SkîadniaAR

Marek Pampuch 1992-3 and opracowaî Stanisîaw 1997

COLLABORATORS			
	TITLE : SkîadniaAR		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Marek Pampuch 1992-3 and opracowaî Stanisîaw 1997	February 24, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

SkîadniaAR

1.1 Spis treôci

FREESPACE FUZZ GETCLIP GETSPACE HASH IF
IMPORT INDEX INSERT INTERPRET ITERATE LASTPOS
LEAVE LEFT LENGTH LINES MAX MIN
NOP NUMERIC OPEN OPTIONS PRAGMA PROCEDURE
PULL PUSH QUEUE RANDOM RANDU READCH
READLN REMLIB RETURN REVERSE RIGHT SEEK
SELECT SETCLIP SHELL SHOW SIGN SIGNAL
SOURCELINE STORAGE STRIP SUBSTR SUBWORD SYMBOL
TIME TRACE TRACE TRANSLATE TRIM TRUNC
UPPER VALUE VERIFY WHEN WORD WORDINDEX
WORDLENGHT WORDS WRITECH WRITELN X2C X2D
XRANGE
R O Z K A Z Y
ALLOCMEM CLOSEPORT FREEMEM GETPKT OPENPORT
REPLY SHOWDIR SHOWLIST STATEF WAITPKT
R O Z K A Z Y U Û Y T K O W E (AREXX, SHELL)
HHU RX RXSET TC TG TT TU

1.2 OPERATORY

~ negacja (NOT)
& koniunkcja (AND)
Operatory | alternatywa (OR)
logiczne ^,&& alternatywa wykluczajâca jednoczesnâ prawdziwoôê (XOR)
i poîâczenia || poîâczenie dwóch îaïcuchów znakowych
[] poîâczenie dwóch îaïcuchów ze spacjâ pomiëdzy nimi
+ dodawanie
- odejmowanie
Operatory * mnoûenie
matematyczne ** potëgowanie
/ dzielenie
% dzielenie caîkowite
// reszta z dzielenia caîkowitego (modulo)
= sprawdzenie równoôci
~= sprawdzenie nierównoôci
Operatory == sprawdzenie identycznoôci
porównania ~== sprawdzenie nie-identycznoôci
> wiëksze
>=,~< wiëksze lub równe
< mniejsze
<=,~> mniejsze lub równe

1.3 abbrev

ABBREV (abbreviation - skrót) *** FUNKCJA
ABBREV <a$,b$,(d)>
(a$,b$ - to îaïcuchy, d-liczba)
W wyniku otrzymujemy wartoôê logicznâ (0-faîsz, 1-prawda) w
czaleûnoôci od tego, czy îaïcuch b$ jest skrótem îaïcucha a$, czy
nie. Jeûeli umieôcisz parametr d (defaultowo = 0) to funkcja bada
dodatkowo, czy dîugoôê îaïcucha b$ jest wiëksza lub równa od d. Przykîad:
SAY ABBREV (’Amiga’,’Ami’)
(da w wyniku 1),
SAY ABBREV (’Amiga’,’Ami’,4)
(da w wyniku 0),
SAY ABBREV (’Amiga’,"")
(w wyniku otrzymasz 0)

1.4 abs

ABS (absolute value -- wartoôê bezwzglëdna) *** FUNKCJA
ABS <n>
(gdzie n -- to liczba).
W wyniku otrzymasz wartoôê bezwzglëdnâ liczby n. Na przykîad:
SAY ABS (-5.4)
(da w wyniku 5),zaô
SAY ABS (17.5)
(da w wyniku 17).

1.5 addlib

ADDLIB (add library -- dodaj (tu: doîâcz) bibliotekë) *** FUNKCJA
ADDLIB <n$,p ,[o,w]>
gdzie n$ -- to îaïcuch, p -- liczba caîkowita z zakresu (-100, 100), zaô
o,w -- liczby caîkowite.
Doîâcza bibliotekë funkcji ARexxa do bibliotek rezydentnych. (W przypadku
jeôli chcesz korzystaê z nierezydentnych funkcji ARexxa -- musisz to
zrobiê). Przy okazji podawana jest wartoôê boole’owska -- wskazujâca, czy
doîâczenie sië powiodîo (1), czy nie (0). Îaïcuch n$ okreôla nazwë
biblioteki lub nazwë programu zarzâdzajâcego funkcjami. UWAGA: Moûna uûyê
tylko tych bibliotek, które wczeôniej zapiszesz w katalogu LIBS:. Parametr

p okreôla priorytet przeszukiwania. Parametrów o i w uûywaj tylko wówczas, gdy przyîâczasz biblioteki (o -- okreôla offset sprawdzajâcy adres wejôciowy biblioteki, w -- oznacza najniûszy dopuszczalny poziom wyjôcia. UWAGA: Doîâczonâ bibliotekë naleûy jeszcze otworzyê, funkcja ADDLIB sama tego nie robi. Przykîad:
SAY ADDLIB ("rexxsupport.library",0,-30,0)
(przy zaîoûeniu, úe biblioteka o nazwie "rexxsupport.library" znajduje sië w katalogu LIBS: -- w wyniku otrzymasz1 a biblioteka zostanie przyîâczona. Inny przykîad:
CALL ADDLIB "Novell",-20
(podîâczy bibliotekë zarzâdzajâcâ sieciâ -- jeûeli Twoja Amiga pracuje w sieci standardu Novell).
Przed opisem kolejnej instrukcji naleûy wyjaôniê pojëcie adresu zarzâdzajâcego. Jest to miejsce, do którego bëdâ przekazywane kolejne rozkazy. Domyôlnie ustawiony jest adres zarzâdzajâcy o nazwie REXX. Aby przekazywaê rozkazy bezpoôrednio do ADOS-u, musisz uûyê nazwy COMMAND. Jeôli spróbujesz przesîaê rozkaz pod nieistniejâcy adres zarzâdzajâcy -- wówczas zostanie wydrukowany komunikat o bîëdzie "Host environment not found" (ôrodowisko zarzâdzania nie znalezione). Aby przesyîaê rozkazy do innego programu -- musisz znaê nazwë portu komunikatów programu. W znalezieniu tej nazwy pomoûe Ci instrukcja ARrexxa o nazwie ADDRESS.

1.6 address

ADDRESS (adres) *** INSTRUKCJA
ADDRESS (a$|b$ ([VALUE] n])
gdzie: a$ -- to îaïcuch, b$ -- nazwa zmiennej, zaô n -- wyraûenie.
Instrukcja okreôla adres wystëpowania w programie zewnëtrznym rozkazów podawanych przez interpreter ARexxa (tzw. host address- adres zarzâdzajâcy). Moûesz mieê do czynienia z dwoma rodzajami adresu zarzâdzajâcego: poprzednim i obecnym. W momencie utworzenia kolejnego adresu zarzâdzajâcego -- adres obecny staje sië poprzednim, zaô poprzedni zostaje usuniëty z pamiëci. Istniejâ 4 gîówne formy instrukcji:
1) ADDRESS a$|b$ n
W tym przypadku wartoôê wyraûenia n zostanie obliczona i przekazana do adresu zarzâdzajâcego, podanego przez îaïcuch a$ lub zmiennâ b$. Nie zostanie utworzony nowy adres zarzâdzajâcy. Pozwala to na îatwe przekazanie do zewnëtrznego programu gîównego pojedynczego rozkazu interpretera -- bez naruszania dotychczasowej struktury. Przykîad:

ADDRESS edycja ’top’
(ustawi wskaúnik interpretera na poczâtku adresu zarzâdzajâcego o nazwie
"edycja").
2) ADDRESS a$|b$
Stworzy nazwë nowego adresu zarzâdzajâcego, którâ to nazwâ bëdzie îaïcuch
a$ lub zmienna b$. Adres poprzedni zostaje usuniëty z pamiëci, adres
obecny staje sië adresem poprzednim. Przykîad:
ADDRESS edycja1
(tworzy nowy adres zarzâdzajâcy o nazwie "edycja").
3) ADDRESS VALUE n
Przyjmuje etykietë, pod którâ bëdzie sië znajdowaê obliczony opcjâ 2 adres
zarzâdzajâcy. Przykîad:
ADDRESS VALUE EDYCJA IN
(przyjmie etykietë dla adresu zarzâdzajâcego edycjâ).
4) ADDRESS
Instrukcja podana bez parametrów -- zamienia adresy obecny i poprzedni --
bez naruszania struktury programu. Przykîad:
ADDRESS
(jeôli uûyîeô opcji 1-3, tak jak w poprzednich przykîadach, wówczas po
uûyciu opcji 4 adresem obecnym bëdzie REXX, zaô poprzednim EDYCJA).
ADDRESS (adres) *** FUNKCJA
ADDRESS <())
Podaje aktualnâ nazwë adresu zarzâdzajâcego. Przykîad:
SAY ADDRESS()
(wyôwietli na ekranie np. EDYCJA)

1.7 arg

ARG ([give number of] arg[ument string] -- podaj liczbë argumentów) *** FUNKCJA
ARG ([n,][EXISTS|OMMITED])
gdzie: n -- liczba, sîowa EXISTS i OMMITED moûna zastâpiê odpowiednio przez E i O.
Podaje iloôê argumentów zwiâzanych z aktualnym ôrodowiskiem programu.
Jeôli opuôcisz wszystkie parametry -- wówczas zostanie podana caîkowita
iloôê argumentów w ôrodowisku. Jeôli podasz tylko parametr n, wówczas
zostanie wypisany argument bëdâcy n-tym z kolei argumentem. W przypadku
podania obu parametrów -- jeôli istnieje parametr o podanym numerze n, a
uûyîeô sîowa EXISTS -- wówczas funkcja przybierze wartoôê 1 (prawda),
jeôli parametr nie istnieje -- wtedy funkcja przybierze wartoôê 0 (faîsz).
Rzecz jasna, w przypadku uûycia sîowa OMMITED (opuszczony) -- funkcja

przybierze wartoôci odwrotne. UWAGA: Jeôli przyjâîeô îaïcuch zerowy jako
argument, to mimo ûe taki argument istnieje -- zostanie potraktowany jako
nieistniejâcy. Przypuôêmy, ûe w ôrodowisku zaîoûyîeô nastëpujâce
argumenty: (’dwa’„5,’b’). Wówczas:
SAY ARG()
(da w wyniku 4),
SAY ARG(1)
(da w wyniku 2), zaô
SAY ARG(2,EXISTS)
(da w wyniku 0).
ARG ([parse upper] arg[ument string] -- przeanalizuj îaïcuch argumentów) *** INSTRUKCJA
[PARSE UPPER] ARG [a$],[b$, ... , z$]
gdzie: a$ -- z$ -- to szablony alfanumeryczne.
Instrukcja pozwala na uzyskanie jednego lub wiëcej
îaïcuchów argumentów. Jeôli uûyjesz sîów PARSE UPPER -- podstawione
zostanâ jedynie te argumenty, które napisaîeô duûymi literami. UWAGA:
Jeôli szablonem jest rozkaz -- w jednej instrukcji moûesz uûyê tylko
jednego parametru, w przypadku instrukcji -- moûesz ich uûyê nawet 15.
Instrukcja nie zmienia wartoôci îaïcucha szablonu. Jeôli na przykîad na liôcie
argumentów znajdujâ sië (’dwa’„0,’Marek’,’AMIGA’), wówczas:
ARG pierwszy, drugi, trzeci
(pobierze trzy argumenty z listy -- czyli: dwa, îaïcuch pusty i zero),
PARSE UPPER ARG pierwszy, drugi
(pobierze z listy M i Amiga).

1.8 b2c

B2C (b[inary string into character] c[onversion] -- przeksztaîcenie
îaïcucha dwójkowego na znakowy] *** FUNKCJA
B2C (a$)
gdzie a$ -- jest to îaïcuch alfanumeryczny, przedstawiajâcy kod ASCII znaku.
Funkcja jest odpowiednikiem BASIC-owej funkcji CHR$(a$). Przeksztaîca
îaïcuch "binarny" (zawierajâcy zera i jedynki) na znak, którego kod ASCII
równa sië liczbie, której kod przedstawia îaïcuch a$. UWAGA: Liczba musi
byê z zakresu 33 do 255, a zatem odpowiadajâcy jej îaïcuch z zakresu
00100001 do 11111111. Najczëstszym zastosowaniem jest maskowanie
bitów. Trochë skomplikowany opis rozjaôniâ przykîady:
SAY B2C(’00110011’)
(da w wyniku 3, bo 00110011 to binarnie 51, a kodowi ASCII=51 odpowiada
cyfra 3),
SAY B2C(’01100011’)
(da w wyniku c, bo 01100011 to binarnie 99, a kodowi ASCII=99 odpowiada
maîa litera "c").

