
30 PRINT#6,AT(1,3);"NA POLU STARTOWYM"
40 PA=2000:GOSUB 300
50 GRAPHICS 0
60 POKE 752,1
70 SETCOLOR 6,0,0
80 PA=200
90 FOR T=1 TO 24:PRINT "":NEXT
100 PRINT TAB(11);CHR$(8);CHR$(10)
110 PRINT TAB(11);CHR$(22);CHR$(2)
120 PRINT TAB(11);CHR$(22);CHR$(2)
130 PRINT TAB(11);CHR$(13);CHR$(13)
140 PRINT TAB(11);CHR$(6);CHR$(7)
150 FOR VOL=15 TO 0 STEP -1
160 SOUND 2,77,8,VOL
170 PRINT CHR$(155)!RUCH RAKIETY
180 GOSUB 300
190 NEXT VOL
200 PRINT "KONIEC"
210 PA=2000:GOSUB 300
220 GRAPHICS 0
230 END
299 !PETLA OPOZNIAJACA (PODPROGRAM)
300 FOR DELAY=1 TO PA:NEXT:RETURN
```

# Polecenia tekstowe, liczbowe i formatujące

## CLEAR

CLEAR

Zeruje wszystkie tekstowe i liczbowe zmienne, w tym wielowymiarowe. Jeśli po użyciu CLEAR potrzebna jest tablica, wskazane jest zadeklarowanie jej poleceniem DIM.

## COMMON

COMMON zmienna

COMMON A,B,C,X$

Zachowuje wskazane zmienne i ich wartości w różnych programach. Ułatwia to wymianę danych między programami lub tworzenie programów składających się z dwóch lub więcej części.

COMMON ALL

Zachowuje wszystkie zmienne i ich wartości.

## DATA

DATA informacja,informacja,informacja...

Polecenie DATA przechowuje dane (*informacje*) wewnątrz programu. Do ich odczytania służy polecenie READ. Można umieszczać dowolne ciągi tekstowe i liczbowe oddzielając je przecinkiem. Zobacz też RESTORE.

Uwagi.

- Jeżeli w ciągu znaków musi być przecinek, takie wyrażenie trzeba ująć w cudzysłów (normalnie przecinek służy do oddzielania danych).
- W celu uzyskania tekstu w cudzysłowie należy go ująć potrójnym cudzysłowem (patrz niżej przykład programu).

- Pobierając dane poleceniem READ należy pamiętać o przypisaniu ich właściwym zmiennym, tzn. ciągów liczbowych - zmiennym liczbowym i ciągów tekstowych - zmiennym tekstowym.
- Polecenie READ pobiera dane kolejno, od pierwszego elementu w linii DATA. Próba pobrania większej ilości danych niż zapisano w DATA wywoła błąd.

```
10 READ X
20 FOR A=1 TO X
30 READ A$:PRINT A$
40 NEXT:END
50 DATA 4,FRED,"JOLA, EWA",STEFAN,""""ABC""""
```

Efekt uruchomienia programu będzie następujący:

```
FRED
JOLA, EWA
STEFAN
"ABC"
```

## INPUT

```
INPUT#iocb,zmienna
INPUT#iocb,zmienna,zmienna,zmienna...
INPUT"tekst";zmienna
INPUT"tekst";zmienna,zmienna,zmienna...
INPUT#1,A$
INPUT"PODAJ NAZWISKO I WIEK";NAZWISKO$,WIEK
INPUT X(2,7)
INPUT NUM
```

INPUT umożliwia wprowadzanie informacji do programu. Wpisywane znaki są umieszczane w odpowiednich zmiennych. Jeżeli pominiemy blok *#iocb*, to dane zostaną pobrane z klawiatury.

- Możliwe jest wprowadzanie danych do kilku zmiennych.
- Przejście do następnej zmiennej następuje po wpisaniu przecinka, dlatego przecinek nie może znaleźć się wewnątrz wpisywanego tekstu.
- Zakończenie wpisywania danych następuje po naciśnięciu RETURN.
- Zmienna może być tablicą.
- INPUT automatycznie wstawia znak zapytania przed wprowadzanym tekstem, o ile nie został umieszczony dodatkowy *tekst*.
- INPUT tymczasowo zatrzymuje program, aż z klawiatury zostaną wprowadzone wszystkie wymagane dane.
- INPUT nie może być użyte w trybie bezpośrednim.
- Tekst w cudzysłowie (wraz ze średnikiem na końcu) jest opcjonalny.
- Nie można pobrać danych z klawiatury w przypadku określenia bloku *#iocb*.
- W przypadku wpisania numeru *#iocb* nie można wpisywać *tekstu*; dane są pobierane automatycznie przez wskazany blok.

## INPUT...AT

```
INPUT#iocb,AT(x,y)zmienna
INPUT#1,AT(144,16)J
INPUT AT(0,0)A$,B$
```

To polecenie od ↦INPUT odróżnia brak możliwości dodania tekstu informacyjnego w zamian za możliwość pozycjonowania - parametry *x* i *y* w nawiasie po słowie kluczowym TO.

Jeżeli specyfikacja bloku *#iocb* zostanie pominięta, to wprowadzanie danych nastąpi od pozycji na ekranie określonej przez *x* (pozioma pozycja ekranu) i *y* (pionowa pozycja ekranu). Natomiast gdy kanał *#iocb* został otwarty do współpracy ze stacją dysków, parametr *x* określa sektor, a parametr *y* bajt na dyskietce.

## LINE INPUT

```
LINE INPUT"tekst";zmienna_tekstowa
LINE INPUT X$
LINE INPUT"PODAJ MODEL";MOD$
```

Umożliwia wprowadzenie danych tylko do jednej zmiennej tekstowej, ale z możliwością użycia dowolnego ciągu znaków, również przecinka. *Tekst* po LINE INPUT jest opcjonalny. Jeśli zostanie pominięty, w miejscu wprowadzania danych zostanie wyświetlony znak „?”.

```
LINE INPUT#iocb,AT(sektor,bajt)zmienna
LINE INPUT#5,AT(147,64)A$
LINE INPUT#5,AT(147,64)D
```

Po otwarciu odpowiedniego bloku IOCB, można wykorzystać LINE INPUT do pobierania danych z urządzenia zewnętrznego. Porównaj z poleceniem INPUT.

## PRINT

```
PRINT
PRINT"tekst"
PRINT zmienna_tekstowa
PRINT liczba
PRINT zmienna_liczbowa
PRINT"ATARI ";X$,Z
PRINT"ATARI";800;M$,Z+1
PRINT#6,"TEST"
```

Umieszcza na ekranie ciąg znaków. Użyte samodzielnie (bez parametrów) przesuwa kursor w dół do następnej linii pozostawiając na poprzedniej pozycji pustą linię. W celu umieszczenia danych w GRAPHICS 1 i 2 lub trybach graficznych należy użyć polecenia PRINT#6... jak w ostatnim przykładzie. Poszczególne wyrażenia w poleceniu PRINT można rozdzielać:

- przecinkiem („,”), wówczas wyrażenie po nim zostanie umieszczone na ekranie od nowej pozycji tabulacji;
- średnikiem („;”), wówczas wyrażenie po nim zostanie umieszczone na ekranie bez odstępu od poprzedniego.

Uwagi.

- Liczby dodatnie i zero, wyświetlane na ekranie, zawsze poprzedzone są spacją, natomiast ujemne znakiem „-” (minus).
- Nie jest konieczne zamykanie cudzysłowu, gdy tekst do wydrukowania jest ostatnim elementem linii np. »190 PRINT"DOWOLNY TEKST«. Jeśli program zawiera dużo tekstów do wyświetlenia, oszczędza się w ten sposób pamięć.
- Zamiast słowa „PRINT” można użyć znaku zapytania („?”).

