TURBO-BASIC XL 1.5

(c) 1985 Frank Ostrowski

TURBO-BASIC XL 1.5

Nowe polecenia z przykładami

Bluki, 2013

Spis treści

Alfabetyczny spis poleceń

Wprowadzenie

W porównaniu do Atari BASIC-a, Turbo BASIC XL 1.5 (TBXL) oferuje 42 polecenia i 22 funkcje więcej; także daje użytkownikowi ok. 1600 bajtów więcej wolnej pamięci RAM, a to dlatego, że część swojego kodu „ukrywa" w pamięci RAM pod systemem operacyjnym. Posiada zaawansowane polecenia do obsługi grafiki, rozszerzoną obsługę dźwięku, elementy programowania strukturalnego (m. in. procedury) i obsługę DOS-a. Pozwala na wprowadzanie poleceń nie tylko dużymi literami, ale również małymi i znakami w inwersie. Ponadto jest 3 do 5 razy szybszy.

W programie napisanym w TBXL możemy użyć aż 256 różnych zmiennych. W większości interpreterów BASIC-a, w tym Atari BASIC-u – 128.

TBXL wyposażony jest we wszystkie polecenia występujące w Atari BASIC-u. Umożliwia to uruchamianie (z pewnymi ograniczeniami) programów napisanych pod interpreterem Atari BASIC. Warto zwrócić uwagę, że taki program będzie działał pod TBXL szybciej, co niekoniecznie musi być zaletą – zwłaszcza w przypadku gier zręcznościowych.

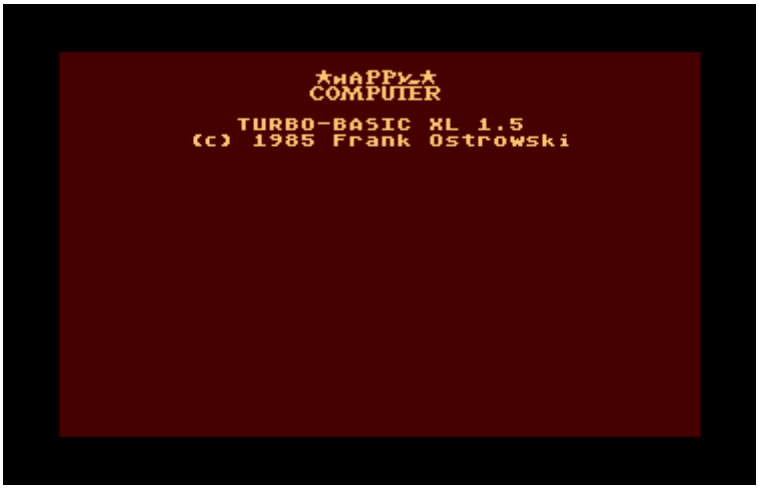
Niżej opisane zostaną tylko polecenia i funkcje charakterystyczne dla TBXL, czyli takie, które odróżniają TBXL od Atari BASIC-a. Jeśli, czytelniku, teraz zaczynasz przygodę z BASIC-em, to zanim przejdziesz do dalszej części opisu, powinieneś zapoznać się z jednym z wielu dostępnych podręczników podstaw programowania w Atari BASIC-u.

Uwagi.

Opis poleceń zorganizowany jest w następujący sposób:

- na pomarańczowym tle nazwa polecenia i w nawiasie kwadratowym dopuszczalny skrót (jeśli jest)
  - a niżej:
- format polecenia wraz z opisem;
- poprzedzony „•►" przykład użycia polecenia;
- na niebieskim tle – listing będący prostym przykładem użycia polecenia (niekiedy w przykładach listingów przed niektórymi liczbami pojawia się znaczek „%", jaką spełnia rolę opisano na str. 14).

••• – dowolne instrukcje lub linie programu.



Polecenia TBXL

Operacje dyskowe

Uwagi.
➢ W opisie założono, że TBXL będzie pracował pod DOS-em 2.5 lub z nim zgodnym. W przypadku użycia innego DOS-a mogą wystąpić różnice w działaniu niektórych niżej opisanych poleceń. W razie wątpliwości zajrzyj do instrukcji używanego DOS-a.
➢ Jeśli używamy napędu nr 1 to w specyfikacji numer napędu (oznaczony jako „n") można pominąć. Np. zamiast „D1:nazwa", piszemy „D:nazwa".

BLOAD [BL.]

BLOAD"Dn:nazwa" – wczytuje plik binarny z napędu dyskowego o numerze *n* i wskazanej *nazwie*, ale nie uruchamia go. Odpowiada to poleceniu „L" DOS-a z opcją /N.

•► BLOAD"D:ABC.TXT"
•► BLOAD"D2:PROGRAM.COM"

BRUN

BRUN"Dn:nazwa" – wczytuje plik binarny z napędu dyskowego o numerze *n* i wskazanej *nazwie*, a następnie go uruchamia. Odpowiada to poleceniu „L" DOS-a.

•► BRUN"D2:PROGRAM.COM"

DELETE [DEL.]

DELETE"Dn:nazwa" – kasuje plik o wskazanej *nazwie* w napędzie *n*. Plik i dyskietka nie mogą być zabezpieczone przed zapisem, wówczas otrzymamy komunikat o błędzie. Odpowiada poleceniu „D" DOS-a.

•► DELETE"D:PROGRAM.COM"
•► DELETE"D2:PROGRAM.*"

DIR

DIR"Dn:nazwa" – odczytuje katalog dysku. W przypadku użycia tego polecenia bez parametrów (tylko DIR), TBXL przyjmuje domyślnie, że chodzi o napęd numer 1. DIR odpowiada poleceniu „A" DOS-a.

•► DIR – odczytuje całą zawartość katalogu dysku w napędzie nr 1.
•► DIR"D8:*.*" - wyświetla nazwy wszystkich plików znajdujących się na dysku w napędzie nr 8.
•► DIR"D:*.BAS" – wyświetla wszystkie pliki o rozszerzeniu „BAS" znajdujące się na dysku w napędzie nr 1.
Jeśli poszukiwanych plików na dysku nie ma, wówczas zostanie wyświetlona tylko informacja o ilości wolnych sektorów na dysku.

LOCK

LOCK"Dn:nazwa" – zabezpiecza plik przed zapisem lub skasowaniem. Próba użycia tego polecenia na pliku już wcześniej zabezpieczonym nie powoduje błędu. Odpowiada poleceniu „F” DOS-a.

•► LOCK"D:*.*" - zabezpiecza wszystkie pliki na dysku nr 1.
•► LOCK"D8:*.BAS" – zabezpiecza tylko pliki o rozszerzeniu „BAS” w napędzie nr 8.
•► LOCK"D2:A*.*" - zabezpiecza wyłącznie pliki o nazwie rozpoczynającej się literą „A” i dowolnym rozszerzeniu, w napędzie nr 2.

RENAME [REN.]

RENAME"Dn:stara_nazwa,nowa_nazwa" – zmienia nazwę pliku na dysku. Odpowiada poleceniu „E” DOS-a.

•► RENAME"D:GRA.BAS,NOWAGRA.BAS" – zmienia dotychczasową nazwę „GRA.BAS”, na nową, „NOWAGRA.BAS” (w napędzie nr 1).
•► RENAME"D:PLIK.COM,PLIK.XEX" – zamienia rozszerzenie programu o nazwie „PLIK” z „COM” na „XEX” (w napędzie nr 1).

Uwaga. Choć polecenie RENAME pozwala na używanie „dżokerów” („*”, „?”), to nie zaleca się się ich używać. Może się się zdarzyć, że w wyniku przypadkowego błędu ze strony użytkownika, po zmianie nazwy dwa lub więcej plików w tym samym katalogu dysku będzie posiadało identyczną nazwę. Wówczas możliwe będzie odczytanie tylko pierwszego ze zdublowanych. Do pozostałych dostęp będzie utracony.

UNLOCK [UNL.]

UNLOCK"Dn:nazwa" – odbezpiecza plik, wcześniej zabezpieczony poleceniem LOCK. Próba użycia tego polecenia na niezabezpieczonym pliku nie powoduje błędu. Odpowiada poleceniu „G” DOS-a.

•► UNLOCK"D:*.*" - odbezpiecza wszystkie pliki na dysku nr 1.
•► UNLOCK"D:GRA.BAS" – odbezpiecza tylko plik o nazwie „GRA.BAS” na dysku nr 1.

Programowanie strukturalne – procedury i pętle

Uwaga. Spójrz także na polecenie EXIT w rozdziale „Sterownie przebiegiem programu” na str. 18.