1.9 bitand

BITAND (bit[by bit] AND -- wykonaj "i" logiczne) *** FUNKCJA
BITAND (a$,b$ [,p])
gdzie: a$,b$ -- to îaïcuchy alfanumeryczne, zaô p -- liczba.
Wykonuje dziaîanie logiczne "i" na îaïcuchach (w BASIC-u wyglâdaîoby to jak: a$ AND b$). Jeôli uûyjesz argumentu p, wówczas krótszy îaïcuch jest przesuniëty w prawo o p pozycji, zaô pozostaîe znaki îaïcucha dîuûszego sâ dopisywane na poczâtku îaïcucha wynikowego bez AND-owania. Jeôli nie uûyjesz parametru p -- operacja zakoïczy sië, gdy porównanie "dojdzie" do koïca krótszego îaïcucha. Reszta znaków dîuûszego îaïcucha zostanie wówczas dopisana do wyniku bez wykonywania na niej operacji logicznej.
Przykîad:
BITAND('0313'X,'FFFO'X)
(da w wyniku '0310'),
BITAND('00100100','0111')
(da w wyniku '00100100'), zaô
BITAND('00100100','0111',4)
(da w wyniku '00100100')

1.10 bitchg

BITCHG (bit ch[an]g[e] -- zamieï bit) *** FUNKCJA
BITCHG (a$,n)
gdzie:a$ -- to îaïcuch binarny lub heksadecymalny, zaô n -- liczba.
Zmienia stan bitu îaïcucha okreôlonego jako n-ty. UWAGA: bity îaïcucha liczone sâ od koïca. Pierwszy bit ma numer 0. Zmiana stanu polega na zapaleniu bitu zgaszonego lub zgaszeniu zapalonego (potocznie mówiâc -- gdy bit = 1 wówczas zmienia sië w 0 i na odwrót). Przykîad:
BITCHG('01101',3)
(da w wyniku îaïcuch '01001').

1.11 bitclr

BITCLR (bit cl[ea]r -- wyczyôê (tu:wyzeruj) bit) *** FUNKCJA
BITCLR (a$,n) gdzie: a$ -- îaïcuch binarny lub heksadecymalny, zaô n-liczba.
Gasi (ustawia na zero n-ty bit îaïcucha).
UWAGA: bity îaïcucha liczone sâ od koïca. Pierwszy bit ma numer 0. Przykîad:
BITCLR('0313'X,4)
(daje w wyniku '0303'. UWAGA: Wynik moûe Ci sië wydaê dziwny, ale nie zapominaj, ûe na îaïcuchach heksadecymalnych dziaîa sië nieco inaczej niû na binarnych. Musisz najpierw danâ liczbë przekształciê na binarnâ, a potem znaleúê -- i zgasiê -- wyspecyfikowany bit).

1.12 bitcomp

BITCOMP (bit [by bit] comp[are] -- porównaj bit po bicie) *** FUNKCJA
BITCOMP (a$,b$,[p])
gdzie: a$,b$ -- to îaïcuchy,zaô p -- liczba.
Porównuje îaïcuchy bit po bicie, poczynajâc od bitu nr 0 (na koïcu).
Parametr p -- przesuwa krótszy îaïcuch w prawo. Operacja koïczy sië w momencie napotkania, w którymô z îaïcuchów, znaku róûnego od wystëpujâcego na tej samej pozycji w drugim z îaïcuchów. Jako wartoôê funkcja przyjmuje numer bitu, na którym wystëpuje róûnica. Jeôli oba îaïcuchy sâ identyczne, wówczas funkcja przyjmuje wartoôê =-1. UWAGA: Îaïcuchy mogâ byê binarne lub heksadecymalne. Przykîad:
BITCOMP (’FF’X,’FF’X)
(przyjmie wartoôê -1), zaô
BITCOMP (’01000010’,’10000010)
(przyjmie wartoôê 7).

1.13 bitor

BITOR (bit [by bit] OR -- wykonaj "lub" logiczne) *** FUNKCJA
BITOR (a$,b$ [,p])
Znaczenie parametrów jest identyczne jak przy BITAND.
Wykonuje "lub" logiczne na îaïcuchach a$ i b$. Pozostaîe uwagi jak przy BITAND.
Przykîad:
BITOR(’0313’X,’003F’X)
(da w wyniku îaïcuch ’033F’),
BITOR (’01010101’,’1010’)
(da w wyniku îaïcuch ’11110101’), zaô
BITOR (’01010101’,’1010’,4)
(da w wyniku ’01011111’).

1.14 bitset

BITSET (ustaw bit) *** FUNKCJA
BITSET (a$,n)
gdzie: a$ -- to îaïcuch heksadecymalny lub binarny, zaô n -- liczba.
"Odwrotnoôê" funkcji BITCLR (oznacza to, úe "zapala" zgaszone bity). Przykîad:
BITSET(’0313’X,2)
(da w wyniku ’0317’X).

1.15 bittst

BITTST (bit t[e]st -- zbadaj bit) *** FUNKCJA

BITTST (a$,n)

(Oznaczenia parametrów jak przy funkcji BITTSET).

Wynikiem dziaîania jest 0, gdy n-ty bit (liczâc od koïca i rozpoczynajâc

liczenie od 0) jest zgaszony(=0), a 1, gdy jest zapalony (=1). Przykîad:

BITTST('0313',X,4)

(da w wyniku 1).

1.16 bitxor

BITXOR (bit [by bit] XOR -- wykonaj logiczne "lub" rozîâczne) *** FUNKCJA

BITXOR (a$,b$ [,p])

Znaczenie argumentów jak w funkcji BITAND.

Wykonuje operacjë logicznâ "lub" rozîâczne na îaïcuchach a$ i b$.

Pozostaîe uwagi jak przy BITAND. Przykîad:

BITXOR('0313'X,'001F'X)

(da w wyniku '030C'),

BITXOR('01110110','0101')

(da w wyniku îaïcuch '11010110'), zaô

BITXOR('01110110','0101',4)

(da w wyniku îaïcuch '01111100').

1.17 break

BREAK (zatrzymaj) *** INSTRUKCJA

BREAK

Instrukcji BREAK uûyjesz wówczas, gdy bëdziesz chciaî opuôciê pëtlë DO

przed wykonaniem caîej pëtli lub po to, aby wyjôê z instrukcji

INTERPRET. BREAK pozwala na wyjôcie z pëtli "nieskoïczonych" jak np.:

Do

A=A+1

Y.A = nazwisko

SAY "WPROWADZONO" nazwisko

END

Moûesz tu sobie wprowadzaê nazwiska "ad mortam defecatam", chyba

ûe po drugim wierszu umieôcisz (przykîadowo):

IF A>50 THEN BREAK

1.18 c2b

C2B ([convert] c[haracter string into] b[inary string] -- przekształê
îaïcuch znakowy w binarny) *** FUNKCJA
C2B(a$)
gdzie: a$ -- to (jak îatwo sië domyôliê) îaïcuch znakowy.
Tworzy îaïcuch binarny bëdâcy odpowiednikiem îaïcucha a$. UWAGA:
Znaczenie sîowa "Odpowiednik" naleûy potraktowaê w ten sposób, ûe kaûdemu
znakowi îaïcucha a$ zostaje przypisany jego kod ASCII, a nastëpnie kody
te zostajâ przekształcone na postaê binarnâ. Przykîad:
SAY C2B('abc')
(da w wyniku îaïcuch '011000010110001001100011').

1.19 c2d

C2D ([convert] c[haracter string into] d[ecimal] -- przekształê îaïcuch
znakowy w liczbë) *** FUNKCJA
C2D(a$ [,n])
gdzie: a$ -- to îaïcuch binarny lub heksadecymalny, zaô n -- cyfra.
Zamienia îaïcuch a$ na odpowiadajâcy mu kod ASCII
(podany cyfrâ dziesiëtnâ). Podczas konwersji dodawany jest znak bitu.
Jeôli podasz parametr n, wówczas îaïcuch a$ zostanie potraktowany jako
"sîowo" n-bajtowe. W takim przypadku îaïcuch zostanie odpowiednio
zaokrâglony lub uzupeîniony na poczâtku zerami. Jeôli wartoôê îaïcucha
przekracza (dziesiëtnie) 256, a nie uûyîeô n lub n jest wiëksze od 2,
wówczas funkcja przyjmie wartoôê -1. Przykîad:
SAY C2D('0020'X)
(da w wyniku 32),
SAY C2D('FFFF')
(da ten sam wynik), zaô
SAY C2D('FF01000'X,2)
(da w wyniku 256).

1.20 c2x

C2X ([convert] c[haracter string into he] x[adecimal string] --
przekształê îaïcuch znakowy w heksadecymalny) *** FUNKCJA
C2X(a$)
gdzie: a$ -- to îaïcuch znakowy.
Zamienia îaïcuch znakowy w jego "odpowiednik" heksadecymalny. Przykîad:
SAY C2X('abe')
(da w wyniku '616265').

1.21 call

CALL (wywoîaj) *** INSTRUKCJA
CALL <a$|b$> [w1 [,w2, ... , wn]]
gdzie: a$ -- to zmienna îaïcuchowa, b$ -- îaïcuch, zaô w1 ... wn -- wyraûenia.
Wywoîuje funkcjë wewnëtrznâ lub zewnëtrznâ o nazwie a$ lub b$. Uûyte
wyraûenia stajâ sië parametrami wywoîanej funkcji (tzn. wartoôciami, z
jakimi funkcja bëdzie obliczana). Wartoôê przyjëta lub obliczona przez
wywoîanâ funkcjë zostanie podstawiona pod zmiennâ o nazwie RESULT.
Przykîad:
CALL center nazwa,l+4,'+'
wywoîa funkcjë wewnëtrznâ CENTER (opisanâ niûej), obliczy jej wartoôê
z parametrami a$="nazwa", l=l+4 i p$="+". Jeôli przed wywoîaniem funkcji
zmienna l miaîa wartoôê na przyklad 3 -- wówczas pod zmiennâ RESULT zostanie
podstawiony îaïcuch '+nazwa+'.

1.22 close

CLOSE (zamknij) *** FUNKCJA
CLOSE(a$)
gdzie:a$ -- îaïcuch alfanumeryczny.
Zamyka zbiór o podanej nazwie a$ (uwaga: îaïcuch a$ musi byê podany w
apostrofach). Jeûeli zbiór o takiej nazwie nie zostaî wczeôniej otwarty,
wówczas funkcja przyjmie wartoôê = 1. Przykîad:
SAY CLOSE('moj_zbior')
(zamknie zbiór o nazwie "moj_zbior". Jeôli nie otworzyîeô go wczeôniej -- da
w wyniku 1).

1.23 compress

COMPRESS (zgnieê) *** FUNKCJA
COMPRESS (a$ [,b$])
gdzie: a$,b$ -- îaïcuchy.
Usuwa z îaïcucha a$ wszystkie spacje (zarówno wewnëtrzne, jak i
zewnëtrzne). Jeôli uûyjesz parametru b$ -- z îaïcucha a$ zostanâ usuniëte
wszystkie znaki, jakie wystëpujâ w zmiennej b$. Przykîad:
COMPRESS(' Commodore_Amiga ')
(da w wyniku îaïcuch 'CommodoreAmiga'),
COMPRESS('+12.34-56+12.82','-+.')
(da w wyniku '1234561282').

1.24 compare

COMPARE (porównaj) *** FUNKCJA
COMPARE (a$,b$ [,c$])
gdzie: a$,b$ -- îaïcuchy, c$ -- îaïcuch 1-znakowy.
Porównuje îaïcuchy a$ i b$ i podaje numer pierwszej pozycji, na której sâ
róúnice, lub 0 -- gdy îaïcuchy sâ identyczne. Jeôli îaïcuchy majâ róûnâ
dîugoôê, wówczas krótszy z nich zostaje uzupeîniony na poczâtku spacjami
(lub znakiem podanym przez zmiennâ c$) -- tak, aby oba îaïcuchy byîy tej
samej dîugoôci. Przykîad:
SAY COMPARE (’abcde’,’abcie’)
(da w wyniku 4),
SAY COMPARE(’abcd’,’abcd’)
(da w wyniku 0), zaô
SAY COMPARE(’abc’,’+++asm’,’+’)
(da w wyniku 5)

1.25 copies

COPIES (skopiuj) *** FUNKCJA
COPIES (a$,n)
gdzie:a$ -- îaïcuch, n -- liczba.
Tworzy nowy îaïcuch bëdâcy n-krotnym powtórzeniem îaïcucha a$.
Jeôli podasz n=0 -- wówczas w wyniku otrzymasz îaïcuch pusty. Przykîad:
SAY COPIES(’bar’,4)
(da w wyniku ’barbarbarbar’).