```
100 X$="130 XE":M$="XL":Z=1
110 PRINT"ATARI ";X$,Z
120 PRINT"ATARI";800;M$,Z+1
```

Efektem uruchomienia powyższego programu będzie następujący wydruk na ekranie:

```
ATARI 130 XE   1
ATARI 800 XL   2
```

## PRINT...AT

```
PRINT#iocb,AT(a,b)wyrażenie
PRINT#6,AT(0,2)"ATARI",X,A$
```

Wysyła dane przez wskazany blok IOCB. Parametry *a* i *b* określają położenie wysyłanych danych. W przykładzie dane wysyłane są na ekran - pozycja pozioma równa 0 i pionowa równa 2. Gdy blok IOCB zostanie otwarty do współpracy ze stacją dysków parametr *a* oznacza numer sektora, a parametr *b* numer bajtu w tym sektorze.

Pominięcie *#iocb* spowoduje wydruk przez kanał IOCB#0, czyli na ekranie w trybie GRAPHICS 0. Więcej przy poleceniu PRINT, wyżej.

```
10 GRAPHICS 0:L=64:A$="XE
20 PRINT AT(2,12)"ATARI";65;A$,L;"kB RAM
```

Wynikiem działania tego programu będzie:

```
ATARI 65 XE    64 kB RAM
```

## PRINT SPC

```
PRINT SPC(n)
PRINT SPC(3)
PRINT SPC(AX)
PRINT AT(2,2)SPC(7)
PRINT AT(2,1)"CZTERY SPACJE";SPC(4)
```

Drukuje spacje. Ilość wydrukowanych spacji określa zmienna *n*.

## PRINT TAB

```
PRINT TAB(n)
PRINT TAB(20)
PRINT AT(2,10)TAB(8)
```

Przesuwa kursor drukując tyle spacji, aby ten znalazł się na pozycji poziomej zgodnej z wartością parametru *n*. Odliczanie zawsze zaczyna się od lewego skraju ekranu bez względu na ustawioną pozycję lewego marginesu (komórka 82 pamięci).

```
1 PRINT AT(0,2);STRING$(20,"*")
2 PRINT AT(0,3)"01234567890123456789
3 PRINT AT(5,2)TAB(10);"TAB 10
```

## PRINT USING (Dostępne tylko z rozszerzeniem na dyskietce).

Umożliwia formatowanie danych wyjściowych na wiele sposobów, m.in.:

- zmienna liczbowa może być precyzyjnie umieszczona tam gdzie chcesz;
- można wstawiać przecinek (kropkę dziesiętną) w drukowanych kwotach;
- można umieścić znak dolara („$”) przed pierwszą cyfrą kwoty w dolarach;
- kwoty mogą być z lewej strony wypełnione gwiazdkami (***$99.98) dla ochrony;
- można wymusić wyświetlanie liczb w formacie wykładniczym albo podwójnej precyzji;
- można drukować znak plus („+”) przed liczbami dodatnimi i minus („-”) przed ujemnymi.

**PRINT USING #**

Wypełnia spacjami wolne pola z lewej strony (gdy liczba zajmuje mniej pól niż przewidzianych w formatowaniu), przesuwając całą liczbę do prawej krawędzi formatowania. Gdy liczba posiada więcej znaków niż przewiduje pole formatowania, drukowana jest ze znakiem „%” na początku.

```
10 X=485:PRINT USING"######";X
20 X=100098:PRINT USING"###";X
```

Wynikiem działania tego programu będzie:

```
   485
%100098
```

**PRINT USING .**

Kropka umieszczona wewnątrz ciągu znaków „#” określa punkt dziesiętny. Niewykorzystane pola „#” po kropce uzupełniane są zerami. Większa liczba liczba cyfr niż pól powoduje automatyczne zaokrąglenie wartości.

```
10 X=485.9
15 PRINT USING"###.##";X
20 X=1.4142135
25 PRINT USING"###.##";X
```

Wynikiem działania tego programu będzie:

```
485.90
  1.41
```

**PRINT USING ,**

Drukuje przecinek co trzy cyfry od lewej strony.

```
5 DEFDBL X
10 X=2933604.53
15 PRINT USING"########,.###";X
20 Y=1.4142135
25 PRINT USING"######,##.##";Y
30 Z=29755.88
35 PRINT USING"######,##.#";Z
```

Wynikiem działania tego programu będzie:

```
2,933,604.530
        1.41
   29,755.9
```

**PRINT USING ***

Drukuje gwiazdkę na pierwszym niewykorzystanym polu z lewej strony liczby. Użycie dwóch gwiazdek powoduje zapełnienie gwiazdkami wszystkich wolnych pól z lewej strony liczby.

```
10 X=1.41
15 PRINT USING"*#####.##";X
20 PRINT USING"**####.##";X
```

Wynikiem działania tego programu będzie:

```
*    1.41
*****1.41
```

**PRINT USING $**

Drukuje znak dolara („$") z lewej strony liczby. Podwójny znak dolara drukuje go bezpośrednio przy liczbie, z lewej strony.

```
10 X=24.98
15 PRINT USING"$#####.##";X
20 PRINT USING"$$####.##";X
```

Wynikiem działania tego programu będzie:

```
$   24.98
   $24.98
```

**PRINT USING **$**

Połączenie ↦PRINT USING ** z ↦PRINT USING $.

```
10 X=24.98
20 PRINT USING"**$###.##";X
```

Wynikiem działania tego programu będzie:

```
***$24.98
```

**PRINT USING ^^^^**

Cztery znaki potęgowania oznaczają, że liczba ma zostać wydrukowana w postaci wykładniczej. Znaki „^^^^" muszą być poprzedzone co najmniej jednym znakiem „#".

```
10 X=1200
20 PRINT USING"###^^^^";X
30 PRINT USING"####^^^^";X
```

Wynikiem działania tego programu będzie:

```
 12E+02
 120E+01
```

**PRINT USING +**

Drukuje znak plus dla liczb dodatnich i minus dla liczb ujemnych, odpowiednio przed pierwszą cyfrą lub za ostatnią.

```
10 X=725
20 PRINT USING"+####.##";X
30 PRINT USING"####.##+";X
40 X=-725
50 PRINT USING"+####.##";X
60 PRINT USING"####.##+";X
```

Wynikiem działania tego programu będzie:

```
 +725.00
 725.00+
 -725.00
 725.00-
```

**PRINT USING -**

Dla liczb ujemnych znak minus będzie widoczny po prawej stronie, a dla liczb dodatnich znak „+” nie będzie wyświetlany.

```
100 A=-311:B=311
110 PRINT USING "###-";A
120 PRINT USING "###-";B
```

Wynikiem działania tego programu będzie:

```
311-
311
```

**PRINT USING !**

Wykrzyknik pokazuje pierwszy znak zmiennej tekstowej.

```
100 A$="PRZYGODA"
110 PRINT USING"!";A$
```

Wynikiem działania tego programu będzie:

```
P
```

**PRINT USING %␣␣␣%**

Spacje pomiędzy znakami „procent”.
Pokazuje początkowe znaki zmiennej tekstowej. Ilość wyświetlanych znaków równa jest liczbie spacji (symbolizuje je znaczek „␣”) łącznie ze znakami „%”.

```
10 A$="DRUKARNIA"
20 PRINT USING"%  %";A$
```

Wynikiem działania tego programu będzie:

```
DRUK
```

## READ

```
READ wyrażenie
READ wyrażenie,wyrażenie,wyrażenie...
READ A,B,TXT$
```

Służy do pobierania danych. Więcej przy poleceniu DATA.

## RESTORE

RESTORE
RESTORE numer_linii
RESTORE 1150

Umożliwia ponowne pobranie danych z linii ↦DATA, czyli takich danych, które już wcześniej zostały pobrane. Jeżeli nie zostanie wskazany numer linii od którego ma nastąpić pobieranie, to dane będą pobrane z pierwszej (o najniższym numerze) linii DATA występującej w programie. *Nr_linii* musi być stałą liczbową.

## ?

↦PRINT (wyżej).

# Funkcje matematyczne

Uwaga. Funkcje trygonometryczne obliczane są w radianach.

## ABS

ABS(wyrażenie_liczbowe)
A=ABS(-12)
PRINT ABS(X)

Zwraca wartość bezwzględną liczby, np. ABS(-12)=12, ABS(6)=6, ABS(-3.33)=3.33, czyli wynikiem tej funkcji jest zawsze liczba dodatnia.

## ATN

ATN(wyrażenie_liczbowe)
A=ATN(.75)
PRINT ATN(X)

Zwraca arcus tangens z *wyrażenia_liczbowego*.

## COS

COS(wyrażenie_liczbowe)
A=COS(.125)
PRINT COS(X)

Oblicza cosinus z *wyrażenia_liczbowego*.

## EXP

EXP(wyrażenie_liczbowe)
A=EXP(3)
PRINT EXP(X)

Oblicza liczbę e (liczba Eulera) podniesioną do potęgi reprezentowanej przez *wyrażenie_liczbowe.*

## INT

INT(wyrażenie_liczbowe)
A=INT(9.9)
A=INT(-12.1)
PRINT INT(X)

Zwraca liczbę całkowitą z *wyrażenia_liczbowego*. INT zawsze zaokrągla w dół do liczby całkowitej. Dlatego INT(9.9)=9, a INT(-12.1)=-13.

## LOG