PROC-ENDPROC [-ENDP.]

PROC nazwa
•••
ENDPROC

Instrukcje i linie programu zawarte pomiędzy *PROC nazwa*, a *ENDPROC* tworzą procedurę. Procedurę można przyrównać do podprogramu wywoływanego przez GOSUB i kończącego się poleceniem RETURN w Atari BASIC-u. Są jednak różnice na korzyść procedury.

Procedura posiada nazwę, która może, a nawet powinna być znacząca, czyli określać co procedura robi (patrz przykłady). Zwiększa to przejrzystość kodu, ułatwia pisanie programu, w tym wnoszenie poprawek lub modyfikacji. Ponadto odwołanie do nazwy jest szybsze niż do numeru linii (GOSUB), gdyż interpreter nie musi przeszukiwać programu, aby odnaleźć linię o określonym numerze, tylko od razu skacze pod adres „ukrywający się" pod nazwą procedury. Polecenie PROC musi występować jako pierwsze w linii programu.

Wywołanie procedury uzyskuje się poleceniem EXEC nazwa_wywoływanej_procedury.

•► PROC RYSUJ_POLE_GRY
•► PROC OBLICZ_WYNIK

```
530 PROC USTAW_PMG_NA_ZERO
535   FOR X=53248 TO 53255:POKE X,0:NEXT X
540 ENDPROC
```

Powyższa procedura ustawia na zero poziomą pozycję wszystkich graczy i pocisków (PMG).

EXEC

EXEC nazwa – tak jak w Atari BASIC-u polecenie GOSUB nr_linii wywołuje podprogram, tak w TBXL wykonuje to *EXEC nazwa*, wywołując procedurę, która jest niczym innym jak podprogramem z nazwą. Więcej o procedurze wyżej, przy poleceniu PROC-ENDPROC.

•► EXEC USTAW_PMG_NA_ZERO

ON-EXEC

ON warunek EXEC nazwa_procedury – warunkowe wywołanie procedury. Gdy *warunek* po ON jest spełniony następuje wywołanie procedury o nazwie podanej po EXEC. Polecenie to funkcjonuje analogicznie do ON warunek GOSUB numer_linii w Atari BASIC-u.

•► ON A$="ABC" EXEC NAPIS
•► ON A EXEC POZIOM1,POZIOM2,POZIOM3

DO-LOOP

DO ••• LOOP – bezparametrowa pętla bez końca. Elementy programu zawarte pomiędzy DO a LOOP będą cały czas powtarzane. Jest to znacznie lepsze (czytelniejsze i szybsze) rozwiązanie niż stosowane w Atari BASIC-u:
1000 REM POCZATEK PETLI
•••
1900 GOTO 1000

•► DO :? "ABC":LOOP

```
1000 DO
1110 PRINT "TO JEST PETLA BEZ KONCA"
•••
1900 LOOP
```

Jedynym dopuszczalnym w programowaniu strukturalnym sposobem „rozerwania” tej pętli jest polecenie EXIT.

IF-ELSE-ENDIF

IF warunek ••• ELSE ••• ENDIF – gdy *warunek* jest spełniony, wówczas wykonywana jest część programu przed ELSE i pomijana między ELSE i ENDIF. Natomiast gdy warunek nie jest spełniony, pomijana jest część programu przed ELSE, a wykonywana między ELSE i ENDIF.

•► IF A$="ABC":B$="DEF":ELSE :B$="XYZ":ENDIF

```
860 IF WYNIK>100
870   PRINT "BRAWO!"
880 ELSE
890   PRINT "BUUU!"
900 ENDIF
```

Słowo kluczowe ELSE może być pominięte, wówczas polecenie będzie działać podobnie jak znane z Atari BASIC-a IF THEN..., z tą różnicą, że wewnątrz takiej pętli IF-ENDIF może znajdować się dowolna ilość linii programu.

```
800 IF W>%0
810   IF W<10 THEN ? "TAK SOBIE"
820   IF W>=10 AND W<100 THEN ? "DOBRZE"
830   IF W>=100 AND W<200 THEN ? "BARDZO DOBRZE"
840   IF W>=200 AND W<300 THEN ? "SUPER!"
850   IF W>=300 AND W<400 THEN ? "NADZWYCZAJNIE!!!"
•••
900 ENDIF
```

REPEAT-UNTIL [REP.-UN.]

REPEAT ••• UNTIL warunek – ciąg instrukcji zawarty pomiędzy słowami kluczowymi REPEAT i UNTIL jest powtarzany tak długo, jak długo *warunek* pozostaje niespełniony, czyli ma wartość logiczną 0. Gdy *warunek* zostanie spełniony, czyli przyjmie wartość logiczną 1, nastąpi opuszczenie pętli. Warto zauważyć, że sprawdzenie *warunku* następuje na końcu pętli, po napotkaniu polecenia UNTIL. Oznacza to, że instrukcje zawarte między REPEAT i UNTIL zawsze zostaną wykonane co najmniej jeden raz.

•► REPEAT :? "Test":UNTIL PEEK(53279)=6

```
100 REPEAT
•••
200 UNTIL X=12
210 END
```

W tym przykładzie pętla będzie wykonywana do momentu osiągnięcia przez zmienną X wartości 12.

WHILE-WEND [W.-WE.]

WHILE warunek ••• WEND – tak długo, jak *warunek* będzie spełniony (czyli posiadać będzie wartość logiczną 1) pętla będzie wykonywana.

Warunek sprawdzany jest na początku pętli. Jeśli przed pierwszym obiegiem nie będzie spełniony (czyli jego wartość logiczna = 0), to pętla nie zostanie wykonana ani razu.

```
300 WHILE PEEK(53279)=7:WEND
310 END
```

W tym przypadku pętla będzie wykonywana dopóki nie zostanie naciśnięty klawisz konsoli (START, SELECT lub OPTION).

Programowanie strukturalne – etykiety

#

 # nazwa – etykieta; nadaje bieżącej linii programu nazwę. Etykieta musi być pierwszym (lub jedynym) poleceniem w linii. Z punktu widzenia interpretera jest nazwą zmiennej, ponieważ jest umieszczana razem innymi „zwykłymi" zmiennymi w tabeli nazw zmiennych. Nadanie etykiety linii pozwala na wykonywanie poleceń skoków nie do numeru linii, a właśnie do etykiety. Takie odwołanie działa szybciej, gdyż skok następuje od razu do adresu „ukrywającego się" pod nazwą etykiety. W przypadku odwołania się do numeru linii interpreter musi przeszukać program i dopiero po znalezieniu właściwego numeru linii wykonuje do niej skok. Wskazane jest, aby nazwa była znacząca, czyli mówiła „o co chodzi".

```
300 ON A GO# POZIOM1,POZIOM2,POZIOM3
400 IF A<%1 OR A>%3 THEN ? "POZA ZAKRE
SEM":GO# KONIEC
500 # POZIOM1:? "P1":GO# KONIEC
600 # POZIOM2:? "P2":GO# KONIEC
700 # POZIOM3:? "P3"
1000 # KONIEC:END
```

W powyższym przykładzie programu widać sposób użycia etykiet, a niżej zostały omówione polecenia współpracujące z etykietami, również te z przykładu.

GO#

GO# nazwa – bezwarunkowy skok do linii o wskazanej *nazwie* – odpowiednik polecenia GOTO nr_linii w Atari BASIC-u. Patrz też przykład wyżej - # nazwa.

•► GO# KONIEC_GRY

ON-GO#

ON warunek GO# nazwa – polecenie skoku warunkowego do linii o wskazanej *nazwie*. Odpowiednikiem w Atari BASIC-u jest polecenie ON warunek GOTO numer_linii. Patrz też przykład wyżej - # nazwa.

•► ON FUNKCJA=10 GO# DRUGA_FAZA
•► ON A GO# POZIOM1,POZIOM2,POZIOM3

RESTORE# [RES.#]

RESTORE# nazwa – odpowiednik polecenia RESTORE numer_linii w Atari BASIC-u.

•► RESTORE# NOWE_ZNAKI

TRAP# [T.#]

TRAP# nazwa – odpowiednik polecenia TRAP numer_linii w Atari BASIC-u.

•► TRAP# DALEJ

Operacje na pamięci

Uwaga. Przy żadnej z operacji na pamięci TBXL nie sprawdza sposobu adresowania. Błędne użycie większości tych poleceń może doprowadzić do zawieszenia się komputera i konieczności ponownego jego restartu – w efekcie utraty zawartości pamięci RAM.