1.26 d2c

D2C ([convert] d[ecimal into] 2[binary string] -- przekształîê liczbë w
îaïcuch dwójkowy) *** FUNKCJA
D2C (n)
gdzie: n -- liczba.
Tworzy îaïcuch, którego wartoôciâ jest binarny odpowiednik podanej liczby
dziesiëtnej n. Zera znajdujâce sië po konwersji na poczâtku îaïcucha
wynikowego zostanâ uciëte. Przykîad:
D2C(4)
(da w wyniku ’100’).

1.27 d2x

D2X ([convert] d[ecimal into he] x[adecimal string -- przekształîê liczbë w
îaïcuch heksadecymalny) *** FUNKCJA
D2X (l [,n])
gdzie: l,n -- liczby.
Zamienia liczbë
dziesiëtnâ l na îaïcuch hexadecymalny. Jeôli uûyjesz parametru n --
îaïcuch wynikowy zostanie przedstawiony jako liczba n-znakowa,
na przykîad:
D2X(10)
(da w wyniku ’0A’X),
D2X(10,1)
(da w wyniku ’A’X).

1.28 datatype

DATATYPE (typ danej) *** FUNKCJA
DATATYPE (a$ [,b$])
gdzie: a$ -- îaïcuch, b$ -- îaïcuch 1-znakowy.
Jeôli nie uûyjesz parametru b$ -- wówczas funkcja sprawdzi, jakim typem
zmiennej jest îaïcuch a$ (jeôli jest zmiennâ liczbowâ -- wtedy wynikiem
bëdzie NUM, jeôli znakowâ -- wynikiem bëdzie CHAR). Uûycie parametru b$
spowoduje testowanie îaïcucha a$ w zaleúnoôci od znaku, jakim jest b$.
Masz do wyboru:
* b$=’A’ -- badanie, czy îaïcuch a$ skîada sië ze znaków
alfanumerycznych (maîe i duûe litery alfabetu, cyfry 0-9),
* b$=’B’ -- badanie, czy a$ jest îaïcuchem binarnym,
* b$=’L’ -- sprawdzenie, czy zmienna a$ skîada sië wyîâcznie
z maîych liter alfabetu,
* b$= ’M’ -- czy skîada sië wyîâcznie z liter (maîych i duûych),
* b$ = ’N’ -- czy skîada sië z cyfr,
* b$ = ’S’ -- czy zmienna a$ zawiera wyîâcznie prawidîowe (z punktu
widzenia jëzyka ARexx) symbole,
* b$ = ’U’ -- sprawdzi, czy zmienna a$ skîada sië wyîâcznie z duûych
liter alfabetu,
* b$ = ’W’ -- zbada, czy a$ jest liczbâ caîkowitâ, zaô
* b$= ’X’ -- czy jest îaïcuchem heksadecymalnym.
Jeôli îaïcuch a$ speînia warunki postawione przez test (okreôlony przez b$),

funkcja przyjmuje wartoôê true(1), w przeciwnym wypadku -- false (0). Przykîad:
SAY DATATYPE('123')
(da w wyniku NUM),
SAY DATATYPE('1AF2','X')
(wynikiem bëdzie 1), zaô z
SAY DATATYPE('AbCdEf','U')
(wyjdzie 0).

1.29 date

DATE (data) *** FUNKCJA
DATE ([a$],[d],[b$])
gdzie: a$,b$ -- îaïcuchy, d-liczba caîkowita.
Ustawia aktualnâ datë w formacie zaleûnym od podanych parametrów.
Parametr a$ moûe przyjmowaê nastëpujâce wartoôci:
B (beginning -- poczâtek) -- liczba dni od 1.01.0001,
C (century -- wiek) -- liczba dni od 1.01.1900 ,
D -- liczba dni od 1.01 obecnego roku,
E -- data w formacie DD/MM/RR (gdzie: DD -- dzieï (liczba),
MM-miesiâc (liczba), RR-rok),
I -- liczba dni w formacie wewnëtrznym (liczba),
J -- data w formacie RRMMM, gdzie MMM to trzyliterowy skrót angielskiej
nazwy miesiâca (na przykîad Jan -- January) -- styczeï),
M -- peîna angielska nazwa miesiâca (na przykîad February -- luty),
N -- data w formie DD MMM RRR (ustawiona domyôlnie,
gdy opuôcisz parametr a$),
O -- data w formie RR/MM/DD,
S -- data w formie RRRR MM DD,
U (US Standard) -- data w formacie MM/DD/RR (jak na IBM),
W (weekday) -- angielska nazwa dnia tygodnia.
Parametr d okreôla, czy data ma byê podana w postaci prostej
(rok na koïcu -- ustawione defaultowo), czy "sortowanej" (rok
na poczâtku). Parametr b$ -- ma znaczenie podobne jak a$ -- z tym,
ûe moûe przybieraê tylko 2 wartoôci (I lub S). Dla daty 10 luty 1991:
SAY DATE()(da w wyniku 10 Feb 1991),
SAY DATE('M')
(wynikiem jest: February),
SAY DATE('S')(w wyniku otrzymasz: 1991021.)
UWAGA: Mimo ûe dziô mamy (zaîóûmy) 10.2.91, w mojej Amidze data
systemowa (bo nie mam zegara) jest akurat: 20.4.1988. Zatem:
SAY DATE('S',DATE('I')+21) (daje '19890609'), zaô
SAY DATE('W',19890609,'S')
(powoduje wynik:'Friday').

1.30 delstr

DELSTR (del[ete] str[ing] -- usuï îaïcuch) *** FUNKCJA
DELSTR (a$,n [,l])
gdzie: a$ -- îaïcuch, n,l -- liczby caîkowite.
Usuwa z îaïcucha a$ jego fragment zaczynajâcy sië od n-tego znaku i o
dîugoôci l znaków. Jeôli opuôcisz parametr l -- wówczas defaultowo
zostanie usuniëta reszta îaïcucha (tzn. od pozycji n do koïca).
Przykîadowo:
SAY DELSTR('123456',2,3)
(da w wyniku îaïcuch '156').

1.31 delword

DELWORD (del[ete] word [from string] -- usuï sîowo z îaïcucha) *** FUNKCJA
DELWORD (a$,n [,l])
gdzie: a$ -- îaïcuch, n,l -- liczby caîkowite.
Usuwa z îaïcucha a$ -- l sîów (za sîowo jest uwaûany fragment îaïcucha,
po którym nastëpuje spacja). Usuwanie rozpoczyna sië od n-tego z kolei
sîowa. Jeôli opuôcisz parametr l -- zostanâ usuniëte wszystkie sîowa
poczynajâc od n-tego sîowa do koïca îaïcucha. UWAGA: Îaïcuch wynikowy
zawiera spacjë, która nastëpuje po ostatnim nie usuniëtym sîowie.
Przykîad:
SAY DELWORD('Opowiedz mi ladna bajke',2,2)
(wypisze na ekranie "Opowiedz bajke"),
SAY DELWORD('001 002 003 004 005',3)
(da w wyniku îaïcuch '001 002')

1.32 digits

DIGITS ([return] digits [number set] -- podaj liczbë ustawionych miejsc
dziesiëtnych (tu: precyzjë) *** FUNKCJA
DIGITS ()
Podaje precyzjë, jaka zostaîa ostatnio ustawiona instrukcjâ NUMERIC. Jeôli
jest to (przykîadowo) NUMERIC DIGITS 8, wówczas
SAY DIGITS()
(da w wyniku 8).

1.33 do

DO (zrób) *** INSTRUKCJA
DO [n1=w1 [TO w2] [BY w3] [FOR w4] [FOREVER] [WHILE w5 | UNTIL w6]
Znaczenie parametrów: n1 -- nazwa zmiennej , w1 ... w6 -- wyraûenia.
Jest to odpowiednik Basicowego FOR. Instrukcja DO otwiera grupë rozkazów,
które majâ byê wykonywane wielokrotnie (czyli "pëtli"). Kaûdy blok
instrukcji otwartych przez DO -- musi koïczyê sië instrukcjâ END.
Wszystkie rozkazy, zawarte pomiëdzy DO i END, sâ w ARexx'ie nazywane
zakresem dziaîania instrukcji DO. Nie przestrasz sië zatem, jeôli
napotkasz gdzieô takâ nazwë. Wszystkie parametry w DO sâ opcjonalne (tzn.
moûesz je opuôciê). Jeôli opuôcisz wszystkie parametry -- wówczas
instrukcje zawarte w bloku DO ... END zostanâ wykonane tylko raz.
Parametr n1 jest zmiennâ licznikowâ przyjmujâcâ wartoôê wyraûenia w1.
Jeôli uûywasz jakichkolwiek opcji (poza FOREVER), wówczas zmienna
licznikowa musi byê pierwszym wyraûeniem nastëpujâcym bezpoôrednio po
instrukcji DO.
Wyraûenie w2 (po sîowie TO) okreôla wartoôê zmiennej licznikowej, przy
której nastâpi zakoïczenie dziaîania bloku DO ... END.
Wyraûenie w3 (po sîowie BY) -- okreôla przyrost zmiennej licznikowej i
moûe przyjmowaê dowolne wartoôci numeryczne. Jeôli opuôcisz parametr BY --
wówczas wartoôê w3 zostanie domyôlnie ustawiona na 1.
Wyraûenie w4 (po sîowie FOR) musi dawaê wynik, bëdâcy liczbâ caîkowitâ
okreôlajâcâ minimalnâ iloôê iteracji, jaka ma byê wykonana. Opcji FOR moûna
uûywaê razem z opcjami BY i TO.
Uûycie sîowa kluczowego FOREVER -- spowoduje wykonywanie instrukcji do
momentu napotkania w bloku DO ... END instrukcji LEAVE lub BREAK. UWAGA:
W tym przypadku nie naleûy uûywaê zmiennej licznikowej. Z tego wynika
wniosek, ûe FOREVER powoduje wykonywanie pëtli bez nadawania indeksów
zmiennym (na przykîad, jeôli chcesz zatrzymaê przez chwilë program).
Identyczny efekt jak opcja FOREVER da instrukcja:
DO i=1
Wyraûenie w5 (po sîowie WHILE) jest obliczane przy kaûdej nowej iteracji.
Wynik wyraûenia musi byê równy 1 (wówczas nastëpuje kolejna iteracja) lub
0 -- wówczas "pëtla" koïczy swoje dziaîanie.
Na identycznych zasadach (tyle ûe "odwrotnie" -- tj: 0-dalsza iteracja,
1-koniec) "dziaîa" wyraûenie w6 w opcji UNTIL. Przykîad:
DO i=1 TO limit FOR 5 WHILE time<10
y.i=i*time

END
teraz (dla porównania) "robiâca to samo" pëtla w BASIC-u:
10 i=1
20 y(i)=i*time:if time>10 then end
if i<5 then 30
if i=limit then end
30 i=i+1:goto 20
I co jest prostsze? Jeôli jeszcze nie jesteô przekonany, spróbuj
"uproôciê" sobie program w BASIC-u przez zastosowanie pëtli FOR ... NEXT.

1.34 drop

DROP (wypadnij) *** INSTRUKCJA
DROP a [b ... z]
gdzie: a-z -- symbole zmiennych dowolnego typu.
Jako wartoôê zmiennej zostanie podstawiona nazwa zmiennej. Jeôli uûyjesz
nazwy zmiennej rdzeniowej -- wówczas operacja DROP zadziaîa na
wszystkich -- moûliwych do uzyskania z tej zmiennej -- zmiennych zîoûonych.
Czynnoôê wykonywana przez funkcjë DROP nazywana bywa czasem
"antyinicjalizacjâ". Jeôli usiîujesz "zDROPiê" zmiennâ, którâ juû wczeôniej
tak potraktowaîeô -- wówczas nic sië nie stanie. Na przykîad:
a=1;b=12
SAY a b
(tu zostanie wydrukowane 1 i 12), ale jeôli uûyjesz
DROP a b
SAY a b
(to zostanie wydrukowane A i B).
I to by byîo tyle (w tym miesiâcu).

1.35 echo

ECHO *** INSTRUKCJA
ECHO w1
Parametr w1 -- to wyraûenie.
Wyôwietla wynik wyraûenia na ekranie (tak samo jak SAY). Uwaga: Jeôli wyraûenie
jest îaïcuchem -- wówczas musi byê tu podane w cudzysîowie. Przykîadowo:
ECHO "AMIGA 500"
(da ten sam efekt co
SAY ’AMIGA 500’
a bëdzie to wypisanie na ekranie najmilej brzmiâcej na ôwiecie nazwy.)