```
LOG(wyrażenie_liczbowe)
A=LOG(8)
PRINT LOG(X)
```

Zwraca logarytm naturalny z *wyrażenia_liczbowego*, którego wartość musi być większa od zera. Próba wywołania LOG(0) zakończy się komunikatem błędu: „FUNCTION CALL ERROR”.

## RND

Generuje liczbę losową:

```
RND
RND(0)
```

– z przedziału (0,1) – czyli wygenerowana liczba będzie zawsze większa od zera i mniejsza od 1;

```
RND(n)
RND(10)
RND(LC)
```

– z przedziału od 1 do *n* (włącznie z 1 i *n*), przy czym *n* musi być liczbą naturalną większą lub równą 2. W drugim przykładzie zostanie wygenerowana liczba naturalna z przedziału <1,10>.
Podanie *n* jako liczby ujemnej lub z ułamkiem nie spowoduje błędu, ale zostanie ona potraktowana jak dodatnia liczba całkowita.

Z RND współpracuje polecenie ↦RANDOMIZE.

## SGN

```
SGN(wyrażenie_liczbowe)
A=SGN(8)
A=SGN(-100)
PRINT SGN(X)
```

Zwraca znak *wyrażenia_liczbowego*. Dla liczby dodatniej wynikiem będzie +1 (przykład pierwszy), dla ujemnej wynikiem będzie -1 (przykład drugi), a dla zera będzie 0.

## SIN

```
A=SIN(wyrażenie_liczbowe)
A=SIN(1)
PRINT SIN(X)
```

Zwraca sinus z *wyrażenia_liczbowego*.

## SQR

```
A=SQR(wyrażenie_liczbowe)
A=SQR(144)
PRINT SQR(X)
```

Oblicza pierwiastek kwadratowy z *wyrażenia_liczbowego*, będącego dowolną liczbą dodatnią (lub zerem). Próba użycia liczby ujemnej zakończy się komunikatem błędu: „FUNCTION CALL ERROR”.

## TAN

```
TAN(wyrażenie_liczbowe)
A=TAN(.18)
PRINT=TAN(X)
```

Zwraca tangens z *wyrażenia_liczbowego*.

## +

Operator konkatenacji (łączenia dwóch tekstów w jeden).

```
10 A1$="Nad"
20 A2$="zwyczajny
30 PRINT A1$:PRINT A2$:PRINT
40 X$=A1$+A2$
50 PRINT X$
```

Po uruchomieniu programu zobaczymy:
Nad
zwyczajny

Nadzwyczajny

## ASC

ASC(wyrażenie_tekstowe)
PRINT ASC(TXT$)
X=ASC("ABCDE")
Zamienia pierwszy znak *wyrażenia_tekstowego* na odpowiadający mu kod dziesiętny. W ostatnim przykładzie będzie to liczba 65, gdyż taki kod ma litera „A”. ASC jest odwrotnością polecenia CHR$. Patrz również Aneks III: kody ATASCII.

## CHR$

CHR$(kod_znaku)
PRINT CHR$(90)
Zamienia liczbę z zakresu 0 ÷ 255 na odpowiadający jej znak ATASCII. W przykładzie będzie to „Z”. CHR$ jest odwrotnością polecenia ASC. Patrz również Aneks III: kody ATASCII.

## INKEY$

INKEY$
X$=INKEY$
Odczytuje znak ostatnio naciśniętego klawisza. Jeżeli żaden klawisz nie został naciśnięty, zwraca pusty ciąg - w przykładzie X$="", inaczej: LEN(X$)=0. Nie zatrzymuje programu. INKEY$ nie rozpoznaje spacji.

## INSTR

INSTR(indeks,wyrażenie_tekstowe_1,wyrażenie_tekstowe_2)
P=INSTR(5,A$,B$)
I=INSTR(A$,"ABC")
Odszukuje w dłuższym *wyrażeniu_tekstowym_1* ciąg znaków zawarty w krótszym *wyrażeniu_tekstowym_2*. Gdy go odnajdzie, zwraca pozycję pierwszego znaku *wyrażenia_tekstowego_2* w tym pierwszym. Wyszukiwanie rozpoczyna się od pozycji *indeks*. Jeżeli *indeks* zostanie pominięty, wyszukiwanie rozpocznie się od pierwszego znaku *wyrażenia_tekstowego_1*. Wynikiem działania funkcji będzie 0, gdy ciąg nie zostanie odnaleziony.

```
10 A$="1234567890ABCDEFGHXE"
20 B$="ABC"
30 PRINT INSTR(A$,B$)
40 PRINT INSTR(10,A$,"XE")
50 PRINT INSTR(A$,"65XE")
```

Po uruchomieniu tego programu zobaczymy:
```
 11
 19
 0
```

## LEFT$

```
LEFT$(wyrażenie_tekstowe,n)
PRINT LEFT$(A$,9)
PRINT LEFT$(A$,DT)
```

Zwraca *n* pierwszych znaków *wyrażenia_tekstowego*. Podanie *n* większego od długości *wyrażenia_tekstowego* nie powoduje błędu.

```
10 A$="1234567890XEABCDEFGH"
20 X$="1200XL":DT=9
30 PRINT LEFT$(A$,DT)
40 PRINT LEFT$(X$,30)
```

Wynikiem działania tego programu będzie:
123456789
1200XL

## LEN

```
LEN(wyrażenie_tekstowe)
Z=LEN(A$)
```

Podaje długość (ilość znaków) w *wyrażeniu_tekstowym*. Pod uwagę brane są wszystkie znaki łącznie ze spacjami i znakami specjalnymi. Jeżeli zmienna tekstowa nie zawiera żadnych znaków, wynikiem jest 0.

## MID$

```
MID$(wyrażenie_tekstowe,m,n)
PRINT MID$(A$,8,5)
X$=MID$(A$,I,J)
```

Wyodrębnia część znaków z *wyrażenia_tekstowego*. Wskaźnik *m* określa położenie pierwszego znaku w *wyrażeniu_tekstowym*, a *n* określa ilość znaków, jaka ma być wyodrębniona.

```
10 A$="1234567890XEABCDEFGH"
20 PRINT "ATARI 65";MID$(A$,11,2)
```

Wynikiem działania tego programu będzie:
ATARI 65XE

## RIGHT$

```
RIGHT$(wyrażenie_tekstowe,n)
PRINT RIGHT$(A$,8)
X$=RIGHT$(A$,Z)
```

Wyodrębnia *n* ostatnich znaków z *wyrażenia_tekstowego*. Wówczas gdy *n* jest większe niż długość *wyrażenia_tekstowego* wynikiem będzie całe *wyrażenie_tekstowe*.

## SCRN$

```
SCRN$(x,y)
A$=SCRN$(5,5)
X=ASC(SCRN$(10,3)
```

W trybie graficznym podaje, w formacie ATASCII, rejestr koloru piksela na ekranie o współrzędnej poziomej *x* i pionowej *y*, natomiast w trybie tekstowym (za wyjątkiem GRAPHICS 0) kod ATASCII znaku.

Uwagi.

- Jeżeli odczytany punkt jest tłem ekranu (ma wartość 0) wówczas zwracany ciąg jest pusty (w przykładzie A$=""). W pewnych sytuacjach, jak np. zamiana znaku ATASCII na kod dziesiętny, może to spowodować błąd. Dlatego dobrze upewnić się o tym za pomocą polecenia LEN.
- SCRN$ korzysta z kanału #6, dlatego może pojawić się problem z użyciem tego polecenia w GRAPHICS 0, który korzysta z kanału #0. Poniżej, w trzecim przykładzie programu można zobaczyć sposób na ominięcie tego problemu.
- W GRAPHICS 0 znak po odczytaniu zostaje usunięty (na jego miejscu pojawia się spacja), a kursor zostaje przesunięty o jedną pozycję w prawo w stosunku do współrzędnych odczytywanego znaku.