BGET

BGET #n,adres,ilość – pobiera z kanału (o numerze *n*) ciąg bajtów i zapisuje go w pamięci od wskazanego *adresu*. *Ilość* określa liczbę bajtów jaka ma być pobrana i zapisana. Przed użyciem BGET należy pamiętać o otwarciu do odczytu jednego z bloków IOCB.

•► BGET #1,1536,256
•► BGET #5,DPEEK(88),960

BPUT

BPUT #n,adres,ilość – przesyła z pamięci komputera do wybranego kanału *n* ciąg bajtów. *Adres* wskazuje miejsce w pamięci od którego ma rozpocząć się przesyłanie, a *ilość* informuje ile bajtów ma być przesłanych. Przed użyciem BPUT należy pamiętać o otwarciu do zapisu jednego z bloków IOCB.

•► BPUT #5,DPEEK(88),960

```
800 OPEN #5,8,%0,"D:EKRAN1.SCR"
810 BPUT #5,DPEEK(88),960
820 CLOSE #5
```

Tu zostanie zapisany na dyskietkę ekran w grafice 0.

DPEEK

DPEEK(wyrażenie) – odczytuje wartość słowa spod adresu pamięci określonego przez *wyrażenie*. Słowo, czyli dwa kolejne bajty odczytywane są w kolejności młodszy bajt - starszy bajt według wzoru: *wyrażenie* + 256 * (*wyrażenie* + 1). Efektem jest liczba całkowita z przedziału 0 ÷ 65535. *Wyrażenie* powinno mieścić się w zakresie liczb 0 ÷ 65534.

•► PRINT DPEEK(88)
•► DL=DPEEK(560)

DPOKE [DP.]

DPOKE wyrażenie1,wyrażenie2 – dwubajtowe POKE. Zapisuje w dwóch kolejnych komórkach pamięci, począwszy od adresu wskazanego przez *wyrażenie1*, wartość *wyrażenia2*. Kolejność zapisu jest następująca: *wyrażenie1* – młodszy bajt, *wyrażenie1*+1 – starszy bajt. *Wyrażenie1* musi mieścić się w zakresie liczb dodatnich od 0 do 65534, *wyrażenie2* od 0 do 65535.

•► DPOKE 560,1536
•► DPOKE 45056,AD

%GET

%GET #n,zmienna_liczbowa – pobiera z urządzenia zewnętrznego (stacja dysków, magnetofon) liczbę w sześciobajtowym formacie dziesiętnym (BCD) i umieszcza w *zmiennej_liczbowej*. Numer kanału z którego %GET pobierze dane określa parametr *n* (1 ÷ 7). Oczywiście kanał ten musi być wcześniej otwarty do odczytu poleceniem OPEN. Patrz też %PUT, tam również przykład użycia %GET.

•► %GET #1,X
•► %GET #1,A,B,C

%PUT

%PUT #n,wyrażenie_liczbowe – zapisuje na urządzeniu zewnętrznym wartość *wyrażenia_liczbowego* w sześciobajtowym formacie dziesiętnym (BCD). Odwrotność polecenia %GET. Wcześniej należy poleceniem OPEN otworzyć kanał do zapisu, numer kanału (1 ÷ 7) reprezentuje *n*.

•► %PUT #7,A
•► %PUT #3,TX,POZIOM,WYNIK

```
100 A=1536:B=-48.125
110 OPEN #1,8,0,"D:TEST"
120 %PUT #1,A,B
130 ? A,B
140 CLOSE :A=0:B=0
150 ? A,B
160 OPEN #1,4,0,"D:TEST"
170 %GET #1,A,B
180 CLOSE
190 ? A,B
200 END
```

Oto efekt działania powyższego programu:

```
1536      -48.125
0         0
1536      -48.125
```

MOVE [M.]

MOVE adres_źródłowy,adres_docelowy,ile – kopiuje obszar pamięci zaczynający się od *adresu_źródłowego* w obszar zaczynający się od *adresu_docelowego*. *Ile* określa ilość przemieszczanych bajtów.

•► MOVE 48192,44096,40*24
•► MOVE DPEEK(560),DL,64

-MOVE

-MOVE adres_źródłowy,adres_docelowy,ile – to samo co MOVE, ale z jedną różnicą. Gdy *adres_docelowy* jest mniejszy niż *adres_źródłowy* + *ile* nastąpi nadpisanie danych. -MOVE przesuwa blok danych od końca, unikając w ten sposób nadpisania.

•► -MOVE 1540,1560,120

Grafika

CIRCLE [CI.]

CIRCLE X,Y,R1,R2 – rysuje okrąg lub elipsę. Parametry *X* i *Y* wyznaczają punkt środkowy rysowanego obiektu. *R1* to pozioma oś elipsy, a *R2* – pionowa. Parametr *R2* można pominąć, wówczas domyślnie R2=R1, czyli otrzymamy okrąg.

•► CIRCLE 40,65,16,8
•► CIRCLE 60,33,10
•► CIRCLE P,P*2,15

CLS

CLS – bez parametrów – czyści ekran w trybie tekstowym (GRAPHICS 0) oraz okna tekstowe w trybach graficznych, po czym ustawia kursor w górnym lewym rogu. W Atari BASIC-u ten efekt osiąga się poleceniem PRINT CHR$(125).

CLS#n – czyści ekran obrazu utworzony przez blok IOCB o numerze „n” (1 ÷ 7). CLS#6 kasuje zawartość ekranu utworzonego w standardowych trybach graficznych.

FCOLOR [FC.]

FCOLOR wyrażenie_liczbowe – ustala, którego rejestru koloru używać będzie polecenie FILLTO.

•► FCOLOR 1
•► FCOLOR A

FILLTO [FI.]

FILLTO X,Y – wypełnia „czysty” obszar ekranu, w prawo od rysowanego punktu. *X* to pozioma pozycja na ekranie, a *Y* pionowa. Kolor wypełniania ustalany jest poleceniem FCOLOR.
W Atari BASIC-u odpowiednikiem FILLTO jest: POSITION X,Y:XIO 18,#6,0,0,"S:".

•► FILLTO 42,V

Użycie tego polecenia wymaga wcześniejszych przygotowań (patrz przykład niżej):

- należy narysować granicę wypełniania z prawej strony ekranu (linie 20 ÷ 30);
- poleceniem PLOT lub DRAWTO ustawić punkt od którego rozpocznie się wypełnianie (linia 40);
- wpisać do polecenia FILLTO parametry punktu do którego ma być realizowane wypełnienie (linia 50).

```
10 GRAPHICS 7
20 COLOR %2:PLOT 100,30
30 DRAWTO 85,15:DRAWTO 100,%0
40 PLOT 10,%0
50 FCOLOR %1:FILLTO 10,30
60 END
```

PAINT

PAINT X,Y – wypełnia pusty obszar wewnątrz figury (lub taki obszar na zewnątrz figury). *X* i *Y* ustala punkt ekranu, odpowiednio współrzędną poziomą i pionową, od którego rozpocznie się wypełnianie. Wypełnianie następuje w poziomie i pionie, ale nie ukośnie. Rejestr koloru, jaki zostanie użyty do wypełnienia, wybierany jest poleceniem COLOR.

•► PAINT 47,100
•► PAINT HPZ,VPZ

Dźwięk

DSOUND [DS.]

DSOUND – użyte bez parametrów zamyka wszystkie kanały dźwiękowe. Taki sam efekt uzyskuje poleceniem SOUND bez parametrów. W Atari BASIC-u musi to być realizowane w następujący sposób:

```
100 FOR X=0 TO 3
110 SOUND X,0,0,0
120 NEXT X
```

Oczywiście SOUND z parametrami działa tak samo jak w Atari BASIC-u.

DSOUND głos,częstotliwość,zniekształcenia,głośność – łączy dwa generatory dźwięku POKEY-a w jeden. *Głos*, to numer tak utworzonego generatora dźwięku. Dopuszczalne zakresy poszczególnych parametrów to:

- głos: 0 lub 1;
- częstotliwość: 0 ÷ 65535;
- zniekształcenia: 0 ÷ 14, tylko liczby parzyste. Czysty dźwięk uzyskuje się przez wpisanie tu wartości 10 lub 14;
- głośność: 0 ÷ 15.

•► DSOUND 0,F,14,L

SOUND [SO.]

- Patrz DSOUND.