1.36 else

ELSE (w innym przypadku) *** INSTRUKCJA
ELSE [;] [w1]
gdzie: w1 -- to wyraûenie warunkowe.
Instrukcja (podobnie jak w BASIC-u) pozwala na wybór róûnych moûliwoôci
w instrukcji IF. UWAGA: Nie moûe byê uûyta poza zakresem dziaîania instrukcji
IF i musi wystëpowaê po instrukcji THEN. Jeôli zagnieúdzisz w sobie kilka IF-ów,
wówczas ELSE dziaîa w zakresie najbliûszego poprzedniego IF.
W przypadku zagnieûdûenia, nie wolno stosowaê "pustego" ELSE (zamiast tego musisz
uûyê instrukcji NOP), ani teû ELSE ze ôrednikiem (ELSE;, które jest równowaûne
"pustemu" ELSE). Przykîad:
IF i<>2 THEN SAY 'chyba nie'
ELSE SAY 'chyba tak'

1.37 end

END (koniec) *** INSTRUKCJA
END [n]
Parametr n -- jest symbolem zmiennej licznikowej w instrukcji DO.
Instrukcja END koïczy zakres dziaîania instrukcji DO lub SELECT.
Zmiennej opcjonalnej n uûyj wtedy, jeôli masz w programie wiele
instrukcji DO i chcesz byê pewny, ûe END odnosi sië do
wîaôciwego DO (jeôli nazwy symboli sië nie zgadzajâ, wówczas
wystâpi bîâd). Przykîad:
DO i=1 TO 11 BY 2
SAY i
END i
(na ekranie zostanâ wydrukowane cyfry 1,3,5,7,9,11).

1.38 eof

EOF (e[nd] o[f] f[ile] -- koniec zbioru) *** FUNKCJA
EOF (a$)
gdzie: a$ -- to îaïcuch znakowy.
Funkcja sprawdza, czy zbiór o nazwie a$ (UWAGA: Tu nazwa jest
podana bez apostrofów) ma ustawiony znacznik koïca zbioru (end
of file). Jeôli tak -- wówczas funkcja przyjmuje wartoôê "true" (1),
w przeciwnym wypadku "false" (0). Przykîad:
SAY EOF(moj_zbior)
(Jeôli zbiór "moj_zbior" ma znacznik koïca, wówczas na ekranie
pojawi sië 1, jeôli zaô go nie ma, wtedy pojawi sië 0).

1.39 errortext

ERRORTEXT ([return] error [message] text -- podaj tekst komunikatu o
bîëdzie) *** FUNKCJA
ERRORTEXT (n)
gdzie: n -- liczba z zakresu 1 -- 48.
Funkcja przyjmuje jako wartoôê informacjë o bîedzie, którego kod wynosi
n. Jeôli liczba n jest niecaîkowita lub spoza zakresu -- wówczas zostanie
utworzony îaïcuch pusty. ARexx informuje o nastëpujâcych bîëdach:
* kod=1 Program not found -- nie znaleziono programu o podanej nazwie, lub
nie jest to program ARexxa.
* 2 -- Execution halted -- program zostaî zatrzymany -- albo z klawiatury
(jeôli nacisnâîeô jednoczeônie klawisze [Ctrl] i [C]), lub z uûyciem
przerwania programowego. UWAGA: Z tego bîëdu moûesz wyjôê, jeûeli wczeôniej
uaktywniîeô przerwanie typu HALT.
* 3 -- Insufficient memory -- brak wystarczajâcej iloôci pamiëci do
wykonania programu.
* 4 -- Invalid character -- w programie uûyîeô nie zdefiniowanego
wczeôniej znaku niestandardowego (na przykîad o kodzie 281, spoza zakresu ASCII).
Aby zdefiniowaê taki znak -- musisz przed pierwszym uûyciem przedstawiê go
w postaci îaïcucha binarnego lub heksadecymalnego.
* 5 -- Unmatched quote -- Otworzyîeô cudzysîów lub apostrof i zapomniaîeô go
zamknâê.
* 6 -- Unterminated comment -- Otworzyîeô komentarz (znakiem /*),
ale nie zamknâîeô go (przez */).
* 7 -- Clause too long -- Uûyte przez Ciebie wyraûenie jest zbyt dîugie,
aby zmieôciîo sië w buforze. Musisz podzieliê je na kilka krótszych.
* 8 -- Invalid token -- w wyraûeniu znajduje sië nieprawidîowy (z punktu
widzenia ARexxa) znak lub operator.
* 9 -- Symbol or string too long -- uûyta zmienna lub îaïcuch ma zbyt
dîugâ nazwë.
* 10 -- Invalid message packet -- próbowaîeô przesîaê nieprawidîowy rozkaz
do programu rezydentnego ARexx'a.
* 11 -- Command string error -- bîâd w nazwie rozkazu,
* 12 -- Error return from function -- Niewîaôciwe parametry funkcji
zewnëtrznej spowodowaîy generacjë kodu bîëdu funkcji.
* 13 -- Host environmen not found -- brak portu komunikatów
odpowiadajâcego obecnemu adresowi zarzâdzajâcemu lub adres zarzâdzajâcy nie jest
uaktywniony.

* 14 -- Requested library not found -- Próbowaîeô otworzyê bibliotekë
o nazwie (lub numerze wersji) niezgodnej z tâ, jakâ zapisaîeô w spisie
bibliotek (Library List), lub biblioteki o podanej nazwie nie ma na dysku.
* 15 -- Function not found -- Wywoîaîeô funkcjë, która:
-- nie jest wbudowana,
-- nie jest zdefiniowana jako program zewnëtrzny lub
-- nie ma jej w spisie bibliotek.
* 16 -- Function did not return value -- wywoîaîeô funkcjë, a ta funkcja
zîoôliwie nie przyjëîa ûadnej wartoôci (bo na przyklad nie spodobaîy sië
jej Twoje parametry, choê na oko iloôê i jakoôê parametrów sië zgadza).
* 17 -- Wrong number of arguments -- tu równieû wywoîana funkcja
zastrajkowaîa, gdyû podaîeô jej zbyt maîâ lub zbyt duûâ liczbë parametrów
(w stosunku do liczby wymaganej).
* 18 -- Invalid argument to function -- Opuôciîeô argument funkcji,
który jest niezbëdny do prawidîowego dziaîania funkcji lub podaîeô go w sposób
niewîaôciwy.
* 19 -- Invalid PROCEDURE -- Uûyîeô instrukcji PROCEDURE w niewîaôciwym
kontekôcie. Bîâd ten wystëpuje równieû, gdy w PROCEDURE wywoîujesz
nie uaktywnionâ funkcjë wewnëtrznâ lub PROCEDURE zostaîo wczeôniej
umieszczone w aktualnym ôrodowisku pamiëciowym.
* 20 -- Unexpected THEN or WHEN -- uûyîeô jednej z tych dwu instrukcji w
niewîaôciwym kontekôcie (WHEN moûna uûywaê tylko w zakresie SELECT lub
DO, THEN moûe wystepowaê po IF lub WHEN).
* 21 -- Unexpected ELSE or OTHERWISE -- uûyîeô którejô z tych instrukcji w
zîym kontekôcie (OTHERWISE moûe byê uûyte tylko w zakresie SELECT, ELSE
moûna uûyê tylko po IF).
* 22 -- Unexpected BREAK, LEAVE or ITERATE -- uûyîeô którejô z tych
instrukcji poza zakresem instrukcji DO.
* 23 -- Invalid statement in SELECT -- W zakresie instrukcji SELECT mogâ
znajdowaê sië tylko WHEN, THEN i OTHERWISE oraz wyraûenia warunkowe
zwiâzane z tymi instrukcjami.
* 24 -- Missing or multiple THEN -- Zapomniaîeô umieôciê THEN po WHEN lub
IF albo uûyîeô zbyt duûo THEN po pojedyïczym IF (WHEN).
* 25 -- Missing OTHERWISE -- jeôli ûaden z warunków postawionych przez WHEN
nie jest speîniony -- na koïcu MUSI wystëpowaê instrukcja OTHERWISE.
* 26 -- Missing or unexpected END -- nie zamknâîeô zakresu instrukcji
DO(SELECT) przez END lub uûyîeô END poza zakresem wyznaczonym przez të
instrukcjë (na przykîad po kolejnym DO).
* 27 -- Symbol mismatch -- symbol okreôlajâcy zmiennâ uûyty w instrukcjach

END, ITERATE lub LEAVE nie zgadza sië z symbolem bëdâcym zmiennâ
licznikowâ poprzedzajâcej je instrukcji DO, lub dwa zakresy wyznaczone przez DO
zachodzâ na siebie.
* 28 -- Invalid DO syntax -- jeûeli uûyîeô parametrów TO lub BY w
instrukcji DO -- muszâ byê one poprzedzone okreôleniem zmiennej indeksujâcej.
* 29 -- Incomplete IF or SELECT -- W instrukcji IF lub SELECT nie umieôciîeô
wszystkich wymaganych przez te instrukcjeparametrów (na przykîad wyraûeï
warunkowych po sîowach THEN, ELSE, OTHERWISE).
* 30 -- Label not found -- brak w programie etykiety uûytej w instrukcji
SIGNAL, lub etykiety do której odwoîujesz sië w programie bezpoôrednio
przez przerwanie. UWAGA: Bîâd nie wystâpi, jeôli etykieta zdefiniowana
jest dynamicznie instrukcjâ INTERPRET lub przez interaktywne wprowadzenie
z klawiatury.
* 31 -- Symbol expected -- w miejscu gdzie w programie powinna znajdowaê
sië nazwa zmiennej (na przykîad w instrukcjach END, LEAVE, UPPER itd)
umieôciîeô inny znak lub pominâîeô të nazwë.
* 32 -- Symbol or string expected -- W miejscu, gdzie moûe wystepowaê
wyîâcznie nazwa zmiennej lub îaïcuch -- umieôciîeô inny znak.
* 33 -- Invalid keyword -- bîëdnie wpisaîeô sîowo kluczowe
(czyli to, co w opisie jest duûymi literami).
* 34 -- Required keyword missing -- Zapomniaîeô umieôciê w instrukcji
sîowa kluczowego, które musi wystëpowaê (na pzrykîad sîowa kluczowe dotyczâce
obsîugi przerwaï typu SYNTAX itd. w instrukcji SIGNAL).
* 35 -- Extraneous character -- uûyîeô niepotrzebnych (nadmiarowych)
znaków na koïcu wyraúenia.
* 36 -- Keyword conflict -- Uûyîeô dwóch wzajemnie wykluczajâcych sië sîów
kluczowych (czyli takich, które w opisie rozdzielone sâ znakiem |) lub
zastosowaîeô dwa identyczne sîowa kluczowe w tej samej instrukcji.
* 37 -- Invalid template -- szablon zamieszczony w instrukcjach
ARG, PARSE lub PULL zostaî niewîaôciwie napisany,
* 38 -- Invalid TRACE request -- sîowo kluczowe w instrukcji TRACE lub
argument w funkcji wewnëtrznej TRACE sâ nieprawidîowe.
* 39 -- Uninitialized variable -- przy uaktywnionych przerwaniach typu
NOVALUE próbowaîeô wywoîaê zmiennâ niezainicjalizowanâ.
* 40 -- Invalid variable name -- próbowaîeô podstawiê nowâ wartoôê pod
nazwë oznaczajâcâ staîâ.
* 41 -- Invalid expression -- podczas obliczania wartoôci wyraûenia
nastâpiî bîâd (na przykîad dzielenie przez 0). Bîâd wystâpi takûe
przy niewîaôciwej liczbie operandów w wyraûeniu lub przy nadmiarowych znakach

w tym wyraûeniu
* 42 -- Unbalanced parentheses -- nie zamknâîeô otwartego nawiasu.
* 43 -- Nesting limit exceeded -- uûyîeô za duûej liczby zagnieûdûonych
instrukcji DO (wiëcej niû 10) lub "podprogramów" w wyraûeniu (wiëcej niû 32).
Moûesz usunâê bîâd zamieniajâc "pëtlë" DO na iteracjë lub rozdzielajâc wyraûenie
na dwa lub wiëcej wyraûeï.
* 44 -- Invalid expression result -- obliczona wartoôê wyraûenia jest
nieprawidîowa (na przykîad skok lub wartoôê koïcowa zmiennej licznikowej w
instrukcji DO jest nienumeryczna).
* 45 -- Expression required -- opuôciîeô wymagane w danym kontekôcie
wyraûenie (na przykîad jeôli w instrukcji SIGNAL opuôcisz sîowo kluczowe ON
(lub OFF) wówczas musisz uûyê wyraûenia).
* 46 -- Boolean value not 0 or 1 -- wyraûenie logiczne przyjëîo innâ
wartoôê niû 0 lub 1.
* 47 -- Arithmetic conversion error -- Uûyîeô operandu nienumerycznego w
wyraûeniu numerycznym lub bîëdnie wpisaîeô îaïcuch binarny czy heksadecymalny.
* 48 -- Invalid operand -- uûyîeô nieprawidîowego operandu (na przyklad
dzielenie przez 0) lub w funkcji wykîadniczej uûyîeô wykîadnika niecaîkowitego.
UWAGA: Kaûdy bîâd powoduje generacjë Amigowskiego kodu zwrotnego (z wszelkimi
wynikajâcymi z tego konsekwencjami). Przewaûnie kodem zwrotnym jest 10,
jedynie bîëdy o kodzie ARrexx'a 3 i 43 generujâ Amigowski kod zwrotny = 20,
zaô bîâd ARexx'a o kodzie 1 generuje Amigowski kod zwrotny = 5. Przykîad:
SAY ERRORTEXT(42)
(spowoduje wypisanie komunikatu "Unbalanced parentheses).