```
10 GRAPHICS 5
20 COLOR 2
30 PLOT 5,5
40 A$=SCRN$(5,5)
50 PRINT"TEN REJESTR KOLORU TO:";ASC(A$)
60 END
```

```
10 GRAPHICS 2
20 PRINT #6,AT(5,5);"A"
30 A$=SCRN$(5,5)
40 PRINT "TEN ZNAK TO: ";A$
50 END
```

```
100 CLOSE#6:OPEN#6,"S:"UPDATE
120 PRINT AT(2,2)"ABC"
140 A$=SCRN$(2,2)
160 IF LEN(A$) THEN PRINT AT(2,5)A$,ASC(A$)
200 END
```

## STR$

```
STR$(wyrażenie_liczbowe)
A$=STR$(1024.18)
A$=STR$(X)
```

Zamienia liczbę reprezentowaną przez *wyrażenie_liczbowe* na odpowiadający jej ciąg tekstowy. Należy zauważyć, że gdy następuje konkatenacja, powstaje przestrzeń między nimi (spacja) będąca domyślnym znakiem plusa - dla liczb dodatnich.

```
10 NR1=-7.125:NR2=1024:NR3=-678
20 A$=STR$(NR1):B$=STR$(NR2):C$=STR$(NR3)
30 X$=A$+B$+C$
40 PRINT A$:PRINT B$:PRINT C$:PRINT X$
```

Wynikiem działania tego programu będzie:

```
-7.125
 1024
-678
-7.125 1024-678
```

## STRING$ (Dostępne tylko z rozszerzeniem na dyskietce).

```
STRING$(n,wyrażenie)
PRINT STRING$(5,"*")
PRINT STRING$(5,42)
A$=STRING$(10,X$)
```

Zwraca ciąg znaków zawarty w *wyrażeniu* (będącym wyrażeniem tekstowym lub wyrażeniem liczbowym lub stałą tekstową lub stałą liczbową) powtórzony *n* razy.

```
5 X$="*Q* "
10 PRINT STRING$(5,"*")
20 PRINT STRING$(5,42)
30 PRINT STRING$(5,X$)
```

Wynikiem działania tego programu będzie:

```
*****
*****
*Q* *Q* *Q* *Q* *Q*
```

## TIME$

Zegar czasu rzeczywistego, który można wykorzystać we własnych programach. Wyświetla czas w formacie „hh:mm:ss" (godziny:minuty:sekundy). Uwaga, zegar nie odmierza precyzyjnie czasu.

```
PRINT TIME$
ZEGAR$=TIME$
```

Odczyt aktualnego stanu zegara.

```
TIME$="godziny:minuty:sekundy"
TIME$="22:09:59"
TIME$="00:00:00"
TIME$="08 30 00"
```

Ustawianie zegara.

## VAL

```
VAL(wyrażenie_tekstowe)
L=VAL(X$)
```

Zamienia ciąg tekstowy na odpowiadającą mu liczbę. Jeżeli pierwszym znakiem nie jest cyfra, otrzymujemy zero, a jeśli ciąg zaczyna się cyframi i kończy innymi znakami, to te znaki są ignorowane, np. VAL("GODZINA 22")=0, a VAL("22 GODZINA")=22

# Pozostałe polecenia

## DEF (Dostępne tylko z rozszerzeniem na dyskietce).

```
DEF nazwa_funkcji=definicja_funkcji
DEF nazwa_funkcji(zmienna)=definicja_funkcji
DEF nazwa_funkcji(zmienna_1,zmienna_2...)=definicja_funkcji
DEF WYNIK(I,J)=(I+J)/2
DEF SUMA=A+B+C+D
DEF JOY(X)=PEEK(632+X)
DEF TEKST$=A$+B$+C$
```

Zdefiniowana przez użytkownika funkcja pełni rolę nowego polecenia BASIC-a. Definicja funkcji musi mieścić się w jednej linii programu. Dopuszczalne jest użycie zmiennych tekstowych, ale efektem wynikowym takiej definicji musi być ciąg tekstowy.

```
100 !DEFINIOWANIE FUNKCJI
110 DEF SUMA=A+B+C+D
120 DEF TEKST$=A$+B$+C$
130 DEF WYNIK(I,J)=(I+J)/2
140 !TEST ZDEFINIOWANYCH FUNKCJI
150 A=1:B=10:C=100:D=1000
160 PRINT SUMA
170 A=0:B=20:C=200:D=2000
180 PRINT SUMA
```

*(Ciąg dalszy listingu na następnej stronie).*

*(Dokończenie listingu z poprzedniej strony).*

```
190 A$="A-":B$="B-":C$="C"
200 PRINT TEKST$
210 A$="a-":C$="c"
220 PRINT TEKST$
230 PRINT WYNIK(100,87)
240 I=5:J=9.9
250 PRINT WYNIK(I,J)
```

Wynikiem działania tego programu będzie:

```
 1111
 2220
A-B-C
a-B-c
 93.5
 7.45
```

## DIM

```
DIM wyrażenie(liczba_elementów)
DIM wyrażenie_1(liczba_elementów),wyrażenie_2(liczba_elementów)...
```

DIM deklaruje *liczbę_elementów* (rozmiar) tablicy liczbowej lub tekstowej.
Najprostszą tablicą jest jednowymiarowa tablica liczbowa:

```
DIM SC(25),A(16)
```

Możliwe jest definiowanie tablic wielowymiarowych liczbowych i tekstowych:

```
DIM SC(10,10),A$(10,2,2,8)
```

- Rozmiar pojedynczego elementu tablicy tekstowej to 255.
- Maksymalna liczba wymiarów to 255, ale w praktyce liczba wymiarów ograniczona jest dostępną pamięcią RAM.
- Numeracja tablic liczbowych standardowo rozpoczyna się od 0, chyba że zostanie to zmienione poleceniem ↦OPTION BASE. Numeracja tablic tekstowych rozpoczyna się od indeksu 1.

```
10 DIM SC(25),A$(10,2,2)
20 SC(0)=85:SC(2)=12
30 A$(10,0,0)="WYMIAR 10-0-0"
40 A$(0,1,2)="WYMIAR  0-1-2"
50 PRINT SC(0),SC(2):PRINT A$(10,0,0):PRINT A$(0,1,2)
```

## END

```
END
```

Kończy (zatrzymuje) działanie programu. Zamyka otwarte kanały IOCB i wyłącza dźwięk. Po zakończeniu programu pojawia się na ekranie znak zachęty „>”. W Atari Microsoft BASIC-u nie ma wymogu kończenia programu poleceniem END.

## LET