Funkcje arytmetyczne i logiczne

%0 %1 %2 %3

%0, **%1**, **%2**, **%3** – są to cztery stałe liczbowe odpowiadające odpowiednio liczbom 0, 1, 2 i 3. W przeciwieństwie do liczb zmiennoprzecinkowych, które zawsze zajmują 7 bajtów pamięci, stałe liczbowe zajmują tylko 1 bajt.

Warto ich używać jak najczęściej, gdyż pozwalają znacząco zaoszczędzić pamięć RAM, zwłaszcza przy dłuższych programach. Przykład użycia stałych liczbowych można zobaczyć w dodatku A.

DEC

DEC(wyrażenie_tekstowe) – dokonuje konwersji liczb szesnastkowych na dziesiętne. Przetwarzane są najwyżej pierwsze cztery znaki *wyrażenia_tekstowego*. Jeśli pierwszy znak nie jest liczbą szesnastkową, to konwersja nie jest wykonywana i w efekcie otrzymamy 0. Gdy *wyrażeniem_tekstowym* jest stała, a nie zmienna tekstowa, należy ją ująć w cudzysłów. Patrz też polecenie HEX$.

```
•►  PRINT DEC(A$)
•►  A=DEC("FF8A")
```

DIV

dzielna DIV dzielnik – dzielenie bez reszty; wynikiem będzie liczba całkowita powstała po odrzuceniu reszty z dzielenia. Np. 15 / 4 = 3.75, a 15 DIV 4 = 3. Dzielna i dzielnik może być stałą lub zmienną liczbową.

```
•► X=80 DIV 15
•► PRINT -465 DIV 16
```

EXOR

a EXOR b - alternatywa wykluczająca. W informatyce operację alternatywy wykluczającej stosuje się do par liczb naturalnych wykonując operacje na cyfrach zapisów binarnych tych liczb. Jest to zwykła logiczna alternatywa wykluczająca rozszerzona na ciągi bitów. Wykonuje się ją bit po bicie. Np. 7 EXOR 5:
00001112 EXOR 00001012 = 00000102 (efekt operacji na kolejnych cyfrach - liczby w systemie binarnym), a wynik w postaci dziesiętnej: = 2.
[za pl.wikipedia.org]
Parametr *a* i *b* może być całkowitą liczbą lub zmienną liczbową z zakresu 0 ÷ 65535). Warto zauważyć, że TBXL nie operuje na liczbach dwójkowych, dlatego wynik podawany jest w formacie dziesiętnym.

```
•► AW=65535 EXOR 251
```

FRAC

FRAC(wyrażenie_liczbowe) – zwraca część ułamkową *wyrażenia_liczbowego* poprzez odrzucenie wartości przed przecinkiem. Zostaje jednak zachowany znak liczby, np. wynikiem FRAC(-7.75) będzie liczba -0.75. Porównaj też polecenie TRUNC.

•► PRINT FRAC(18.8)
•► X=FRAC(SUMA)

HEX$

HEX$(wyrażenie liczbowe) – dokonuje konwersji całkowitej liczby dziesiętnej na szesnastkową. Przekształcana liczba musi zawierać się w zakresie 0 ÷ 65535. Patrz też polecenie DEC.

•► A$=HEX$(255)
•► PRINT HEX$(L)

MOD

a MOD b – polecenie to realizuje dzielenie modulo według wzoru: a-b*TRUNC(a/b). Parametry *a* i *b* mogą być liczbami lub zmiennymi liczbowymi.

•► PRINT 3.3 MOD 2.9

RAND

RAND(n) – generuje całkowitą liczbę losową z przedziału:
- 0 ÷ *n*–1, gdy *n*>0;
- -*n* ÷ -1, gdy n<0.

•► A=RAND(100)
•► A=RAND(-12)
•► PRINT RAND(Z)+25

RND

RND – to samo co RND(0) w Atari BASIC-u, ale w krótszej formie.

•► A=RND
•► PRINT RND

TIME

TIME – podaje trzybajtową liczbę stanu zegara czasu rzeczywistego (RTCLOCK). Wartość ta obliczana jest według wzoru: 65536*PEEK(18)+256*PEEK(19)+PEEK(20).

•► PRINT TIME

TIME$

**TIME$** - pokazuje czas w formacie „hhmmss” (godziny-minuty-sekundy). Aby ustawić czas należy użyć sekwencji: TIME$="hhmmss",gdzie pod „hh” postawiamy godzinę, pod „mm” minuty i pod „ss” sekundy. Zawsze musi to być 6 cyfr. Próba podstawienia mniejszej ilości spowoduje błąd. Podstawienie więcej niż sześciu cyfr, nie spowoduje błędu, ale znaki na pozycjach powyżej 6. zostaną zignorowane. Np. TIME$="12450020" spowoduje ustawienie godziny „124500”.

Uwaga. Zmienna TIME$ nie pokazuje dokładnego czasu, a to dlatego, że częstotliwość z jaką Atari generuje obraz nie wynosi dokładnie 50Hz.

```
•► PRINT TIME$
•► A$=TIME$

•► TIME$="221500"
•► TIME$=NOWY_CZA$
```

TRUNC

TRUNC(wyrażenie_liczbowe) – zwraca część całkowitą *wyrażenia_liczbowego* poprzez odrzucenie części ułamkowej. Zachowuje znak liczby, np. TRUNC(-9.998) to -9. Warto zwrócić uwagę na różnicę w stosunku do funkcji INT, która zaokrągla liczby „w dół”. Oto krótki program porównujący efekty działania tych funkcji:

```
100 ? "--------------------------"
105 ? "LICZBA     TRUNC      INT"
110 ? "--------------------------"
115 FOR PE=%0 TO %3
120   READ A
125   PRINT A,TRUNC(A),INT(A)
130   A=ABS(A)
135   PRINT A,TRUNC(A),INT(A)
140 NEXT PE
145 DATA -7.998,-7.012,-0.125,-6858.5
```

Porównaj też z poleceniem FRAC.

!

a!b – alternatywa. Wynikiem jest 0 (fałsz) gdy te same bity w wyrażeniu *a* i *b* mają wartość 0. W innym przypadku = 1 (prawda). Wynik podawany jest w formacie dziesiętnym. Np. 8!12 = 12. Dopuszczalny zakres liczb całkowitych to 0 ÷ 65535. Oba parametry (*a* i *b*) mogą być zarówno stałymi jak i zmiennym liczbowymi.

```
•► PRINT AB!CD
```

$

$aaaa – TBXL pozwala na umieszczanie w programie liczb szesnastkowych. Aby interpreter mógł je odróżnić od dziesiętnych, muszą być poprzedzone znakiem „$”. Liczby te muszą mieścić się w przedziale 0 ÷ 65535 ($0 ÷ $FFFF), a więc mogą składać się maksymalnie z 4 znaków.

```
•► $4F
•► $A8B
•► $ED0A
```

&

a&b – koniunkcja. Wynikiem jest 1 (prawda) tylko gdy te same bity w wyrażeniu *a* i *b* mają wartość 1, inaczej = 0 (fałsz). Wynik podawany jest w formacie dziesiętnym. Np. (2*2+7)&(4+7)=11, a 255&254=254. Dopuszczalny zakres liczb całkowitych to 0 ÷ 65535. Oba parametry (*a* i *b*) mogą być zarówno stałymi jak i zmiennym liczbowymi.

```
•► PRINT X&255
•► K=(L1*2)&(L2/2)
```

Operacje tekstowe

INKEY$

**INKEY$** - gdy zostanie naciśnięty klawisz, INKEY$ zostanie przypisany odpowiedni znak ATASCII. Nie zatrzymuje programu. Gdy żaden klawisz nie został naciśnięty, otrzymujemy zbiór pusty - LEN(INKEY$)=0.

•► A$=INKEY$
•► KEY=ASC(INKEY$)

INSTR

INSTR(zmienna_tekstowa1,zmienna_tekstowa2,n) – poszukuje ciągu zawartego w *zmiennej_tekstowej2* (krótszej) w *zmiennej_tekstowej1* (dłuższej). Parametr *n* określa pozycję od jakiej ma się rozpocząć przeszukiwanie *zmiennej_tekstowej1*. Jeżeli przeszukiwanie ma rozpocząć się od początku (n=1) parametr ten można pominąć. Po znalezieniu ciągu, INSTR zwraca pozycję pierwszego jego znaku w *zmiennej_tekstowej_1*. Jeśli ciąg nie zostanie odnaleziony wynikiem będzie zero.
Porównaj z UINSTR – tam też przykładowy program.