1.40 exists

EXISTS ([file] exists -- czy zbiór istnieje) *** FUNKCJA
EXISTS (a$)
gdzie: a$ -- îaïcuch (tu podany w apostrofach).
Sprawdza, czy istnieje zewnëtrzny zbiór o nazwie a$. Îaïcuch moûe zawieraê
scieûkë. W wyniku otrzymasz wartoôê = 1 (gdy zbiór istnieje) lub = 0 (gdy
go nie ma). Przykîad:
SAY EXISTS('df1:c/ed')
(da w wyniku na przykîad 1 -- jeôli na dyskietce w stacji df1: w katalogu c
jest zbiór o nazwie "ed").

1.41 exit

EXIT (wyjôcie) *** INSTRUKCJA
EXIT [w1]
Parametr w1 jest wyraûeniem.
Instrukcja przerywa wykonywanie programu. Moûesz jâ umieôciê w dowolnym miejscu programu. Jeûeli uûyîeô parametru w1 -- wówczas wartoôê wyraûenia zostanie obliczona i przesîana do miejsca w programie zewnëtrznym, z którego nastâpiîo wywoîanie programu przerwanego przez EXIT. Przykîad:
EXIT
(przerywa wykonanie programu i
powraca bez obliczonej wartoôci wyraûenia).
EXIT 10
(przerywa wykonanie programu i generuje kod zwrotny 10).

1.42 export

EXPORT (wyôlij) *** FUNKCJA
EXPORT (a$ [,b$] [,l] [,c$])
Znaczenie parametrów jest nastëpujâce: a$,b$ -- îaïcuchy, c$-îaïcuch 1-znakowy, l-liczba caîkowita.
Funkcja EXPORT kopiuje îaïcuch b$ w obszar pamiëci o adresie a$ (który musi byê 4-bitowym îaïcuchem binarnym lub heksadecymalnym).
Jeôli uûyjesz parametru l -- wówczas zostanie skopiowane l pierwszych znaków îaïcucha b$ (domyôlnie przyjëta jest caîkowita dîugoôê laïcucha b$).
Jeôli l jest wiëksze niû dîugoôê îaïcucha, wówczas îaïcuch zostanie uzupeîniony na poczâtku zerami, a jeôli uûyîeô parametru c$ -- znakiem, jaki wystëpuje w c$.
Jednoczeônie funkcja przyjmie wartoôê = l. UWAGA: Poczâtkujâcym "AReksiom" radzë NIE EKSPERYMENTOWAÊ z tâ funkcjâ, jako ûe kopiowany îaïcuch b$ zapisuje sië na tym, co juû jest w pamiëci pod adresem a$. Od momentu uûycia funkcji nie wolno przeîâczaê dostëpów. Zabawa w EXPORT-erów moûe zakoïczyê sië zawieszeniem komputera!!!. Przykîad:
co=EXPORT('0004 0000"X,'AMIGA')
(skopiuje sîowo "Amiga" do komórki o adresie 40000, a zmienna co przyjmie wartoôê 5).

1.43 form

FORM ([return] form -- podaj formë) *** FUNKCJA
FORM()
Podaje nazwë aktualnie ustawionego formatu zapisu cyfr. Przykîad:
NUMERIC FORM SCIENTIFIC
SAY FORM()
(wypisze na ekranie sîowo SCIENTIFIC (tu notacja "naukowa"))

1.44 find

FIND (znajdú) *** FUNKCJA
FIND (a$,b$), gdzie: a$,b$ -- to îaïcuchy.
Okreôla pozycjë zestawu sîów b$ w wiëkszym zestawie sîów a$ i przyjmuje
wartoôê równâ miejscu wystëpowania pierwszego sîowa îaïcucha b$ w
îaïcuchu a$. Jeôli nie znaleziono danego zestawu sîów b$ -- wówczas
funkcja przyjmie wartoôê = 0. Przykîad:
SAY FIND ('Najlepszym komputerem jest Commodore Amiga','jest Commodore')
(da w wyniku 3).

1.45 freespace

FREESPACE (wolne miejsce [tu: iloôê pamiëci]) *** FUNKCJA
FREESPACE (a$,b)
Znaczenie parametrów: a$ -- îaïcuch, b -- liczba.
Przekazuje blok pamiëci rozpoczynajâcy sië od adresu a$ (îaïcuch
4-bajtowy) i majâcy dîugoôê l do wewnëtrznego bufora
interpretera. Jeôli opuôcisz argumenty [UWAGA: skîadnia w tym przypadku musi byê:
FREESPACE()] --- wówczas funkcja poda wielkoôê pamiëci wewnëtrznego bufora
interpretera. Nie jest konieczne uûywanie tej funkcji na poczâtku
programu, poniewaû wówczas alokacja zostaje wykonana automatycznie,
naleûy jednak z niej korzystaê wtedy, gdy zaczynasz mieê problemy z
pamiëciâ w programie. Funkcja przyjmuje wartoôê 1 -- gdy operacja
przebiegnie pomyôlnie, lub wartoôê 0 -- gdy jest niemoûliwa. UWAGA:
Wygodnie jest uzyskaê adres pierwszej wolnej komórki bufora interpretera
za pomocâ opisanej dalej funkcji GETSPACE. Przykîad:
SAY FREESPACE ('0004 2000'X,32)
(przetransferuje 32 bajty pamiëci do komórki 42000 i (gdy
wszystko przebiegnie pomyôlnie) przyjmie wartoôê równâ 1.

1.46 fuzz

FUZZ (zakîócenia) *** FUNKCJA
FUZZ()
Podaje aktualnie ustawiony poziom zakîóceï numerycznych. Przykîad:
NUMERIC FUZZ 3
SAY FUZZ()
(da w wyniku 3).

1.47 getclip

GETCLIP (get [from] Clip[board list] -- pobierz z listy bufora) ***
FUNKCJA
GETCLIP(a$)
gdzie: a$ -- to îaïcuch.
Szuka na liôcie bufora (Clip List) danej odpowiadajâcej nazwie
podanej îaïcuchem a$ (tu a$ musi byê umieszczone w
apostrofach) i przyjmuje wartoôê tej danej (lub 0, gdy nic
nie zostaîo znalezione). Jeôli w buforze znajduje sië na
przykîad wyraûenie ’liczba’ o wartoôci równej ’PI=3.142’ i uûyjesz
SAY GETCLIP (’liczba’)
wówczas w wyniku otrzymasz: PI=3.14.

1.48 getspace

GETSPACE (pobierz [adres pamiëci z wewnëtrznego bufora interpretera] ***
FUNKCJA
GETSPACE(l) gdzie: l -- to liczba caîkowita.
Alokuje blok pamiëci o podanej dîugoôci l w wewnëtrznym buforze interpretera
i przyjmuje jako wartoôê adres pierwszej wolnej komórki w tym buforze w postaci
4-bajtowego îaïcucha. Nie zaleca sië jej stosowania w programach zewnëtrznych
obsîugiwanych przez ARexx. Przykîad:
SAY C2X(getspace(32))
(przealokuje 32-bajtowy blok pamiëci, przeksztaîci otrzymany adres na
liczbë heksadecymalnâ i wypisze na ekranie ’0003BF40’X [w przypadku gdy
jest to pierwsze uûycie funkcji od poczâtku programu])

1.49 hash

HASH (podaj kod pomyîkowy [niedozwolony -- tzw. opcode]) *** FUNKCJA
HASH (a$) gdzie: a$ -- to îaïcuch 1-znakowy.
Podaje tzw. op-code dla znaku a$ (w postaci cyfry dziesiëtnej) i
aktualizuje ten op-code. Przykîad:
SAY HASH('1')
(wyôwietli na ekranie 49).

1.50 if

IF (jeôli) *** INSTRUKCJA
IF w1 THEN [;] [w2]
Znaczenie parametrów: w1 -- wyraûenie, w2 -- wyraûenie warunkowe.
Pozwala na sterowanie dziaîaniem programu w zaleûnoôci od tego, czy
wyraûenie w1 jest speînione (tzn. przyjmuje wartoôê logicznâ "true" (1),
czy nie ("false"(0)). W przypadku "true" zostanie wykonane wyraûenie w2,
w przypadku "false" program przejdzie do nastëpnego wiersza, który moûe
zaczynaê sië instrukcjâ ELSE. Przykîad w instrukcji ELSE.

1.51 import

IMPORT (przyôlij, tu: pobierz) *** INSTRUKCJA
IMPORT(a$ [,l])
gdzie:a$ -- îaïcuch 4-bajtowy, l -- liczba caîkowita.
Wynikiem dziaîania instrukcji jest îaïcuch utworzony przez skopiowanie z
pamiëci obszaru, rozpoczynajâcego sië w komórce o adresie a$ i o dîugoôci
l. Jeûeli pominiesz l -- zawartoôê pamiëci bëdzie dopisywana do îaïcucha
wynikowego aû do momentu napotkania bajtu zerowego. Przykîad:
SAY IMPORT('0004 0000'X,8)
(da w wyniku zawartoôê 8 komórek pamiëci, poczâwszy od 40000 (na przykîad
FF FF FF 00 3A BE FF FF), zaô
SAY IMPORT('0004 0000'X)
(da w wyniku îaïcuch 'FF FF FF').

1.52 index

INDEX *** FUNKCJA
INDEX(a$,b$ [,l])
Parametrami sâ tu: a$,b$ -- îaïcuchy, l -- liczba caîkowita.
Szuka miejsca wystëpowania wzorca b$ w îaïcuchu a$ -- rozpoczynajâc
szukanie od l-tej pozycji îaïcucha a$ (gdy opuôcisz l, wówczas
przeszukiwanie nastëpuje od poczâtku îaïcucha. Jako wartoôê przyjmuje
pozycjë znalezionego wzorca w îaïcuchu a$ (liczâc od poczâtku îaïcucha
a$) lub wartoôê 0, gdy dany wzorzec w îaïcuchu nie wystëpuje. UWAGA: W
tej funkcji -- îaïcuchy sâ w cudzysîowie. Przykîad:
SAY INDEX("1234567","23")
(da w wyniku 2),
SAY INDEX("1234567","890")
(da w wyniku 0), zaô
SAY INDEX("123123123","23",3)
(da w wyniku 5).

1.53 insert

INSERT (wsuï) *** FUNKCJA
INSERT (a$,b$ [,s] [,l] [,c$])
gdzie: a$, b$ -- to îaïcuchy, c$ -- îaïcuch 1-znakowy, l,s -- liczby
caîkowite.
Wsuwa îaïcuch a$ w îaïcuch b$ poczynajâc od pozycji s+1 (jeôli opuôcisz s
-- wówczas jest dopisywany na poczâtku, jeôli s jest wiëksze niû dîugoôê
laïcucha b$, wówczas na poczâtku îaïcucha wynikowego wystëpuje îaïcuch
b$, dalej znajduje sië tyle spacji, ile wynosi róûnica miëdzy s i
dîugoôciâ îaïcucha b$, zaô na koïcu wystëpuje îaïcuch a$. Jeôli uûyjesz
parametru c$, wówczas w miejscu spacji zostanâ umieszczone znaki, jakie
zawiera zmienna c$. Parametr l podaje, jakâ dîugoôê ma mieê îaïcuch a$
jako czëôê îaïcucha wynikowego. Przykîad:
SAY INSERT('ab','12345')
(da w wyniku îaïcuch 'ab12345'), natomiast
SAY INSERT('abc','12',3,5,'*')
utworzy îaïcuch '12*abc**

1.54 interpret

INTERPRET (zinterpretuj) *** INSTRUKCJA
INTERPRET w1
gdzie: w1 -- to wyraûenie.
W wyniku dziaîania instrukcji INTERPRET -- wyraûenie w1 zostanie wykonane,
a jego wynik zostanie potraktowany jako instrukcja programowa.
Jako wyraûenia moûna uûyê takûe dowolnej instrukcji lub bloku instrukcji
(typu DO ... END). Instrukcja uaktywnia swój zakres kontrolny,
z którego moûna wyjôê poleceniami LEAVE lub ITERATE (z zachowaniem efektu
dziaîania INTERPRET) lub przez BREAK (z likwidacjâ tego efektu).
Instrukcja pozwala na dynamiczne konstruowanie i testowanie programów
(dzięki traktowaniu rozkazu jako zmiennej) lub na uûycie fragmentów programu
jako argumentów instrukcji. Przykîad:
naz ='say'
(tu podstawiamy instrukcjë jako zmiennâ naz)
INTERPRET naz Amiga500
(efektem dziaîania bëdzie wykonanie rozkazu SAY 'Amiga500', czyli
wypisanie tekstu Amiga500 na ekranie).