```
LET nazwa_zmiennej,wyrażenie
LET LICZNIK=55
LET C=(X+Y)/(X-Y)*10
```

Przypisuje nazwie zmiennej wartość. Znak równości w poleceniu LET oznacza „przypisać”, a nie „równa się” w sensie matematycznym. Na przykład: C = 5 przypisuje wartość 5 do zmiennej C. Liczba po prawej stronie równości może być wyrażeniem (patrz przykład). W praktyce słowo kluczowe LET może być pominięte, znak „=” wystarczy do przypisania.
Uwaga. W przeciwieństwie do Atari BASIC-a nie można przypisać wartości zmiennej, której nazwa jest taka sama jak słowa kluczowego w Microsoft BASIC-u, np. LET GOTO=1 zakończy się komunikatem „SYNTAX ERROR” przy próbie uruchomienia programu.

## OPTION BASE

OPTION BASE 0
OPTION BASE 1
Określa, czy tablice liczbowe zaczynają się od indeksu 0 (OPTION BASE 0) czy 1 (OPTION BASE 1). Polecenie RUN ustawia OPTION BASE 0.
OPTION BASE można użyć w programie tylko jeden raz, najlepiej na początku programu, chyba że zostanie poprzedzone poleceniem ↦CLEAR. W innym przypadku wystąpi błąd.

## RANDOMIZE

RANDOMIZE
RANDOMIZE n
RANDOMIZE 50
RANDOMIZE użyte bez parametru zapewnia generowanie przez polecenie ↦RND różnych liczb pseudolosowych po każdym uruchomieniu programu. Podanie parametru *n* (liczby) ustawia określoną, powtarzalną sekwencję liczb pseudolosowych. Jeżeli w programie nie wystąpi te polecenie, wówczas po każdym uruchomieniu programu losowany będzie przez funkcję RND ten sam zestaw liczb pseudolosowych.
Aby zapewnić niepowtarzalność polecenia RND wystarczy raz użyć RANDOMIZE gdzieś na początku programu.

## TIME

TIME
X=TIME
Odczytuje jako liczbę stan wewnętrznego zegara czasu rzeczywistego (RTCLOK) według wzoru: 65536*PEEK(18)+256*PEEK(19)+PEEK(20). Porównaj z TIME$.

TIME=wyrażenie_liczbowe
TIME=0
TIME=4000
TIME=X-90
Zmiana stanu zegara.

## USR

USR(adres,wyrażenie_liczbowe)
X=USR( A,8)
Z=USR(VARPTR(RESERVE))
Umożliwia przeniesienie wykonywania programu do procedury w języku maszynowym. Jest to zaawansowana funkcja programowania, która pozwala w pełni korzystać ze wszystkich funkcji specjalnych komputera. USR oczekuje dwóch parametrów: *adresu* startu procedury i drugiego, nieobowiązkowego *wyrażenia_liczbowego*. Wartość ta, to zwykle adres tabeli danych, ale może być również przekazaniem parametru dla poszczególnych działań procedury.
Po zakończeniu procedury zwracana jest jako wynik 2-bajtowa liczba (w przykładach do zmiennej „X” i „Z”).

# Stałe i zmienne w programie

Stałe to liczby i inne znaki używane w programie. Pozostają niezmienne w ramach programu. Przykłady stałych: 12, 8, „JAN”, „X”

Zmienne są nazwami do których są przypisane liczby i znaki alfanumeryczne, na przykład: A, LB, A$, TXT$.

Prawidłowy format zmiennej składa się co najmniej z jednego znaku – dużej litery, cyfry lub znaku podkreślenia („_”). Pierwszym znakiem nie może być cyfra, np. zmienna A6 jest poprawna, a 6A nieprawidłowa.

W Microsoft BASIC-u istnieje pięć typów stałych i zmiennych. Wymaga to określenia precyzji. Można to zrobić definiując (DEFINT, DEFSNG, DEFDBL, DEFSTR) pierwszą literę zmiennej lub poprzez użycie identyfikatorów (%, #, $).

**Stała całkowita** (np. 16, 755, -9999)
Są to wszystkie liczby całkowite od -32768 do 32767. Jeżeli stała całkowita zostanie pomnożona przez liczbę rzeczywistą pojedynczej precyzji, to wynikiem będzie liczba rzeczywista pojedynczej precyzji. Wyniki operacji matematycznych są zawsze zapisywane w wyższym typie precyzji.

**Zmienna całkowita** (np. MINI%, X%, ABC%)
Nazwa zmiennej kończąca się znakiem „%" przechowuje liczbę całkowitą. Innym sposobem określenia zmiennej jako całkowitej jest deklaracja:

### DEFINT

DEFINT A, I, J, T-Z
Zmienne, których nazwy rozpoczynają się literami wskazanymi w poleceniu DEFINT (w przykładzie - A, I, J, T, U, V, W, X, Y, Z), będą traktowane jako zmienne całkowite. Zwiększa to szybkość operacji, ale można używać tylko liczb całkowitych z zakresu -32768 ÷ 32767. Na przykład zmienna TEKST będzie pojedynczej precyzji gdyż zaczyna się na literę T. Teraz nie ma potrzeby umieszczania na końcu nazwy zmiennej identyfikatora „%".

Liczby całkowite reprezentowane są przez 2 bajty pamięci.

**Stała rzeczywista pojedynczej precyzji** (np. 64E12, -72999.18, 3.75)
Wszystkie stałe wpisane do programu spoza zakresu liczb całkowitych od -32768 do 32767 są pojedynczej precyzji liczbami rzeczywistymi.

**Zmienna rzeczywista pojedynczej precyzji** (np. Z6H, XY, ABC)
Jeśli typ zmiennej nie został zadeklarowany lub nie posiada na końcu identyfikatora, to jest zmienną rzeczywistą pojedynczej precyzji. Wykładniczy zakres takich liczb to E-38 do E+38.

### DEFSGN

DEFSGN X, J, K, A-D
Nazwy zmiennych zaczynające się od liter wskazanych w poleceniu DEFSNG to zmienne rzeczywiste pojedynczej precyzji. W przykładzie będą to zmienne zaczynające się od liter: X, J, K, A, B, C, D. Na przykład zmienna DANE będzie pojedynczej precyzji gdyż zaczyna się na literę D.

Liczby rzeczywiste pojedynczej precyzji reprezentowane są przez 4 bajty pamięci.

**Stała rzeczywista podwójnej precyzji** (np. 18D7, 7487432D-12)
Liczby te są określane z dokładnością do 16 cyfr po przecinku. W formie wykładniczej należy użyć litery D przed wykładnikiem. Zakres tych liczb jest taki sam jak dla pojedynczej precyzji.

**Zmienna rzeczywista podwójnej precyzji** (np. CD#, X#, ALFA#)
Znak „#" identyfikuje zmienną jako rzeczywistą podwójnej precyzji. Dokładność to 16 znaczących cyfr. Wykładniczy zakres takich liczb to D-38 do D+38.

### DEFDBL

DEFDBL R, C-E
Nazwy zmiennych zaczynające się od liter wskazanych w poleceniu DEFDBL to zmienne rzeczywiste podwójnej precyzji. W przykładzie są to zmienne zaczynające się od liter: C, D, E, R. Na przykład zmienna CENA będzie podwójnej precyzji gdyż zaczyna się na literę C. Teraz nie ma potrzeby umieszczania na końcu nazwy zmiennej identyfikatora „#".

Liczby rzeczywiste podwójnej precyzji reprezentowane są przez 8 bajtów pamięci. W pierwszym bajcie przechowywany jest wykładnik i znak liczby.

Uwagi.

- Poszczególne litery i zakresy liter w deklaracjach muszą być oddzielone od siebie przecinkami.
- Identyfikator na końcu zmiennej ma pierwszeństwo nad deklaracją, np. jeśli zadeklarujemy zmienne zaczynające się literą A jako zmienne całkowite (DEFINT A), a następnie w programie użyjemy zmiennej A#, to będzie ona traktowana jako zmienna rzeczywista podwójnej precyzji. Interpreter potraktuje zmienną A i A# jako dwie różne zmienne.

**Liczby szesnastkowe** (np. &12, &FF, &A2FC)
Często łatwiej jest umieszczać dane lub określić lokalizację w pamięci używając liczb szesnastkowych (HEX). Atari Microsoft BASIC umożliwia posługiwanie się takimi liczbami. Każda liczba poprzedzona identyfikatorem „&" będzie traktowana jako szesnastkowa. Możliwe jest mieszanie liczb dziesiętnych i szesnastkowych np. w liniach DATA.