•► X=INSTR(A$,B$)
•► X=INSTR(A$,"Test",12)

UINSTR

UINSTR(zmienna_tekstowa1,zmienna_tekstowa2,n) – to samo co INSTR, z jedną różnicą: bit 5 i 7 każdego znaku jest ignorowany. Umożliwia to wyszukiwanie tekstu bez względu czy został napisany literami małymi, dużymi, czy w inwersie.

•► PRINT UINSTR(A$,B$)
•► PRINT UINSTR(A$,B$,15)

```
100 DIM A$(100)
110 A$(88)="Test---TEST"
120 ? INSTR(A$,"TEST",60),
130 ? INSTR(A$,"Test",60),
140 ? UINSTR(A$,"test",60),
150 ? UINSTR(A$,"TEST",60)
```

TEXT

TEXT X,Y,ciąg – umieszcza tekst na ekranie graficznym. Odpowiednik PRINT w trybie tekstowym. *X* i *Y* to współrzędne górnego lewego rogu pierwszego znaku parametru *ciąg*, który może być stałą lub zmienną, tekstową lub liczbową. W odróżnieniu od PRINT, po napotkaniu prawego marginesu wyświetlany tekst jest „ucinany”, a nie przenoszony do następnej linii. Wyboru koloru tekstu dokonuje się poleceniem COLOR.

•► TEXT 0,5,"NAPIS"
•► TEXT A,B,A$
•► TEXT 64,88,NR
•► TEXT H+X,V,120

```
10 GRAPHICS 5
100 COLOR %1:TEXT %0,%0,"Atari"
110 COLOR %2:TEXT 48,%0,65
120 COLOR %3:TEXT 64,%0,"XE"
130 END
```

Sterowanie przebiegiem programu

*B

***B +** - naciśnięcie BREAK wywoła błąd, który może być przechwycony przez TRAP.

***B –** - klawisz BREAK działa normalnie - naciśnięcie zatrzymuje program.

Użycie tego polecenia bez parametru („+”, „-”) jest równoznaczne z *B +.
Po uruchomieniu interpretera lub wykonaniu RUN ustawiany jest tryb *B –.

EXIT

W Atari BASIC-u polecenie POP (łącznie z GOTO) służy do przerywania pętli FOR-NEXT lub podprogramu GOSUB-RETURN. W TBXL można to zrobić prościej.

EXIT - natychmiastowe opuszczenie dowolnej pętli. Po napotkaniu wewnątrz pętli polecenia EXIT, jej wykonywanie zostaje natychmiast przerwane bez względu na stan warunku pętli i zostaje wykonany skok do następnego polecenia po zamykającym pętlę.

Uwagi.

- EXIT jest jedynym dopuszczalnym w programowaniu strukturalnym sposobem opuszczenia pętli bez końca (DO-LOOP).
- EXIT można użyć również do opuszczenia procedury (PROC-ENDPROC), jednak nie jest oto rozwiązanie zalecane, gdyż pogarsza czytelność programu. Poza tym zawsze można znaleźć sposób opuszczenia procedury przez ENDPROC.
- EXIT nie umożliwia skoku w dowolne miejsce programu, dlatego nie w pełni może zastąpić polecenie POP:GO#nazwa (POP:GOTO nr_linii).

```
10 ? "NACISNIJ START"
100 DO
110   IF PEEK(53279)=6 THEN EXIT
120 LOOP:? "PETLA ROZERWANA!"
130 END
```

*F

Polecenie to dotyczy pętli FOR-NEXT. W Atari BASIC-u pętla ta jest zawsze wykonywana co najmniej jeden raz. Dzieje się tak, gdyż sprawdzenie czy warunek został spełniony do opuszczenia pętli następuje na końcu, czyli po instrukcji NEXT. W TBXL można to zmienić tak, że warunek będzie sprawdzany na początku pętli, przed pierwszym jej obiegiem. Wówczas może się zdarzyć, że pętla FOR-NEXT nie zostanie wykonana ani razu.

***F +** - warunek sprawdzany jest na początku pętli.

***F –** - warunek sprawdzany jest na końcu pętli, czyli jak w Atari BASIC-u.

Użycie tego polecenia bez parametru („+”, „-”) jest równoznaczne z *F +. Po uruchomieniu interpretera lub wykonaniu RUN ustawiany jest tryb *F –.

```
100 P1=%0:P2=%1:? "*F -",P1,P2
110 *F -:EXEC PETLA_TESTOWA
120 ? "*F +",P1,P2
130 *F +:EXEC PETLA_TESTOWA
200 P1=%1:P2=%0:? "*F -",P1,P2
210 *F -:EXEC PETLA_TESTOWA
220 ? "*F +",P1,P2
230 *F +:EXEC PETLA_TESTOWA
390 ------------------------------
400 ? :? "KONIEC TESTU":END
410 ------------------------------
500 PROC PETLA_TESTOWA
510   FOR PE=P1 TO P2
520     ? "WEWNATRZ PETLI"
530   NEXT PE
540   ? "NA ZEWNATRZ PETLI"
550   ? "**********"
560 ENDPROC
```

Ten program ukazuje różnicę pomiędzy *F + a *F –. Po jego uruchomieniu zobaczymy:

```
*F -      0         1
 WEWNATRZ PETLI
 WEWNATRZ PETLI
 NA ZEWNATRZ PETLI
 **********
 *F +      0         1
 WEWNATRZ PETLI
 WEWNATRZ PETLI
 NA ZEWNATRZ PETLI
 **********
 *F -      1         0
 WEWNATRZ PETLI
 NA ZEWNATRZ PETLI
 **********
 *F +      1         0
 NA ZEWNATRZ PETLI
 **********

 KONIEC TESTU
```

PAUSE [PA.]

PAUSE n – zatrzymuje wykonywanie programu na czas określony parametrem *n*. Jednostką jest tu 1/50 sekundy, czyli n = 50 spowoduje zatrzymanie programu na jedną sekundę.

•► PAUSE 250
•► PAUSE ABC

Jeśli jesteś użytkownikiem Atari NTSC 60Hz (USA) to zauważ, że w takim przypadku jednostką jest 1/60 sekundy, czyli PAUSE 60 to jedna sekunda.

Edycja programu

Uwaga: polecenia podkreślone nie są akceptowane przez *Turbo BASIC Compiler*, a więc nie mogą być używane w programie przeznaczonym do kompilacji.

--

-- - specjalna instrukcja REM. Dwa myślniki użyte zaraz po numerze linii wyświetlane są w czasie listowania programu jako ciąg 30 znaków „-”. Jakikolwiek tekst umieszczony po „--” zostanie zignorowany.

Jest to doskonały sposób rozdzielania logicznych bloków programu, np. procedur. Zwiększa znacznie czytelność programu. Ponadto „--” zajmuje jeden bajt pamięci mniej niż same REM bez tekstu.

•► 300 --

Przykład użycia można zobaczyć przy poleceniu *L (niżej) i w załączniku A.

DEL

DEL od_nr_linii,do_nr_linii – usuwa linie programu począwszy *od_nr_linii*, a skończywszy na *do_nr_linii*.

Jeśli *od_nr_linii* nie istnieje, jako pierwsza usunięta będzie najbliższa linia o numerze większym od wskazanego.

Jeśli *do_nr_linii* nie istnieje, jako ostatnia usunięta będzie najbliższa linia o numerze mniejszym od wskazanego.

W przykładzie poniżej zostanie usunięta linia 11550, linia 12000 oraz wszystkie linie pomiędzy nimi.

•► DEL 11550,12000

DUMP

DUMP urządzenie – wysyła na wskazane *urządzenie* (np. drukarkę) listę zmiennych użytych w programie. Jeśli *urządzenie* zostanie pominięte, lista zostanie wyświetlona na ekranie. Zmienne na liście wyświetlane są w kolejności ich pojawiania się w programie.

•► DUMP
•► DUMP "P:"

Opis listy:

A =100	zmienna liczbowa
B(10,1	tablica liczbowa [DIM B(9)]
C(0,0	tablica niezwymiarowana
D(10,10	tablica liczbowa wielowymiarowa [DIM D(9,9)]
E$ 10,20	zmienna tekstowa
F$ 0,0	zmienna tekstowa niezwymiarowana
G$ 0,10	DIM G$(10), LEN(G$)=0
H PROC 1100	procedura o nazwie H zaczynająca się od linii 1100
I # 120	etykieta I w linii 120
J ?	niezdefiniowana etykieta lub procedura. Przyczyną może być wprowadzanie etykiety lub procedury, a potem jej usunięcie lub zmiana nazwy.