1.55 iterate

ITERATE (ziteruj) *** INSTRUKCJA
ITERATE [n]
gdzie: n -- to nazwa zmiennej.
Instrukcja ta przerywa dziaîanie iteracji w pëtli DO ... END i zaczyna
iteracjë od poczâtku. Jeôli w pëtli nie umieôcisz rozkazu LEAVE, wówczas
przy 11 kolejnej iteracji moûe wystâpiê bîâd. Jeôli uûyîeô nazwy zmiennej
-- musi byê ona taka sama, jak w odpowiadajâcej instrukcji DO. Zaîóûmy, ûe
j=3. Jeôli wpiszesz poniûszy przykîadowy programik:
DO i=1 to 7IF i=j THEN ITERATE i
ELSE SAY i
LEAVE
END
(wówczas dziesiëciokrotnie wydrukowana zostanie para liczb 1,2 -- po czym
nastâpi wyskok z pëtli).

1.56 lastpos

LASTPOS (last pos[ition] -- ostatnia pozycja) *** FUNKCJA
LASTPOS (a$,b$ [,s])
Znaczenie parametrów: a$ -- îaïcuch 1-znakowy, b$ -- îaïcuch, s -- liczba caîkowita.
Szuka wzorca a$ w îaïcuchu b$ rozpoczynajâc od pozycji s. Róûnica miëdzy LASTPOS i FIND polega na tym, ûe przeszukiwanie odbywa sië w tyî. Jeôli opuôcisz parametr s, wówczas szukanie rozpocznie sië od ostatniego znaku îaïcucha b$. Funkcja przyjmuje wartoôê pozycji, na której znajduje sië wzorzec, lub 0, gdy wzorzec nie zostaî znaleziony. Przykîad:
SAY LASTPOS('2','12345')
(da w wyniku 2),
SAY LASTPOS('2','123123123')
(da w wyniku 8), zaô
SAY LASTPOS('2','123234',3)
(da w wyniku 2).

1.57 leave

LEAVE (pozostaw) *** INSTRUKCJA
LEAVE [n]
gdzie: n -- to nazwa zmiennej.
Pozwala na natychmiastowe wyjôcie z zakresu instrukcji DO ... END lub ITERATE. Jeôli uûyjesz nazwy zmiennej, wówczas musi ona byê taka sama, jak w odpowiednim DO. W przeciwnym razie wystâpi bîâd. Wystâpi on równieû wtedy, gdy uûyjesz LEAVE poza zakresem DO ... END. Przykîad:
DO i=1 TO limit
IF i>10 THEN LEAVE i
END
(pëtla DO ... END zostanie wykonana 10 razy, po czym nastâpi wyskok z niej).

1.58 left

LEFT (left[most substring] -- lewa czëôê îaïcucha) *** FUNKCJA
LEFT (a$,l [,b$])
Parametrami sâ tu: a$ -- îaïcuch, b$ -- îaïcuch 1-znakowy, l -- liczba caîkowita.
Funkcja LEAVE tworzy îaïcuch zîoûony z l pierwszych znaków îaïcucha a$.

Jeôli dîugoôê îaïcucha jest mniejsza niû podana liczba l,
wówczas na koïcu îaïcucha wynikowego dopisywane sâ spacje.
Jeûeli uûyjesz parametru b$ -- w miejsce spacji zostanâ dopisane znaki
zawarte w îaïcuchu b$.
Przykîad:
SAY LEFT('1234567',4)
(da w wyniku îaïcuch '1234'), natomiast
SAY LEFT('123456,9,'*')
(wydrukuje îaïcuch '123456***').

1.59 length

LENGTH (length [of string] -- dîugoôê îaïcucha) *** FUNKCJA
LENGTH(a$)
gdzie: a$ -- to îaïcuch.
Oblicza dîugoôê îaïcucha a$, wliczajâc w to ewentualnie wystëpujâce w nim spacje.
Przykîadowo:
SAY LENGTH ('Commodore') (da w wyniku 9), zaô
SAY LENGTH ('Amiga 500)
(da w wyniku równieû 9 [gdyû spacja zostanie wliczona]).

1.60 lines

LINES ([number of] lines -- liczba wierszy) *** FUNKCJA
LINES(a$)
gdzie: a$ -- îaïcuch.
Podaje liczbë wierszy wpisanych uprzednio do zbioru, który jest aktualnie
przyporzâdkowany polu interaktywnemu. Przykîad:
PUSH 'to jest wiersz'
PUSH 'a to nastëpny'
SAY LINES(stdin)
(da w wyniku 2).

1.61 max

MAX (max[imum]) *** FUNKCJA
MAX (n1,n2 [,n3, ... , nn])
Znaczenie parametrów: n1 -- nn cyfry.
Znajduje wartoôê maksymalnâ spomiëdzy dwóch lub wiëkszej liczby
cyfr, które sâ argumentami funkcji. Przykîad:
SAY MAX(2,7,-4,12,8,15,3,9)
(da w wyniku 15).

1.62 min

MIN (min[imum]) *** FUNKCJA
MIN (n1,n2 [,n3, ... ,nn])
gdzie:n1 ... nn - cyfry.
Analogicznie jak MAX -- znajduje najmniejszy spoôród podanych argumentów cyfrowych. Przykîad:
SAY MIN (4,8,-2,7,18,0,-8,12,4)
(da w wyniku -8).

1.63 nop

NOP (n[o] op[eration] -- nie rób nic) *** INSTRUKCJA
NOP
Ta instrukcja nie robi nic. Jest jednak konieczna, gdy chcesz uûyê "pustej" instrukcji ELSE np. w zagnieûdûonych instrukcjach IF ... ELSE (uûycie samego ELSE spowodowaîoby bîâd polegajâcy na powrocie do pierwszej instrukcji IF. Przykîad: (przy próbie nie wpisuj komentarzy w nawiasach, które sâ konieczne dla zrozumienia dziaîania przykîadu):
IF i=j THEN
(to jest zewnëtrzna instrukcja IF)
IF j=k THEN a=0
(to jest wewnëtrzna instrukcja IF)
ELSE NOP
(ta instrukcja ELSE zwiâzana jest z wewnëtrznym IF)
ELSE a=a+1
(natomiast to ELSE zwiâzane jest z zewnëtrznym IF).

1.64 numeric

NUMERIC ([set] numeric [precision] -- ustaw precyzjë) *** INSTRUKCJA
skîadnia 1: NUMERIC <DIGITS | FUZZ> w1
(gdzie: w1 -- wyraûenie),
skîadnia 2:NUMERIC FORM <SCIENTIFIC | ENGINEERING>
Instrukcja (w zaleûnoôci od uûytej skîadni i sîowa kluczowego) ustawia format i precyzjë dla stosowanych w dalszej czëôci programu wyraûeï numerycznych. Ustalone w ten sposób format i precyzja bëdâ obowiâzywaê równieû przy przekazywaniu wartoôci jako argumentów funkcji wewnëtrznych. Przy skîadni pierwszej:

* DIGITS w1 -- ustawia precyzjë (liczbë miejsc, jaka bëdzie obliczana po
przecinku w obliczeniach arytmetycznych na wartoôê obliczonâ przez
wyraûenie w1 . UWAGA: Zamiast wyraûenia moûesz podaê wprost liczbë.
Wartoôê w1 musi byê caîkowitâ liczbâ dodatniâ. Przykîad:
NUMERIC DIGITS 14
(obliczenia arytmetyczne bëdâ prowadzone do 14 miejsca po przecinku),
* FUZZ w1 -- liczba miejsc dziesiëtnych oznaczona przez wyraûenie w1 bëdzie
ignorowana przez program podczas obliczeï, np.:
NUMERIC FUZZ 4
a=1.23456 ; b= 4.59824
SAY a*b
(da w wyniku 5.4, a nie jak moûna by sië spodziewaê 5.67680318294).
Dla skîadni 2:
* NUMERIC FORM SCIENTIFIC -- okreôla, ûe wszystkie wyniki od tej pory bëdâ
podawane w notacji naukowej, zaô:
* NUMERIC FORM ENGINEERING -- w notacji inûynierskiej.
[Notacja naukowa to przedstawienie liczby jako mantysy pomnoûonej
przez 10 do jakiejô potëgi, np. 1.23E3 to jest 1230 w tej
notacji. W notacji inûynierskiej mantysa jest z zakresu
1-1000, zaô wykîadnik jest wielokrotnoôciâ liczby 3, np
71D3 to 357911]. Zakîadajâc ûe a=12345 otrzymamy: przy
NUMERIC FORM SCIENTIFIC
SAY a
(da w wyniku 1.2345E4), zaô przy
NUMERIC FORM ENGINEERING
SAY a
(da w wyniku 23.111D3).

1.65 open

OPEN (otwórz) *** FUNKCJA
OPEN(a$,b$ [,'A' | 'R' | 'W']
gdzie:a$, b$ -- îaïcuchy.
Otwiera zbiór zewnëtrzny, aby wykonaê na nim oznaczone dziaîanie. a$ --
oznacza nazwë, pod jakâ dany zbiór bëdzie wywoîywany w dalszej czëôci
programu, b$ -- jest nazwâ wtórnâ (zawiera np. nazwë, pod jakâ zbiór
wystëpuje na dysku, urzâdzenie, na którym jest zainstalowany, itp.).
UWAGA: Jak pokazuje przykîad, moûesz otwieraê przez OPEN nie tylko
zbiory dyskowe. Nie ma ograniczenia co do liczby otwieranych przez OPEN

zbiorów zewnëtrznych w jednym programie. Wszystkie otwarte zbiory zostajâ
automatycznie zamkniëte w momencie wyjôcia z programu. Funkcja przyjmuje
wartoôê "true" (1), gdy operacja otwarcia sië powiedzie, lub "false" (0),
gdy wystâpi bîâd. Sîowa kluczowe oznaczajâ:
* A -- otwarcie zbioru na dopisywanie danych,
* R -- otwarcie zbioru na odczyt,
* W -- otwarcie zbioru na zapis. Oczywiôcie jeôli otwieramy urzâdzenie
inne niû zbiór dyskowy -- wówczas opuszczamy sîowo kluczowe. Przykîad:
SAY OPEN('MojeOkno','CON:160/50/320/100/MojeOkno/cds')
(otwiera okno o podanych parametrach [opis tych parametrów znajdziesz
przy opisie instrukcji WINDOW Amiga BASIC/ tom II tej ksiâûki] i
przyjmuje wartoôê 1, jeôli okno zostanie otwarte),
SAY OPEN('wyniki','df1:roboczy','W')
(otwiera na dysku df1: zbiór o nazwie "roboczy" przeznaczony na zapis.
Jeôli zbiór istnieje i nie jest zabezpieczony przed zapisem -- wówczas funkcja
przyjmie wartoôê 1, w innym przypadku 0. Od tej chwili wszystkie odwoîania
do zbioru "wyniki" spowodujâ zapisywanie danych w zbiorze o nazwie "roboczy",
chyba ûe zamkniesz go przez CLOSE).

1.66 options

OPTIONS (opcje, inne moúliwoôci) *** INSTRUKCJA
OPTIONS [NO] [FAILAT w1 | PROMPT w2 | RESULTS | CACHE]
gdzie: w1 -- wyraûenie.
Pozwala na ustawienie wartoôci poczâtkowych w programie.
Jeôli nie uûyjesz ûadnego ze sîów kluczowych -- wszystkie opcje
kontrolowane przez sîowa kluczowe zawarte w instrukcji
przyjmâ wartoôci wstëpnie ustawione (default). Poszczególne
sîowa kluczowe (pamiëtaj, ûe jeôli uûyjesz jednego z nich,
wówczas nie moûesz juû w tej samej instrukcji uûyê drugiego)
oznaczajâ: * FAILAT:w1 -- liczba caîkowita w1 okreôla poziom,
który spowoduje traktowanie kodu zwrotnego rozkazu jako bîëdu
(podobnie jak w ADOS),
* PROMPT w2 -- tu wyraûenie w2 jest îaïcuchem, który
bëdzie zastosowany jako wskaûnik gotowoôci systemu (prompt)
w instrukcjach PULL lub PARSE PULL,
* RESULTS -- "wskaûe" interpreterowi na koniecznoôê uûycia
îaïcucha wynikowego przy przekazywaniu rozkazów do programu
zarzâdzajâcego,

* CACHE -- uaktywnia wewnëtrznâ procedurë cache'owania (UWAGA:
wîâczona jest ona defaultowo przy starcie programu i przyspiesza pracë
interpretera. Przed kaûdym z tych sîów kluczowych moûesz uûyê sîowa NO,
które spowoduje przywrócenie defaultowej wartoôci dla danej opcji. Przykîad:
* OPTIONS FAILAT 10
(jeôli kod zwrotny instrukcji wynosi 10 lub wiëcej -- wystâpi komunikat o bîëdzie),
zaô
OPTIONS PROMPT "Slucham Cie moj Wladco?"
(spowoduje, ûe jeôli bëdzie potrzebna Twoja interwencja, komputer
w tak miîy dla oka sposób poprosi o îaskawe dotkniëcie klawiatury).