```
10 DEFINT A,B,H
20 GRAPHICS 0
30 PRINT :PRINT
40 AD=PEEK(88)+256*PEEK(89)+14
50 FOR B=0 TO 10
60 READ HEX
70 POKE AD+B,HEX
80 NEXT
100 DATA &21,&34,&21,&32,&29,0,&18,&10,&10,&38,&2C
```

Program ten drukuje na ekranie napis.

**Stała tekstowa** (np. "SUMA", "NAZWA PLIKU TO:", """TWOJA"" GRA")
Stałe tekstowe zawsze są objęte cudzysłowem. Maksymalna długość ciągu to 120 dowolnych znaków. Wyjątkiem są znaki „♥" - CHR$(0) i „|" - CHR$(124), używane w celach wewnętrznych przez interpreter do oznaczenia końca ciągu. Jeśli użyjemy znaku „♥" lub „|" w tekście, to ciąg zostanie „obcięty" na pozycji tego znaku. Jeśli wewnątrz ciągu zostanie umieszczony podwójny cudzysłów („""), to w wydruku otrzymamy zwykły pojedynczy cudzysłów - patrz wyżej ostatni przykład w nawiasie.

**Zmienna tekstowa** (np. ZZ$, NAZWA$, A$)
Zmienną tego typu kończy znak dolara („$"). Długość ciągu, jaki można podstawić do zmiennej tekstowej, ograniczona jest długością linii programu. Aby wewnątrz zmiennej tekstowej zawrzeć znak cudzysłowu należy go użyć podwójnie („"").

## DEFSTR

DEFSTR A-D, X, Z

Definiuje zmienną zaczynającą się od wskazanej litery jako zmienną tekstową. Długość ciągu, jaki można podstawić do zmiennej tekstowej, ograniczona jest długością linii programu. Należy pamiętać, że identyfikatory na końcu zmiennej („%", „#") mają pierwszeństwo przed deklaracją. W przykładzie zmienna A została zadeklarowana jako tekstowa. Jeżeli użyjemy jej np. z identyfikatorem „%", to zostanie potraktowana jako zmienna całkowita.

**Tablica**
Tablica jest zbiorem zmiennych o tej samej nazwie oznaczonych dla rozróżnienia indeksem, np. A(0), A(1), A(2). Tablice z indeksem nie przekraczającym 10 nie muszą być deklarowane poleceniem DIM. Jako, że standardowo (↦OPTION BASE) początkowy indeks = 0, to uzyskuje się dostęp do 11 elementów tablicy bez konieczności wymiarowania.