Przykładowa lista:

```
DL =36918
Y =185
ZN$ 0,10
P1$ 39,61
P2$ 0,5
KOLOR( 2,1
PR =20901
B =127
X =38
A =148
PRZESTAW PROC 680
WYNIK_GRY PROC 775
```

Dygresja. Aby usunąć z listy niepotrzebne zmienne, procedury i etykiety „zaśmiecające” nasz program należy kolejno: zapisać program poleceniem LIST, wykonać NEW i wczytać program poleceniem ENTER.

*L

***L +** - podczas listowania programu (patrz polecenie LIST) włącza tabulację, w efekcie linie programu objęte pętlą oraz wewnątrz procedury będą „wcięte” o dwie pozycje poziome. Dzięki temu kod programu staje się czytelniejszy. Użycie tego polecenia bez parametru („+”, „-”) jest równoznaczne z *L +. Po uruchomieniu interpretera tabulacja jest włączona.

***L –** - wyłącza tabulację.

Oto fragment programu, najpierw wylistowany w trybie *L +, a następnie w trybie *L -:

```
150 ------------------------------
160 FOR Y=B TO B+280 STEP 40
165   READ P1$:MOVE PR,Y,28
170   READ P1$:MOVE PR,486+Y,16
175   READ P1$:MOVE PR,834+Y,40
180 NEXT Y:COLOR %3
185 FOR Y=42 TO 176 STEP 11
190   READ P1$:TEXT 4,Y,P1$(%2)
195 NEXT Y
200 ------------------------------
```

```
150 --
160 FOR Y=B TO B+280 STEP 40
165 READ P1$:MOVE PR,Y,28
170 READ P1$:MOVE PR,486+Y,16
175 READ P1$:MOVE PR,834+Y,40
180 NEXT Y:COLOR %3
185 FOR Y=42 TO 176 STEP 11
190 READ P1$:TEXT 4,Y,P1$(%2)
195 NEXT Y
200 --
```

LIST [L.]

Dopuszczalne są następujące konfiguracje:

LIST – bez parametrów wyświetla na ekranie listing całego programu.

LIST od_nr_linii,do_nr_linii – wyświetla wybrany fragment listingu.

•► LIST 100,500

LIST od_nr_linii, – wyświetla listing *od_nr_linii* do końca programu.

•► LIST 100,

LIST nr_linii – wyświetla linię programu o podanym numerze.

•► LIST 755

Ponadto polecenie LIST działa tak samo jak w Atari BASIC-u.

ERL

ERL – podaje numer linii, w której nastąpiło przerwanie programu np. z powodu wystąpienia błędu. Słowo zawierające ten numer znajduje się pod adresem 186 {DPEEK(186)}. Patrz też ERR.

•► PRINT ERL

ERR

ERR – podaje kod błędu, który wystąpił w trakcie wykonywania programu. Kod ten odczytywany jest z komórki pamięci 195. Patrz też ERL.

```
10 DIM L$(%2):POKE 195,%0
20 INPUT "PODAJ LICZBE (0-99):",L$
30 TRAP #ZLE
40 L=VAL(L$)
50 ? "TWOJA LICZBA/MOJA LICZBA: ";
60 ? L;"/";RAND(100)
100 # ZLE
110 ? "ERR=";ERR,"ERL=";ERL
120 END
```

Jeśli postąpimy zgodnie ze wskazówkami w programie to otrzymamy podobny do tego efekt (proszę zwrócić uwagę na komunikat „ERR” i „ERL”):

```
PODAJ LICZBE (0-99):88
TWOJA LICZBA/MOJA LICZBA: 88/72
ERR=0     ERL=0
```

A teraz jeszcze raz uruchamiamy program i zamiast liczby wciskamy dowolny inny klawisz:

```
PODAJ LICZBE (0-99):Q
ERR=18    ERL=40
```

RENUM

RENUM stary_nr,nowy_nr,skok – zmienia numery linii programu. Linii o numerze *stary_nr* nadaje numer wpisany do *nowy_nr*. Następnej linii nadaje numer *nowy_nr* + *skok* i tak dalej do końca programu.

•► RENUM 200,3500,5

Oto efekt użycia RENUM z parametrami: *stary_nr* = 20, *nowy_nr* = 1000 i *skok* = 10, na przykładzie programu zamieszczonego przy opisie polecenia ERR:

```
10 DIM L$(%2):POKE 195,%0
1000 INPUT "PODAJ LICZBE (0-99):",L$
1010 TRAP #ZLE
1020 L=VAL(L$)
1030 ? "TWOJA LICZBA/MOJA LICZBA: ";
1040 ? L;"/";RAND(100)
1050 # ZLE
1060 ? "ERR=";ERR,"ERL=";ERL
1070 END
```

Uwagi.

- Program przed wykonaniem renumeracji należy dla bezpieczeństwa zapisać na dysku lub innym urządzeniu zewnętrznym na wypadek niepowodzenia operacji.
- RENUM zmienia odpowiednio wszystkie rozkazy skoku (GOTO, GOSUB, TRAP, RESTORE, ON GOTO, ON GOSUB) odwołujące się do numeru linii.
- Gdy odwołanie jest zmienną (np. GOTO A) lub wyrażeniem matematycznym zaczynającym się od zmiennej (np. GOTO A*2+100), to RENUM nie będzie w stanie obliczyć nowego adresu skoku i takie polecenie nie zostanie zmienione.
- Polecenie skoku zaczynające się od stałej liczbowej (np. GOTO 1000+A*10) będzie potraktowane jak GOTO stała_liczbowa (w przykładzie GOTO 1000).
- Gdy polecenie skoku odwołuje się do nie istniejącej linii, wówczas adresowi skoku zostanie przypisana wartość ujemna, np. GOTO 500 po renumeracji przyjmie wartość GOTO -500.
- Jeśli przenumerowujemy część programu należy uważać, aby *nowy_nr* nie był mniejszy od numeru ostatniej linii, która nie będzie zmieniana. Grozi to trwałym uszkodzeniem programu. Dla programu przykładowego (wyżej) RENUM 1050,200,10 skończy się fatalnie. Choć program pozornie działa, to podanie błędnego parametru zamiast przechwycenia błędu przez TRAP, spowoduje przerwanie programu zakończone komunikatem interpretera: „ERROR- 12 ?LINE AT LINE 1020”.

TRACE

TRACE + - włącza śledzenie wykonywania programu. Interpreter będzie pokazywał na ekranie w nawiasach kwadratowych numery linii, które są właśnie wykonywane. Może to ułatwić odnalezienie błędu w programie.

TRACE – - wyłącza śledzenie.

Dopiszmy do przykładu z polecenia RENUM linię:
1025 TRACE +
- oto co możemy zobaczyć po jego uruchomieniu:

```
 PODAJ LICZBE (0-99):67
 [1030]TWOJA LICZBA/MOJA LICZBA: [1040]67/90
 [1050][1060]ERR=0     ERL=0
 [1070]
```

Uwagi.

- Użycie tego polecenia bez parametru („+”, „-”) jest równoznaczne z TRACE +.
- Numery linii zawierające PROC, #, GO#, EXEC nie będą wyświetlane.
- Śledzenie jest wyłączane jeśli program zostanie przerwany z powodu błędu.

Usprawnienia

CLOSE [CL.]

CLOSE użyte bez parametrów zamyka wszystkie kanały IOCB o numerach 1 ÷ 7. To znacznie krócej niż w Atari BASIC-u: FOR I=1 TO 7:CLOSE #I:NEXT I.

DIM

W przeciwieństwie do Atari BASIC-a automatycznie zeruje wszystkie elementy tablic liczbowych i zmiennych tekstowych, jednak długość tych ostatnich nie ulega zmianie (początkowo 0).

GET

GET zmienna_liczbowa – oczekuje na naciśnięcie klawisza, po czym przypisuje *zmiennej_liczbowej* kod ATASCII naciśniętego klawisza. Używa IOCB nr 7. W Atari BASIC-u odpowiednikiem tego polecenia jest ciąg instrukcji:
OPEN #7,4,0,"K:":GET #7,KEY:CLOSE #7

•► GET A

INPUT [I.]