1.67 pragma

PRAGMA (zmiana atrybutów) *** FUNKCJA
PRAGMA (a$,[b$])
Znaczenie parametrów: a$ -- îaïcuch przedstawiajâcy opcjë wg poniûszego opisu,
b$ -- îaïcuch lub cyfra w zaleûnoôci od wybranej opcji.
Funkcja PRAGMA pozwala na zmianë atrybutów w programie zaleûnie od uwarunkowaï
ôrodowiska systemowego. Îaïcuch a$ -- okreôla atrybut i moûe to byê:
* D -- okreôla aktualny katalog, który bëdzie traktowany jako katalog
podstawowy dla zbiorów, które w nazwie nie bëdâ miaîy podanej nazwy
ôcieûki. Funkcja przyjmie jako wartoôê nazwë poprzedniego katalogu
podstawowego. Jeôli nie uûyjesz îaïcucha b$ -- wówczas funkcja przyjmie
nazwë ôcieûki, na której jesteô, bez naruszania nazwy obecnego katalogu,
* P -- okreôla nowy priorytet dostëpu. Teoretycznie moûesz uûyê liczby z
zakresu do -128 do 127, lecz w ARexxie zaleca sië, by najwyûszy priorytet
miaî zawsze program rezydentny, który ma defaultowo ustawiony priorytet nr 4.
Funkcja przyjmie wartoôê poprzednio ustawionego priorytetu,
* ID -- przyjmuje jako wartoôê 8-bajtowy îaïcuch heksadecymalny, który jest
adresem bloku dostëpu -- wykorzystywanym przy wywoîywaniu programu w ARexxie.
(UWAGA: dla kaúdego programu inny). Za pomocâ opcji ID moûesz nazwaê
program tym niepowtarzalnym identyfikatorem.
* Jeôli jako a$ uûyjesz îaïcucha o skîadni ('W',<'N' | 'WB'> -- wówczas
moûesz kontrolowaê dostëp do pola WindowPtr (które powoduje wyôwietlanie
komunikatów systemowych). Jeôli uûyjesz parametru N -- wtedy ûaden
komunikat nie ukaûe sië na ekranie, jeûeli WB -- bîëdy bëdâ sygnalizowane
w postaci requesterów znanych Ci z WB (i opisanych w tomie I), zaô gdy
uûyjesz îaïcucha o skîadni:
'*' [,a$]

pozwoli Ci to zdefiniowaê kanaî lub zbiór o nazwie a$ jako aktualnâ obsîugë
konsoli (dziëki czemu bëdzie moûliwe np. otwarcie dwóch kanaîów danych dla
jednego okna). Jeûeli opuôcisz nazwë -- wtedy obsîuga konsoli zostanie ustawiona
tak, jak w programie wewnëtrznym. Przykîad: SAY PRAGMA('D','df0:c) (jeôli do
tej pory byîeô w katalogu gîównym -- to w wyniku otrzymasz nazwë dyskietki
znajdujâcej sië w stacji df0:. Jednoczeônie od tego momentu katalogiem gîównym
stanie sië podkatalog rozkazowy C:),
SAY PRAGMA('Id') da w wyniku (przykîadowo) 00221AAB),
zaô
SAY PRAGMA('*',STDOUT) (przekaûe obsîugë konsoli na STDOUt).

1.68 procedure

PROCEDURE *** INSTRUKCJA
PROCEDURE [EXPOSE n1 [n2 n3 ... nn]]
gdzie: n1 - nn -- nazwy zmiennych.
Instrukcji PROCEDURE moûna uûyê wewnâtrz jakiejkolwiek z wbudowanych funkcji,
po to, aby stworzyê nowâ macierz nazw. Pozwoli to na zabezpieczenie przed
uszkodzeniem przy wykonywaniu funkcji nazw zdefiniowanych w otoczeniu wywoîania.
PROCEDURE powinna byê pierwszym wyraûeniem w "przeksztaîcanej" funkcji,
chociaû moûna jâ umieôciê w dowolnym miejscu tej funkcji.
UWAGA: Nie wolno uûywaê dwóch lub wiëcej instrukcji PROCEDURE wewnâtrz
tej samej funkcji. Jeôli uûyjesz sîowa kluczowego EXPOSE -- pozwoli Ci to na
dostëp do macierzy nazw i przekazanie zmiennych globalnie do funkcji.
Nazwy zmiennych wystepujâce po sîowie EXPOSE bëdâ odpowiadaê symbolom z macierzy.
Nazwy zmiennych przy EXPOSE mogâ byê nazwami zmiennych rdzeniowych lub zîoûonych.
Jeôli np. wartoôê zmiennej J wynosi 123, wówczas instrukcja
PROCEDURE EXPOSE J A.J (przekaûe do funkcji zmienne J i 123),
zaô
PROCEDURE EXPOSE A.J J (przekaûe A.J i J).
Jeôli uûyjesz w EXPOSE nazwy rdzeniowej -- wówczas wszystkie moûliwe do
utworzenia z niej zmienne zîoûone zostanâ "wziëte pod uwagë" przez EXPOSE,
np. jeôli masz zdefiniowane wczeôniej I, K, L -- wówczas PROCEDURE EXPOSE A.
(przekaûe A.I, A.K, A.L) itp. Przykîad:
SILNIA:
PROCEDURE ARG i IF i<=1 THEN RETURN 1 ELSE RETURN i*fact(i-1)
(obliczy silnië w zaleûnoôci od wartoôci i).

1.69 pull

PULL ([parse upper] pull -- przeanalizuj z wyciâganiem) *** INSTRUKCJA
PULL [a$ [,b$, ... , z$]
gdzie: a$ - z$ -- îaïcuchy.
Instrukcja czyta îaïcuch z konsoli, przekształca go na îaïcuch zîoûony
tylko z duûych liter i przekazuje go do wzorca, którym jest îaïcuch a$.
Jeôli w instrukcji uûyjesz wiëkszej liczby îaïcuchów-wzorców -- wówczas
zostanâ w ten sam sposób potraktowane kolejne îaïcuchy z konsoli.
Przykîad:
PULL pierwszy drugi
(przeczyta dwa îaïcuchy z konsoli i przekształci je w sposób opisany
powyûej).

1.70 push

PUSH (pchaj) *** INSTRUKCJA
PUSH [w1]
Parametr w1 jest wyraûeniem.
Instrukcja PUSH przygotowuje zbiór danych, który bëdzie wykorzystany przez
program zewnëtrzny lub ôrodowisko rozkazowe. Do wyniku otrzymanego z wyraûenia w1
zostanie dodany wysuw linii (czyli CR/LF inaczej [RETURN]) , a caîoôê zostanie
przekazana do stosu pamiëciowego lub do kanaîu STDIN.
UWAGA: Wiersze w stosie sâ umieszczane w kolejnoôci:
pierwszy na "dóî" stosu, kaûdy nastëpny "wyûej", np. po wykonaniu instrukcji:
PUSH wiersz 1
PUSH wiersz 2
PUSH wiersz 3
ze stosu zostanâ odczytane wartoôci (kolejno): "wiersz 3", "wiersz 2" i
"wiersz 1".
Instrukcja PUSH moûe byê uûywana tylko w odniesieniu do
urzâdzenia umoûliwiajâcego pracë interaktywnâ (np. CON: lub
PIPE:). Ponadto w katalogu L: musisz mieê nagrany handler DOS-u,
który zawiera w sobie rozkaz ACTION_STACK. Amiga DOS 1.3 nie
zawiera takiego rozkazu, jednak potrafi "zaakceptowaê" odpowiedni
handler przegrany z dyskietki WB 2.0. Instrukcja PUSH jest
bardzo wygodnym narzëdziem pozwalajâcym przekształciê kanaî STDIN
w podrëczny notatnik sîuûâcy do przygotowania danych. Moûesz na
przykîad poîâczyê kilka zbiorów w jeden w nastëpujâcy sposób:

* wczytaj kolejno te zbiory,
* wypchnij je (PUSH) do kanaîu STDIN, gdzie moûesz je dowolnie edytowaê
nie naruszajâc przy tym zawartoôci zbiorów na dyskietce ani nie tracâc
miejsca na niej na robocze kopie przed i po edycji,
* w momencie, gdy zawartoôê stosu zostanie wyczerpana -- kanaî STDIN
powraca do swojej zwykîej roli. Przypuôêmy, ûe chcesz skompilowaê i
zlinkowaê program wykorzystujâc stos. Wystarczy Ci wówczas wpisaê:
PUSH "blink c.o+main.o library amiga.lib to moj_program"
PUSH "cc main"

1.71 queue

QUEUE (kolejka) *** INSTRUKCJA
QUEQUE [w1]
gdzie: w1 -- to wyraûenie.
Instrukcja QUEQUE jest wîaôciwie identyczna jak PUSH, z dwoma wyjâtkami:
1) do obsîugi konieczny jest handler DOS-owski z rozkazem ACTION_QUEQUE
(opisany handler z WB 2.0 zawiera go takûe),
2) Wiersze ze stosu sâ zapisywane PRZED wierszami wpisanymi tam w sposób
interaktywny, zaô ZA wierszami wpisanymi przez PUSH -- natomiast czytane
sâ w odwrotnej kolejnoôci niû instrukcjâ PUSH. Zatem trzy wiersze:
QUEQUE wiersz 1
QUEQUE wiersz 2
QUEQUE wiersz 3
bëdâ odczytane w kolejnoôci "wiersz 1", "wiersz 2", "wiersz 3".
Analogicznie, aby uzyskaê wîaôciwy efekt kompilowania i linkowania
programu z przykîadu przy PUSH -- trzeba go napisaê "odwrotnie", czyli:
QUEQUE "cc main"
QUEQUE "blink c.o+main.o library amiga.lib to moj_program"

1.72 random

RANDOM (losowy) *** FUNKCJA
RANDOM ([a] [,b] [,c])
Parametry: a, b, c -- to liczby caîkowite.
Generuje pseudo-losowy numer z zakresu pomiëdzy liczbami a i b. Jeôli
opuôcisz te liczby -- wówczas a i b przyjmâ wartoôê (a=0, b=1000). Liczby a
i b mogâ byê dowolne przy zachowaniu nastëpujâcych warunków:
* a musi byê mniejsze niû b,

* róûnica b-a nie moûe przekraczaê 100.
Jeûeli potrzebne Ci sâ wiëksze wartoôci liczb losowych niû 1000, wówczas
musisz posîuûyê sië funkcjâ RANDU. Argumentu opcjonalnego c moûesz uûyê,
aby okreôliê tzw. ziarno generatora, co pozwoli na jeszcze wiëkszâ
losowoôê otrzymanej liczby. Przykîad:
SAY RANDOM (1,10)
(za kaûdym razem powinno wyjôê coô innego. Mnie teraz wyszîo 4).

1.73 randu

RANDU (rand[omize] u[niformed] value -- wartoôê losowa zuniformizowana) *** FUNKCJA
RANDU ([a])
gdzie: a -- to dowolna liczba z zakresu 1-256.
Generuje zuniformizowanâ liczbë pseudolosowâ z zakresu 0-1, która ma takâ
precyzjë, jaka jest ustawiona przez NUMERIC. RANDU Pozwala na uzyskanie
liczby losowej z dowolnego przedziaîu. Opcjonalny parametr a jest
"ziarnem" generatora. Przykîad:
NUMERIC DIGITS 8
SAY RANDU()
(mnie wyszîo 0.62185573), a teraz dajmy np.
NUMERIC DIGITS 2
SAY RANDU(42)
(tym razem u mnie jest 0.35). Aby uzyskaê wiëksze liczby losowe
zastosuj
SAY 1000*RANDU()
(wyszîo 5600, bo nie ustawiîem precyzji. Tobie moûe wyjôê coô innego.
Jeôli trafiasz za kaûdym razem tak samo jak ja -- oznacza to, ûe coô nie
jest w porzâdku albo z Amigâ, albo z Twojâ kopiâ ARexxa).