```
10 FOR X=0 TO 10
20 TABLICA(X)=RND(100)
30 NEXT X
40 FOR I=0 TO 10
50 PRINT"INDEKS TABLICY:";I;":";TABLICA(I)
60 NEXT I
```

### Tablica wielowymiarowa

Tablica wielowymiarowa jest zbiorem tablic dwuwymiarowych. BASIC przechowuje w pamięci kolejno elementy pierwszej tablicy, następnie drugiej itd. Maksymalna liczba wymiarów to 255, ale w praktyce liczba wymiarów ograniczona jest dostępną pamięcią RAM. Maksymalny rozmiar pojedynczego elementu tablicy tekstowej to 255 znaków.

Tabela ilustruje konstrukcję 7-wymiarowej tablicy po cztery elementy w każdej [DIM(6,3)].

| | Kolumny | | | |
|---|---|---|---|---|
| Wiersze | T(0,0) | T(0,1) | T(0,2) | T(0,3) |
| | T(1,0) | T(1,1) | T(1,2) | T(1,3) |
| | T(2,0) | T(2,1) | T(2,2) | T(2,3) |
| | T(3,0) | T(3,1) | T(3,2) | T(3,3) |
| | T(4,0) | T(4,1) | T(4,2) | T(4,3) |
| | T(5,0) | T(5,1) | T(5,2) | T(5,3) |
| | T(6,0) | T(6,1) | T(6,2) | T(6,3) |

## Operatory arytmetyczne, relacji i logiczne

**Operatory arytmetyczne** w kolejności ich wykonywania przedstawia tabelka. W tworzonych wyrażeniach można mieszać symbole arytmetyczne z operatorami logicznymi. Na przykład wyrażenie A/C > D*A jest prawidłowe.

| Operator | Znaczenie |
|---|---|
| ( ) | Arytmetyka w nawiasach wykonywana jest w pierwszej kolejności. |
| = | Znak równości. |
| _ | Liczba ujemna, np. -12, -A, -3.3. Nie jest to jednak odejmowanie, a nadanie liczbie przeciwnego znaku. |
| ^ | Wykładnik, np. 2^3 to dwa do potęgi trzeciej. |
| * | Mnożenie. |
| / | Dzielenie. |
| + | Dodawanie. |
| - | Odejmowanie. |

**Operatory relacji** obliczane są od lewej do prawej strony. Znaki w zmiennych tekstowych traktowane są jako numery kodu ATASCII, np. A to 65, a B to 66. W tabeli przedstawione zostały dopuszczalne operatory relacji.

| Operator | Znaczenie |
|---|---|
| = | Równość. Jeśli A = B to wynikiem jest prawda (-1), inaczej fałsz (0). |
| <> | Nierówność. Jeśli A ma inną wartość niż B, to wyrażenie A <> B (lub A >< B) jest prawdziwe (-1), gdy taką samą - fałsz (0). |
| >< | |
| < | Mniejszy od. Jeśli A < B to wynikiem jest prawda (-1), inaczej fałsz (0). |
| > | Większy od. Jeśli A > B to wynikiem jest prawda (-1), inaczej fałsz (0). |
| <= | Mniejszy lub równy. Jeśli A <= B (lub A =< B) to wynikiem jest prawda (-1), inaczej fałsz (0). |
| =< | |
| >= | Większy lub równy. Jeśli A >= B (lub A => B) to wynikiem jest prawda (-1), inaczej fałsz (0). |
| => | |

**Operatory logiczne** wykonywane są w następującej kolejności:

| Operator | Znaczenie |
|---|---|
| NOT | Negacja. Zmienia na przeciwną wartość poszczególnych bitów w 1- lub 2-bajtowym operandzie (dziesiętnie -32768 ÷ 32767).<br>Wynikiem dla operandu -1, 0, 1 będzie odpowiednio 0, -1 i -2. |
| AND | Iloczyn logiczny. Porównuje te same bity dwóch liczb 1- lub 2-bajtowych (-32768 ÷ 32767), gdy oba są ustawione, wynikiem jest 1, w innym przypadku zero.<br>Jeśli argumentami jest wyrażenie logiczne, to prawda=-1, a fałsz=0. W poniższym przykładzie wynikiem będzie -1 (prawda).<br>`10 A=7:B=13`<br>`20 X= A=7 AND B=13`<br>`30 PRINT X`<br>A B A AND B<br>0 0 0<br>0 1 0<br>1 0 0<br>1 1 -1 |
| OR | Alternatywa logiczna. Porównuje te same bity dwóch liczb 1- lub 2-bajtowych (-32768 ÷ 32767), gdy oba są skasowane (0), to wynikiem jest również zero, w innym przypadku 1.<br>Jeśli argumentami jest wyrażenie logiczne, to prawda=-1, a fałsz=0. W poniższym przykładzie wynikiem będzie -1 (prawda).<br>`10 A=7:B=2`<br>`20 X= A=7 OR B=13`<br>`30 PRINT X`<br>A B A OR B<br>0 0 0<br>0 1 -1<br>1 0 -1<br>1 1 -1 |
| XOR | Alternatywa wykluczająca (*exclusive or*). Porównuje te same bity dwóch liczb 1- lub 2-bajtowych (-32768 ÷ 32767), gdy są różne (0,1 lub 1,0) to wynikiem jest 1, w innym przypadku zero.<br>Jeśli argumentami jest wyrażenie logiczne, to prawda=-1, a fałsz=0. W poniższym przykładzie wynikiem będzie 0 (fałsz).<br>`10 A=7:B=13`<br>`20 X= A=7 XOR B=13`<br>`30 PRINT X`<br>A B A XOR B<br>0 0 0<br>0 1 -1<br>1 0 -1<br>1 1 0 |

# Obsługa manipulatorów i klawiszy konsoli

Atari Microsoft BASIC II nie posiada specjalnych poleceń obsługujących manipulatory, dlatego trzeba posłużyć się poleceniem PEEK. Poniżej podane są lokalizacje (komórki) pamięci związane z manipulatorami. Możliwe jest również użycie takich samych zmiennych jak w opisie, a nawet użycie polecenia DEF do zdefiniowania własnych poleceń obsługujących paletki i dżojstiki.

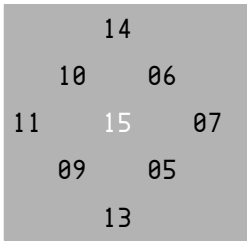
Numer paletki (PADDLE) = komórka pamięci (wartości dziesiętne)
PADDLE(0) = 624
PADDLE(1) = 625
PADDLE(2) = 626
PADDLE(3) = 627
PADDLE(4) = 628
PADDLE(5) = 629
PADDLE(6) = 630
PADDLE(7) = 631

Przycisk paletki (PTRIG) = komórka pamięci (wartości dziesiętne)
PTRIG(0) = 636
PTRIG(1) = 637
PTRIG(2) = 638
PTRIG(3) = 639
PTRIG(4) = 640
PTRIG(5) = 641
PTRIG(6) = 642
PTRIG(7) = 643

Numer dżojstika (STICK) = komórka pamięci (wartości dziesiętne)
STICK(0) = 632
STICK(1) = 633
STICK(2) = 634
STICK(3) = 635

Przycisk dżojstika (STRIG) = komórka pamięci (wartości dziesiętne)
STRIG(0) = 644
STRIG(1) = 645
STRIG(2) = 646
STRIG(3) = 647

Wartość, jaką przyjmuje komórka STICK w zależności od położenia dźwigni dżojstika



Komórka pamięci o adresie 53279 zwraca liczbę określającą, który klawisz lub klawisze konsoli zostały naciśnięte.

Wartość - naciśnięty klawisz
7 - żaden
6 - START
5 - SELECT
4 - START+SELECT
3 - OPTION
2 - START+OPTION
1 - SELECT+OPTION
0 - START+SELECT+OPTION

Stan klawisza HELP można odczytać poleceniem PEEK(732).
Uwaga. Zawartość tej komórki nie jest automatycznie kasowana, czyli zmienia wartość po naciśnięciu HELP i przechowuje ją do następnego naciśnięcia HELP lub skasowania poleceniem POKE 732,0.

Wartość - naciśnięty klawisz
 17 - HELP
 81 - HELP+SHIFT
145 - HELP+CONTROL
209 - HELP+CONTROL+SHIFT

# Aneks I: komunikaty błędów

| Kod | Komunikat | Opis |
|---|---|---|
| 1 | NEXT WITHOUT FOR ERROR IN *nr_linii* | Użyto NEXT bez wcześniejszego FOR lub niewłaściwa kolejność zagnieżdżania pętli. |
| 2 | SYNTAX ERROR IN *nr_linii* | Błąd składni (nieprawidłowe znaki interpunkcyjne, otwarty nawias, niedozwolone znaki, błędne słowa kluczowe). |
| 3 | RETURN WITHOUT GOSUB ERROR | Użyto RETURN bez wcześniejszego GOSUB. |
| 4 | OUT OF DATA IN *nr_linii* | Zabrakło danych dla polecenia INPUT albo READ w liniach DATA lub odczytano już wszystkie dane z urządzenia zewnętrznego. Błąd może wystąpić jeśli nie użyto RESTORE przy próbie ponownego odczytu danych lub został podany nieprawidłowy numer linii po RESTORE. |
| 5 | FUNCTION CALL ERROR IN *nr_linii* | Próba wykonania operacji przy użyciu niedozwolonych parametrów (np. pierwiastek kwadratowy z liczby ujemnej). |
| 6 | OVERFLOW | Liczba lub wynik operacji matematycznej spoza dopuszczalnego zakresu (zbyt mała lub zbyt duża liczba). |
| 7 | OUT OF MEMORY ERROR | Cała dostępna pamięć została wykorzystana. Ten problem może się pojawić m.in. przy deklaracji bardzo dużych tablic, wielu zagnieżdżonych pętlach i poleceniach GOSUB. |
| 8 | UNDEF'D LINE ERROR IN *nr_linii* | Próba odwołania się do nieistniejącej linii. |
| 9 | SUBSCRIPT ERROR IN *nr_linii* | Próba przypisania zmiennej indeksowej (tabeli) wartości z użyciem indeksu spoza dopuszczalnego zakresu dla tej zmiennej. |
| 10 | REDEF'N ERROR IN *nr_linii* | Próba zadeklarowania (DIM) zmiennej już wcześniej zadeklarowanej lub zwymiarowanej domyślnie. |
| 11 | DIVISION BY ZERO | Próba dzielenia przez zero (jest niedopuszczalna, ponieważ dzielenie jest odwrotnością mnożenia, a nie istnieje liczba, która pomnożona przez zero da wynik inny niż zero). |
| 12 | ILLEGAL DIRECT ERROR | Użycie INPUT, GET lub DEF w trybie bezpośrednim jest niedopuszczalne. |
| 13 | TYPE MISMATCH ERROR IN *nr_linii* | Próba przypisania zmiennej numerycznej ciągu tekstowego lub odwrotnie. |
| 14 | FILE I/O ERROR | Ogólny błąd wejścia/wyjścia (I/O). |