INPUT umożliwia teraz wyświetlenie tekstu przed wprowadzaniem danych. Nie ma więc potrzeby stosowania dodatkowo polecenia PRINT. Ponadto zastąpienie średnika przecinkiem eliminuje znak zapytania.

```
•► INPUT"Tu jest tekst ",A$
•► INPUT"Podaj nazwisko i wiek ";A$,X
•► INPUT" ",A
```

Po uruchomieniu pierwszego przykładu zobaczymy: „Tu jest tekst ”, a drugiego - „Podaj nazwisko i wiek ?”. W trzecim przykładzie nie ujrzymy ani tekstu, ani nawet znaku zapytania. Pojawi się tylko kursor.

PUT

PUT wyrażenie_liczbowe – wysyła na ekran znak o kodzie ATASCII reprezentowany przez *wyrażenie_liczbowe*. Znak wyświetlany jest na pozycji kursora. Odpowiada poleceniu PRINT CHR$(*wyrażenie_liczbowe*).

```
•► PUT 65
•► PUT LITERA
```

""

"" - podwójny cudzysłów umieszczony w ciągu tekstu wyświetlany jest jak zwykły cudzysłów.

•► PRINT "To jest ""prawdziwy"" cytat."

Na ekranie zobaczymy: To jest "prawdziwy" cytat.
Zastępuje taką konstrukcję w Atari BASIC-u:
PRINT "To jest ";CHR$(34);"prawdziwy";CHR$(34);" cytat."

_

_ - znak podkreślenia może być używany w nazwach procedur, etykiet i zmiennych, w tym tabel.

•► NOWA_ZMIENNA=100

Użycie IF-ELSE-ENDIF wraz z GO#

Dopuszczalne jest użycie polecenia skoku bezwarunkowego (GOTO, GO#) ze „środka” polecenia IF-ELSE-ENDIF.

```
100 IF A=100
110   PRINT "DOSKONALE"
120   GO# ZAPISZ
130 ELSE
140   PRINT "TAK SOBIE"
150   GO# OD_NOWA
160 ENDIF
```

AUTORUN.BAS

Zaraz po starcie systemu TBXL poszukuje na dysku pliku o nazwie AUTORUN.BAS. Jeżeli taki odnajdzie, wczyta go i uruchomi. Ta cecha może być bardzo użyteczna np. do automatycznego uruchamiania gier.

Dodatek A. Przykład programu w TBXL

Prezentowana tu jako przykład „Gra logiczna” jest nieco zmodyfikowaną wersją gry pod tym samym tytułem, zamieszczonej w „Tajemnicach Atari”, nr 4/1991.

```
1 ------------------------------
2 REM GRA LOGICZNA
3 REM (Turbo Basic XL)
4 REM Specjalnie dla Tajemnic Atari
5 REM Napisal Carampuc z grupy ASF
6 REM Gdansk, marzec 1991
7 REM Modyfikacja: Bluki, 24.09.2009
100 ------------------------------
110 ------------------------------
120 EXEC USTAWIENIA
130 EXEC TABLICE
140 REPEAT
150   EXEC ZMIENNE
160   EXEC LOSUJ
170   EXEC EKRAN
180   EXEC RUCH
190   EXEC SPYTAJ
200 UNTIL NOT KONIEC
210 # WYJSCIE:CLS #6
220 TEXT 120,60,"GAME OVER"
230 PAUSE 30
240 IF RUCH>%0 THEN RUN
250 TRAP #BLAD
260 POKE 566,146
270 RUN "D:AUTORUN.BAS"
280 # BLAD:END
290 ------------------------------
300 ------------------------------
1000 PROC TABLICE
1010   DIM TAB(15),T$(30),KEY$(%1)
1030 ENDPROC
1040 ------------------------------
1100 PROC ZMIENNE
1110   BOK=4:REM dlugosc boku
1120   KL=15:REM ile klockow do przesuwania
1130   RUCH=%0:REM ktory ruch
1140   PX=%0:PY=%0:REM pozycje (wzgledne) "kursora"
1150   DX=%0:DY=%0:REM przyrost wartosci pozycji
1160   XX=19:YY=20:REM wspol. lewego gornego rogu pola
1170 ENDPROC
1180 ------------------------------
1500 PROC LOSUJ
1520   DIM T2(15):REM tablica pomocn.
1530   FOR X=%1 TO KL
1540     T2(X)=%1
1550   NEXT X
1560   FOR X=%0 TO KL-%1
1570     REPEAT
1580       LOS=TRUNC(KL*RND)+%1
1590     UNTIL T2(LOS)=%1
1600     T2(LOS)=%0
1610     TAB(X)=LOS
```

```
1620   NEXT X
1630 ENDPROC
1640 -----------------------------
2000 PROC EKRAN
2020   GRAPHICS 8+16:SETCOLOR %2,%0,%0:COLOR %1
2030   PLOT %0,%0:DRAWTO 319,%0:DRAWTO 319,150:DRAWTO %0,150:DRAWTO %0,%0
2040   PLOT %2,%2:DRAWTO 317,%2:DRAWTO 317,148:DRAWTO %2,148:DRAWTO %2,%2
2050   WX=XX-%3:WY=YY-%3:DL=97:EXEC KWADRAT
2060   REM narysuj plansze
2070   FOR Y=%0 TO %3
2080     DL=22
2090     FOR X=%0 TO %3
2100       WX=XX+X*(DL+%1):WY=YY+Y*(DL+%1)
2110       EXEC KWADRAT
2120       WART=TAB(X+Y*4)
2130       IF WART THEN TEXT WX+4+4*(WART<10),WY+7,WART
2140     NEXT X
2150   NEXT Y
2160   PAINT 17,18
2170   WX=120:WY=20:POKE 756,156
2180   RESTORE #INSTR
2190   READ T$
2200   WHILE T$<>"#"
2210     TEXT WX,WY,T$
2220     WY=WY+8
2230     READ T$
2240   WEND
2250   TEXT 20,120,"RUCH:"
2260   TEXT 60,120,RUCH
2270   # INSTR
2280   DATA GRA LOGICZNA
2290   DATA Tajemnice Atari 4/1991
2300   DATA
2310   DATA Twoje zadanie polega na
2320   DATA u$o^eniu klock&w na pla-
2330   DATA nszy obok w kolejno'ci
2340   DATA rosn_cej. Mo^esz u^ywa%
2350   DATA klawiszy kursora i spa-
2360   DATA cji lub joysticka. ESC:
2370   DATA nowa gra lub wyj'cie.
2380   DATA
2390   DATA Powodzenia...
2400   DATA #
2410 ENDPROC
2420 -----------------------------
2500 PROC KWADRAT
2520   PLOT WX,WY:DRAWTO WX+DL,WY:DRAWTO WX+DL,WY+DL:DRAWTO WX,WY+DL:DRAWTO WX,WY
2530 ENDPROC
2540 -----------------------------
3000 PROC PRZESUN
3020   DX=PX+PY*4
3030   IF NOT (PY-%1<%0)
3040     IF TAB(DX-4)=%0
3050       DY=-4
3060       TAB(DX+DY)=TAB(DX)
3070       EXEC ZMIEN
3080     ENDIF
3090   ENDIF
3100   IF PY+%1<4
```

```
3110     IF TAB(DX+4)=%0
3120       DY=4
3130       TAB(DX+DY)=TAB(DX)
3140       EXEC ZMIEN
3150     ENDIF
3160   ENDIF
3170   IF NOT (PX-%1<0)
3180     IF TAB(DX-%1)=%0
3190       DY=-%1
3200       TAB(DX+DY)=TAB(DX)
3210       EXEC ZMIEN
3220     ENDIF
3230   ENDIF
3240   IF PX+%1<4
3250     IF TAB(DX+%1)=%0
3260       DY=%1
3270       TAB(DX+DY)=TAB(DX)
3280       EXEC ZMIEN
3290     ENDIF
3300   ENDIF
3310 ENDPROC
3320 ------------------------------
3500 PROC KLAWISZ
3530   POKE 764,255
3540   REPEAT
3550     KEY$=INKEY$:X=STICK(%0)
3560     ON KEY$=CHR$(27) GO# WYJSCIE
3570     DX=(KEY$="*")-(KEY$="+")+(X=7)-(X=11)
3580     DY=(KEY$="=")-(KEY$="-")+(X=13)-(X=14)
3590   UNTIL DX+DY OR KEY$=" " OR NOT STRIG(%0)
3600 ENDPROC
3610 ------------------------------
4000 PROC RYSUJ
4010   REM procedura rysuje "kursor"
4020   WX=XX+PX*23+%2:WY=YY+PY*23+%2:DL=18
4030   EXEC KWADRAT
4040 ENDPROC
4050 ------------------------------
4100 PROC SPRAWDZ
4130   LG=%1
4140   FOR X=%0 TO KL-%2
4150     LG=LG*((TAB(X+%1)-TAB(X))=%1)
4160   NEXT X
4170 ENDPROC
4180 ------------------------------
5000 PROC RUCH
5010   REM sterowanie gra
5020   PX=%0:PY=%0:EXEC RYSUJ
5030   REPEAT
5040     EXEC KLAWISZ
5050     IF DY+DX
5060       COLOR %0
5070       DL=18
5080       EXEC KWADRAT
5090       COLOR %1
5100       IF (PX+DX>=%0) AND (PX+DX<4) THEN PX=PX+DX
5110       IF (PY+DY>=%0) AND (PY+DY<4) THEN PY=PY+DY
5120       EXEC RYSUJ
5130     ELSE
```

```
5140       IF TAB(PX+PY*4)<>%0 THEN EXEC PRZESUN
5150     ENDIF
5160     EXEC SPRAWDZ
5170   UNTIL LG
5180   TEXT 40,130,"Koniec! Wreszcie tego dokona$e'!"
5190 ENDPROC
5200 ------------------------------
6000 PROC ZMIEN
6020   TAB(DX)=%0
6030   DL=23
6040   SX=XX+PX*DL+4
6050   SY=YY+PY*DL+7
6060   TEXT SX,SY,"  "
6070   X=PX+DY*(ABS(DY)=%1)
6080   Y=PY+SGN(DY*(ABS(DY)=4))
6090   SX=XX+X*DL+4
6100   SY=YY+Y*DL+7
6110   WART=TAB(X+Y*4)
6120   TEXT SX+4*(WART<10),SY,WART
6130   RUCH=RUCH+%1
6140   TEXT 60,120,RUCH
6150 ENDPROC
6160 ------------------------------
6500 PROC SPYTAJ
6510   TEXT 30,138,"Chcesz spr&bowa% jeszcze raz (T/N)?"
6520   REPEAT
6530     KEY$=INKEY$
6540   UNTIL KEY$="n" OR KEY$="N" OR KEY$="t" OR KEY$="T"
6550   KONIEC=(KEY$="t") OR (KEY$="T")
6560 ENDPROC
6570 ------------------------------
7000 PROC USTAWIENIA
7010   POKE 106,192:GRAPHICS %0
7020   POKE 559,%0:POKE 566,158
7030   LG=156*256:MOVE 57344,LG,1024
7040   RESTORE #ZNAKI
7050   FOR XX=%0 TO %1
7060     READ X,Y,SX
7070     MOVE X,LG+8*INT(Y*0.125),8
7080     POKE LG+Y,SX
7090   NEXT XX
7100   FOR XX=%0 TO %3
7110     READ X,Y
7120     MOVE X,LG+8*INT(Y*0.125),8
7130     READ SX,SY
7140     POKE LG+Y,SX:POKE LG+Y+%1,SY
7150   NEXT XX
7160   # ZNAKI
7170   DATA 58120,511,12,58320,497,24
7180   DATA 58208,35,28,56,58136,40,12,24
7190   DATA 58232,48,12,24,58264,56,12,24
7200 ENDPROC
7210 ------------------------------
```

Dodatek B. Kody błędów

Kod	Komunikat	Wyjaśnienie
2	MEM	Zabrakło wolnej pamięci RAM.
3	VALUE	Próba nadania zmiennej numerycznej wartości spoza dopuszczalnego (dla tej zmiennej) zakresu.
4	#VARS	Zbyt dużo zmiennych (maksymalnie 256).
5	$LEN	Przekroczony rozmiar zmiennej tekstowej. Np.: DIM A$(5):A$(5,7)="ABC"
6	?DATA	Zabrakło danych dla polecenia READ lub nie użyto RESTORE przy ponownej próbie odczytu.
7	>32767	Niepoprawny numer linii. Dopuszczalny przedział to 0÷32767.
8	INPUT	Nie można zmiennej liczbowej przyporządkować wartości nienumerycznej (np. litery).
9	DIM	- Niezadeklarowana zmienna, - próba powtórnego zadeklarowania tej samej zmiennej, - przekroczony wymiar DIM (32767 dla zmiennych tekstowych i 5460 dla tablic).
10	STACK	Przepełniony stos (użyto zbyt dużo nawiasów w wyrażeniu, zbyt dużo równocześnie otwartych pętli lub podprogramów [procedur]).
11	OVERFLOW	- Zbyt mała lub zbyt duża liczba (dopuszczalny zakres to 1E-99 do 1E+98). - Próba dzielenia przez zero (ponieważ dzielenie jest odwrotnością mnożenia, to nie istnieje liczba, która pomnożona przez zero da wynik inny niż zero).
12	?LINE	Nie ma numeru linii wskazanej w poleceniu skoku.
13	?FOR	Brakuje FOR do napotkanego polecenia NEXT.
14	TOO LONG	Zbyt długa linia programu (maksymalnie 128 bajtów).
15	?DEL	Po napotkaniu polecenia powrotnego z podprogramu (procedury) lub pętli program nie może odnaleźć polecenia wywołującego. TBXL pozwala na zatrzymanie (STOP) i edytowanie programu. Gdy wtedy zostanie usunięte polecenie wywołujące, to po wznowieniu (CONT) programu nastąpi błąd nr 15. Błąd może też wystąpić, gdy w programie użyto polecenia DEL.
16	?GOSUB	Napotkano RETURN, ale nie było odpowiedniego wywołania przez GOSUB.
17	GARBAGE	Błąd składni programu.
18	?CHR	Nieprawidłowy ciąg znaków. Np. pierwszym znakiem polecenia VAL nie jest cyfra.
19	MEM	Wczytywany do pamięci program jest zbyt długi (nie mieści się w dostępnej pamięci RAM).
20	#	Błędny numer urządzenia zewnętrznego – spoza przedziału 1÷7.
21	?LOAD	Użyto niewłaściwej instrukcji do wczytania programu. Np. LOAD do programu zapisanego przez LIST.

Kody błędów – ciąg dalszy tabeli.

22	?NEST	Niewłaściwe zagnieżdżenie. Pętle powinny być zamykane w odwrotnej kolejności do ich otwierania, tzn. pętla, która została otwarta jako ostatnia, musi być zamknięta jako pierwsza. Błąd może też wystąpić, gdy wewnątrz procedury (PROC) znajduje się otwarta pętla, a program napotkał polecenie ENDPROC.
23	?WHILE	Brakuje WHILE do napotkanego polecenia WEND.
24	?REPEAT	Brakuje REPEAT do napotkanego polecenia UNTIL.
25	?DO	Brakuje DO do napotkanego polecenia LOOP.
26	?EXIT	Niewłaściwe użycie polecenia EXIT, np. poza pętlą.
27	XPROC	Napotkano polecenie PROC bez wywołania przez EXEC.
28	?EXEC	Brakuje EXEC do napotkanego polecenia ENDPROC.
29	?PROC	Nie ma procedury o podanej nazwie (błędna nazwa).
30	#	Nie ma etykiety o podanej nazwie (błędna nazwa).
128		Użyto klawisza BREAK (błąd zwykle występuje podczas operacji WE/WY).

Kody błędów o numerach powyżej 127 powiązane są z DOS-em i urządzeniami zewnętrznymi, takimi jak stacja dysków czy drukarka. Gdy taki błąd wystąpi, zajrzyj do instrukcji obsługi używanego DOS-a lub określonego urządzenia zewnętrznego.

Dodatek C. Kody generatora znaków

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.

Dodatek D. Kody klawiszy (keycode)

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:

Dave Yearke, Laura Yearke. *TURBO BASIC COMMAND LIST*. Western New York Atari Users Group.

Frank Ostrowski. *Basic, schnell wie der Wind - mit dem Turbo-Basic XL – Interpreter*. „Happy Computer. 1 Atari Sonderheft”, 1985, s. 34-47.

Turbo-BASIC XL Referenz [online]. ABBUC e.V. [dostęp: 12.04.2013]. Dostępny w Internecie: http://www.abbuc.de/~atarixle/standard/turbobas.htm

TURBO BASIC INTERPRETER. ATARI 800XL/130XE. WERSJA DYSKOWA. [Brak danych o autorze i wydawnictwie].

Carampuc/ASF. *GRA LOGICZNA*. „Tajemnice Atari”, 4/1991, s. 12-15.

Podziękowania dla ***Larka*** i ***Sikora*** za udzieloną pomoc przy realizacji niniejszego podręcznika.