| 15 | QUANTITY TOO BIG ERROR IN *nr_linii* | Zmienna tekstowa przekracza 255 znaków. |
| 16 | FORMULA TOO COMPLEX ERROR | Ciąg operacji lub wyrażenie matematyczne jest zbyt skomplikowane. Należy podzielić na kilka kroków. |
| 17 | CAN'T CONTINUE ERROR | Wznowienie programu poleceniem CONT jest niemożliwe (np. w wyniku użycia END). |
| 18 | UNDEF'D FUNCTION ERROR | Funkcja USR nie może być wykonana, np. z powodu złego adresu startu, błędu w procedurze, złego adresu procedury. |
| 19 | NO RESUME ERROR IN *nr_linii* | Po przejęciu błędu przez ON ERROR nie wystąpiło polecenie RESUME. |

*Komunikaty błędów - ciąg dalszy tabeli.*

| | | |
|---|---|---|
| 20 | RESUME WITHOUT ERROR IN *nr_linii* | RESUME wystąpiło przed poleceniem ON ERROR. |
| 21 | FOR WITHOUT NEXT ERROR | NEXT wystąpiło przed poleceniem FOR. |
| 22 | LOCK ERROR | Próba modyfikacji lub listowania programu zapisanego z opcją LOCK. |
| 23 | TIME ERROR | Konflikt przerwań czasowych. Po poleceniu AFTER nie następuje RESUME. |

Komunikaty błędów o numerach powyżej 127 związane są z DOS-em i urządzeniami zewnętrznymi, takimi jak stacja dysków czy drukarka. W przypadku wystąpienia takiego błędu zajrzyj do instrukcji obsługi używanego DOS-a lub określonego urządzenia zewnętrznego.

# Aneks II: lista kodów funkcji STATUS

Uwaga. Interpretacja tych kodów może się nieco różnić w zależności od rodzaju DOS-a i urządzenia zewnętrznego.

| HEX | DEC | Znaczenie |
|---|---|---|
| 01 | 1 | Operacja zakończona bez błędów. |
| 03 | 3 | Napotkano koniec pliku. |
| 80 | 128 | Naciśnięto klawisz BREAK. |
| 81 | 129 | Próba otwarcia IOCB będącego w użyciu. |
| 82 | 130 | Nieistniejące (niepodłączone) urządzenie. |
| 83 | 131 | Otwarty tylko do zapisu (nie można odczytywać). |
| 84 | 132 | Błędne polecenie. |
| 85 | 133 | Nie otworzono urządzenia lub pliku. |
| 86 | 134 | Błędny numer IOCB (spoza zakresu 1 ÷ 7). |
| 87 | 135 | Otwarty tylko do odczytu (nie można zapisywać). |
| 88 | 136 | Napotkano koniec zbioru (nie ma więcej danych). |
| 89 | 137 | Niepełny rekord. |
| 8A | 138 | Przekroczony limit czasu na odpowiedź (urządzenie nie odpowiada). |
| 8B | 139 | Potwierdzenie niezgodności - NAK (błąd transmisji szeregowej). |
| 8C | 140 | Uszkodzona komunikacja komputera z peryferiami. Niekiedy występuje przy wadliwej dyskietce lub kasecie. Częściej wskazuje na uszkodzenie połączenia (kabla sygnałowego), komputera lub urządzenia peryferyjnego. |
| 8D | 141 | Kursor poza dopuszczalnym zakresem dla wybranego trybu graficznego. |
| 8E | 142 | Błąd transmisji poprzez złącze szeregowe. |
| 8F | 143 | Błędna suma kontrolna podczas transmisji przez złącze szeregowe. |
| 90 | 144 | Urządzenie zewnętrzne nie jest w stanie wykonać zadania, np. zapisać pliku na zabezpieczonym dysku. |
| 91 | 145 | Niewłaściwie użyto poleceń dla grafiki w aktualnym trybie graficznym. |
| 92 | 146 | Funkcja nie jest obsługiwana przez sterownik urządzenia. |
| 93 | 147 | Za mało wolnej pamięci dla wybranego trybu graficznego. |
| A0 | 160 | Błędny numer urządzenia zewnętrznego, np. stacji dysków. |
| A1 | 161 | Za dużo równocześnie otwartych plików na dysku. |
| A2 | 162 | Dysk pełny (nie ma miejsca do zapisu). |
| A3 | 163 | Niepowodzenie próby transmisji I/O (np. nie można odczytać dyskietki). |
| A4 | 164 | Wewnętrzna niezgodność numeracji w pliku (np. bajt łącznikowy w sektorze dyskietki wskazuje na sektor poza plikiem). |
| A5 | 165 | Błędna nazwa pliku. |
| A6 | 166 | Polecenie POINT wskazuje na nieistniejący bajt lub położony poza plikiem. |
| A7 | 167 | Plik zabezpieczony (próba zmiany lub skasowania zabezpieczonego pliku). |
| A8 | 168 | Błędne polecenie operacji dyskowej. |
| A9 | 169 | Katalog dysku pełny (zawiera 64 pliki). |

*Lista kodów funkcji STATUS - ciąg dalszy tabeli.*

| | | |
|---|---|---|
| AA | 170 | Nie znaleziono pliku. |
| AB | 171 | Błędne parametry POINT. |
| AC | 172 | Błąd dołączania. |
| AD | 173 | Podczas formatowana stwierdzono wadliwy sektor. |
| AE | 174 | Duplikat nazwy pliku. |
| AF | 175 | Nie można wczytać tego pliku z poziomu DOS-a. |
| B0 | 176 | Nieobsługiwany format dysku. |
| B1 | 177 | Uszkodzona dyskietka. |

# Aneks III: kody ATASCII i keycode

Tabela kodów ATASCII.

| Znak | ATASCII | IC | Znak | ATASCII | IC | Znak | ATASCII | IC | Znak | ATASCII | IC |
|---|---|---|---|---|---|---|---|---|---|---|---|
|  | 0 | 64 |  | 32 | 0 |  | 64 | 32 |  | 96 | 96 |
|  | 1 | 65 |  | 33 | 1 |  | 65 | 33 |  | 97 | 97 |
|  | 2 | 66 |  | 34 | 2 |  | 66 | 34 |  | 98 | 98 |
|  | 3 | 67 |  | 35 | 3 |  | 67 | 35 |  | 99 | 99 |
|  | 4 | 68 |  | 36 | 4 |  | 68 | 36 |  | 100 | 100 |
|  | 5 | 69 |  | 37 | 5 |  | 69 | 37 |  | 101 | 101 |
|  | 6 | 70 |  | 38 | 6 |  | 70 | 38 |  | 102 | 102 |
|  | 7 | 71 |  | 39 | 7 |  | 71 | 39 |  | 103 | 103 |
|  | 8 | 72 |  | 40 | 8 |  | 72 | 40 |  | 104 | 104 |
|  | 9 | 73 |  | 41 | 9 |  | 73 | 41 |  | 105 | 105 |
|  | 10 | 74 |  | 42 | 10 |  | 74 | 42 |  | 106 | 106 |
|  | 11 | 75 |  | 43 | 11 |  | 75 | 43 |  | 107 | 107 |
|  | 12 | 76 |  | 44 | 12 |  | 76 | 44 |  | 108 | 108 |
|  | 13 | 77 |  | 45 | 13 |  | 77 | 45 |  | 109 | 109 |
|  | 14 | 78 |  | 46 | 14 |  | 78 | 46 |  | 110 | 110 |
|  | 15 | 79 |  | 47 | 15 |  | 79 | 47 |  | 111 | 111 |
|  | 16 | 80 |  | 48 | 16 |  | 80 | 48 |  | 112 | 112 |
|  | 17 | 81 |  | 49 | 17 |  | 81 | 49 |  | 113 | 113 |
|  | 18 | 82 |  | 50 | 18 |  | 82 | 50 |  | 114 | 114 |
|  | 19 | 83 |  | 51 | 19 |  | 83 | 51 |  | 115 | 115 |
|  | 20 | 84 |  | 52 | 20 |  | 84 | 52 |  | 116 | 116 |
|  | 21 | 85 |  | 53 | 21 |  | 85 | 53 |  | 117 | 117 |
|  | 22 | 86 |  | 54 | 22 |  | 86 | 54 |  | 118 | 118 |
|  | 23 | 87 |  | 55 | 23 |  | 87 | 55 |  | 119 | 119 |
|  | 24 | 88 |  | 56 | 24 |  | 88 | 56 |  | 120 | 120 |
|  | 25 | 89 |  | 57 | 25 |  | 89 | 57 |  | 121 | 121 |
|  | 26 | 90 |  | 58 | 26 |  | 90 | 58 |  | 122 | 122 |
|  | 27 | 91 |  | 59 | 27 |  | 91 | 59 |  | 123 | 123 |
|  | 28 | 92 |  | 60 | 28 |  | 92 | 60 |  | 124 | 124 |
|  | 29 | 93 |  | 61 | 29 |  | 93 | 61 |  | 125 | 125 |
|  | 30 | 94 |  | 62 | 30 |  | 94 | 62 |  | 126 | 126 |
|  | 31 | 95 |  | 63 | 31 |  | 95 | 63 |  | 127 | 127 |

- IC – kod wewnętrzny Atari (internal code), czyli kolejność w jakiej ustawione są znaki w generatorze znaków.
- Znaki w inwersie uzyskuje się przez dodanie 128 do wartości kodu ATASCII.

Tabela keycode (kody klawiszy).

W kolejności ATASCII.

| Klawisz | Kod | Klawisz | Kod | Klawisz | Kod | Klawisz | Kod |
|---|---|---|---|---|---|---|---|
| * | 7 | 7 | 51 | G | 61 | T | 45 |
| + | 6 | 8 | 53 | H | 57 | U | 11 |
| , | 32 | 9 | 48 | I | 13 | V | 16 |
| - | 14 | ; | 2 | J | 1 | W | 46 |
| . | 34 | < | 54 | K | 5 | X | 22 |
| / | 38 | = | 15 | L | 0 | Y | 43 |
| 0 | 50 | > | 55 | M | 37 | Z | 23 |
| 1 | 3 | A | 63 | N | 35 | Back Space | 52 |
| 2 | 6 | B | 21 | O | 8 | Caps | 60 |
| 3 | 26 | C | 18 | P | 10 | Esc | 28 |
| 4 | 24 | D | 58 | Q | 47 | Return | 12 |
| 5 | 29 | E | 42 | R | 40 | Tab | 44 |
| 6 | 27 | F | 56 | S | 62 |  | 39 |

W kolejności keycode.

| Kod | Klawisz | Kod | Klawisz | Kod | Klawisz | Kod | Klawisz |
|---|---|---|---|---|---|---|---|
| 0 | L | 16 | V | 34 | . | 50 | 0 |
| 1 | J | 18 | C | 35 | N | 51 | 7 |
| 2 | ; | 21 | B | 37 | M | 52 | Back Space |
| 5 | K | 22 | X | 38 | / | 53 | 8 |
| 6 | + | 23 | Z | 39 |  | 54 | < |
| 7 | * | 24 | 4 | 40 | R | 55 | > |
| 8 | O | 26 | 3 | 42 | E | 56 | F |
| 10 | P | 27 | 6 | 43 | Y | 57 | H |
| 11 | U | 28 | Esc | 44 | Tab | 58 | D |
| 12 | Return | 29 | 5 | 45 | T | 60 | Caps |
| 13 | I | 30 | 2 | 46 | W | 61 | G |
| 14 | - | 31 | 1 | 47 | Q | 62 | S |
| 15 | = | 32 | , | 48 | 9 | 63 | A |

## Bibliografia

*Atari Microsoft BASIC II Reference Manual.* © 1983 Atari, Inc.
Wojciech Zientara: *Języki Atari XL/XE cz. 1.* Wyd. SOETO, Warszawa 1989

Oryginalna czcionka: Envy Code R.