1.74 readch

READCH (read ch[aracters] -- czytaj znaki) *** FUNKCJA
READCH (a$,n)
Znaczenie parametrów: a$ -- îaïcuch, n -- liczba.
Tworzy îaïcuch zîoúony z n pierwszych znaków zbioru o nazwie a$. Jeôli
przed wczytaniem n znaków napotkano na znacznik koïca zbioru (EOF) --
operacja zostaje zakoïczona. Przykîad:
nazwa = READCH('moj_zbior',32)
SAY nazwa
(wydrukuje najwyûej 32 pierwsze znaki ze zbioru o nazwie "moj_zbior").

1.75 readln

READLN (read l[ogical until] "n[ewline = chr$(13) -- czytaj
znaki do momentu wystâpienia RETURN) *** FUNKCJA
READLN (a$)
gdzie: a$ -- îaïcuch.
Tworzy îaïcuch poprzez wczytywanie znaków ze zbioru o nazwie a$ do
momentu, gdy zostanie napotkany znak koïca wiersza (CR/LF). Znak ten nie
bëdzie umieszczony w tworzonym îaïcuchu. Przykîad:
nazwa = READLN('moj_zbior')
(nie wymaga chyba komentarza).

1.76 remlib

REMLIB (rem[ove] lib[rary] -- usuï bibliotekë) *** FUNKCJA
REMLIB (a$)
gdzie: a$ -- îaïcuch.
Usuwa z listy bibliotek programu rezydentnego bibliotekë o nazwie
a$. Za pomocâ funkcji REMLIB moûna takûe usunâê funkcje
zarzâdzajâce. Funkcja przyjmuje wartoôê "true" (1) wówczas, gdy
dana nazwa zostanie usuniëta z listy, zaô jeôli sâ z tym kîopoty
-- wówczas przyjmuje wartoôê 0.
Przykîad:
SAY REMLIB('bibl_wlasna.library')
(usuwa z listy bibliotek të o nazwie "Bibl_wlasna" i w razie sukcesu daje
w wyniku 1).

1.77 return

RETURN (powróê) *** INSTRUKCJA
RETURN w1
Parametr w1 -- jest wyraûeniem.
Pozwala na wyjôcie z funkcji i przekazanie kontroli do miejsca, z
którego funkcja zostaîa wywoîana. Jako wartoôê funkcji, z której
wychodzisz, zostanie przyjëta obliczona wartoôê wyraûenia.
Argument w1 moûesz opuôciê w przypadku, gdy funkcja, z której
wyskakujesz, zostaîa wywoîana przez CALL.
RETURN uûyte w programie podstawowym dziaîa identycznie jak EXIT.
Przykîad:
RETURN 6*2
(Wyjdzie z funkcji. Przy okazji -- jako wynik dziaîania instrukcji zostanie
przyjëte 12).

1.78 reverse

REVERSE (odwróê) *** FUNKCJA

REVERSE (a$)

gdzie: a$ -- to îaïcuch.

Tworzy nowy îaïcuch bëdâcy odwróceniem îaïcucha a$. Przykîad:

SAY REVERSE(’retsiniM’)

(wypisze na ekranie sîowo:"Minister").

1.79 right

RIGHT (prawy) *** FUNKCJA

RIGHT (a$,l [,p$])

gdzie: a$ -- îaïcuch, l -- liczba, p$ -- îaïcuch 1-znakowy.

Tworzy îaïcuch zîoûony z l ostatnich znaków îaïcucha a$. Jeôli l jest wiëksze niû dîugoôê îaïcucha a$, wówczas îaïcuch bëdzie uzupeîniony do potrzebnej dîugoôci przez dodanie spacji na poczâtku. Jeûeli posîuûyîeô sië argumentem p$ -- wówczas zamiast spacji bëdâ uûyte znaki, jakie znajdujâ sië w p$. Przykîad:

SAY RIGHT(’Kolobrzeg’,5)

(da w wyniku îaïcuch ’brzeg’), natomiast

SAY RIGHT(’123456’,7,’+’)

(stworzy îaïcuch ’+123456’).

SAY (powiedz) *** INSTRUKCJA (odpowiednik w BASIC-u -- PRINT)

SAY [w1]

gdzie: w1 -- wyraûenie.

Jak sië chyba zdâûyîeô domyôliê z dotychczasowych przykîadów -- instrukcja ta spowoduje wydrukowanie wyraûenia na konsoli (którâ zazwyczaj jest ekran). Kursor zostanie przesuniëty na poczâtek nowego wiersza. Jeôli nie uûyjesz parametru w1 -- wówczas zostanie wydrukowany "pusty" wiersz. Jeôli np. obliczyîeô wczeôniej sumë wydatków (równâ powiedzmy 500000) i postawiîeô jâ pod zmiennâ o nazwie "suma", wówczas:

SAY ’suma WYDATKOW =’ suma

(spowoduje wydruk na ekranie: suma WYDATKOW = 500000).

1.80 seek

SEEK (szukaj) *** FUNKCJA
SEEK (a$,l [, ’B’ | ’C’ | ’E’]
Parametrami sâ tu: a$ -- îaïcuch, l -- liczba.
Powoduje przejôcie do nowej pozycji w zbiorze o podanej nazwie
a$. Nowa pozycja jest okreôlona przez dodanie liczby l do pozycji
okreôlonej przez litery:
* B -- poczâtek zbioru,
* C = obecna pozycja w zbiorze (bëdzie ustawiona domyôlnie wówczas, gdy
pominiesz parametr B lub E),
* E -- koniec zbioru.
Funkcja przyjmuje wartoôê nowej pozycji liczonej od poczâtku
zbioru. Jeôli jest to np. zbiór programowy i "udaîo Ci" së
trafiê na poczâtek wiersza rozkazowego -- wtedy program bëdzie
wykonywany od tego rozkazu. Przykîad:
SAY SEEK(’moj_zbior’,20,’B’)
(przejdzie na 20. pozycjë w zbiorze "mój_zbior" i przyjmie wartoôê 20).
Jeôli teraz dasz
SAY SEEK(’moj_zbior’,64)
wówczas:
(przejdzie na 84. pozycjë w zbiorze "moj_zbior").
Jeûeli zbiór ma 756 pozycji, wtedy:
SAY SEEK(’moj_zbior’,0,’E’)
(da w wyniku 756).

1.81 select

SELECT (wybierz) *** INSTRUKCJA
SELECT
Rozpoczyna blok instrukcji zawierajâcych jednâ lub kilka
instrukcji WHEN i jednâ instrukcjë OTHERWISE. W przypadku
speînienia warunku nastëpujâcego po WHEN nastëpuje wyjôcie z
zakresu SELECT. Jeôli ûaden warunek nie zostanie speîniony,
wówczas bëdzie wykonany warunek z instrukcji OTHERWISE. Zakres
SELECT musi byê zakoïczony instrukcjâ END. Przykîad:
SELECT
WHEN i=1 THEN SAY ’JEDEN’
WHEN i=2 THEN SAY ’DWA’
OTHERWISE SAY ’TRZY’
END
(chyba pojâîeô, o co w tym chodzi?).

1.82 setclip

SETCLIP (set Clip[board List] -- wstaw na listë bufora) *** FUNKCJA
SETCLIP (a$ [,b$])
gdzie: a$,b$ -- to îaïcuchy.
Dodaje do listy bufora programu rezydentnego argumenty funkcji:
a$ -- jest to nazwa (najczëôciej rozkazu), b$ -- wartoôê
przyporzâdkowana nazwie (np. parametry rozkazu). Za pomocâ
SETCLIP moûesz takûe usunâê znajdujâce sië na liôcie nazwy --
przez podanie przy nazwie "pustego" îaïcucha b$. Jeôli podana
nazwa znajduje sië juû na liôcie, wówczas jej aktualna wartoôê
zostanie zastâpiona îaïcuchem b$. Funkcja przyjmuje wartoôê
"true" (1), gdy operacja sië powiodîa, lub 0 w przypadku
niepowodzenia. Przykîad:
SETCLIP ('SAY',")
(usunie rozkaz SAY z listy bufora programu rezydentnego,
przyjmujâc przy tym wartoôê (przykîadowo) = 1).

1.83 shell

SHELL *** INSTRUKCJA
SHELL [a$ | b] [w1]
Parametrami sâ tu: a$ -- îaïcuch, b -- nazwa zmiennej, w1 -- wyraûenie.
Instrukcja dziaîa identycznie jak instrukcja ADDRESS, tyle ûe nieco
szybciej. Przykîad:
SHELL edycja
(ustawi adres zarzâdzajâcy na edycja).

1.84 show

SHOW (pokaû) *** FUNKCJA
SHOW (
SÎOWO KLUCZOWE [,n$] [,p$])
gdzie: n$ -- to îaïcuch, p$ -- îaïcuch 1-znakowy.
Przeszukuje nazwy znajdujâce sië na liôcie opisanej przez sîowo
kluczowe, sprawdzajâc, czy istnieje tam nazwa podana przez
îaïcuch n$. Funkcja przyjmuje wartoôê 1 -- gdy podana nazwa
znajduje sië na liôcie, lub 0 -- gdy jej tam nie ma. Jeôli
opuôcisz parametr n$ -- wówczas zostanâ wypisane wszystkie nazwy

z danej listy -- oddzielone spacjâ, zaô jeûeli uûyîeô parametru
p$ -- oddzielone bëdâ one znakiem znajdujâcym sië w tym îaïcuchu.
Jako sîowa kluczowego moûesz uûyê:
* CLIP -- przeszukuje listë CLIP LIST,
* FILES -- przeszukuje listë nazw dotâd otwartych zbiorów,
* LIBRARIES -- przeszukuje listë LIBRARY LIST (nazwy bibliotek i funkcji
zarzâdzajâcych),
* PORTS -- przeszukiwana jest lista nazw portów. Przykîad:
SHOW (LIBRARIES, mathieeedoubbas.library)
(jeôli na liôcie bibliotek jest poszukiwana biblioteka, wówczas funkcja
przyjmie wartoôê 1), zaô
SHOW (LIBRARIES,,'*')
(wypisze na przykîad):
diskfont
icon
info
mathieeedoubbas
translator.

1.85 sign

SIGN (znak) *** FUNKCJA
SIGN (n)
Parametr n jest liczbâ.
Funkcja SIGN okreôla znak liczby n -- przyjmujâc wartoôci: 1 --
gdy liczba n jest dodatnia, -1 -- gdy jest ujemna i 0 -- gdy n=0.
Przykîad:
SAY SIGN(-55)
(da w wyniku -1)

1.86 signal

SIGNAL (sygnaî) *** INSTRUKCJA
Skîadnia 1):
SIGNAL < ON | OFF > w1
gdzie w1 jest warunkiem,
Skîadnia 2):
SIGNAL [VALUE] w2
gdzie: w2 jest wyraûeniem.

Instrukcja SIGNAL dziaîa zaleûnie od zastosowanej skîadni. W
przypadku skîadni 1 -- pokazuje stan zarzâdzania flagami przerwaï
wewnëtrznych. Jako warunku w1 -- moûesz uûyê:
* BREAK_C -- wykrywa przerwanie klawiatury spowodowane naciôniëciem [Ctrl]
i [C] jednoczeônie,
* BREAK_D -- wykrywa przerwanie typu [Ctrl]+[D],
* BREAK_E -- przerwanie typu [Ctrl]+[E], natomiast
* BREAK_F -- przerwanie typu [Ctrl]+[F],
* ERROR -- bada, czy rozkaz zarzâdzajâcy wygenerowaî niezerowy kod
zwrotny,
* HALT -- sprawdza, czy program zostaî zatrzymany przez rozkaz z programu
lub urzâdzenia zewnëtrznego,
* IOERR -- wykrywa zatrzymanie spowodowane przez bîâd na urzâdzeniu
wejôcia/wyjôcia,
* NOVALUE -- bada, czy uûyîeô niezainicjalizowanej zmiennej,
* SYNTAX -- sprawdza, czy wystâpiî bîâd skîadniowy. Powyûsze sîowa bëdâ
traktowane jako etykiety programowe, do których nastâpi skok w momencie
wykrycia przerwania spowodowanego odpowiedniâ przyczynâ. Przykîad:
SIGNAL ON ERROR
/* tu jest tekst Twojego programu */
...
/* tu koïczy sië tekst programu */
ERROR:
/* etykieta, podobnie jak w Amiga DOS, z dwukropkiem na koïcu */
/* a teraz przykîadowo moûe tu byê */
SAY 'tak byê niestety nie moûe [niewîaôciwy returncode]'
END
/* tu moûe sië znajdowaê dalsza czëôê programu */
UWAGA:
Etykieta o nazwie brzmiâcej tak samo jak warunek w SIGNAL -- MUSI
znajdowaê sië w programie -- w przeciwnym razie komputer moûe sië
zawiesiê.
Jeôli uûyjesz drugiej skîadni, wówczas zostanie obliczona wartoôê
wyraûenia w2, zaô instrukcja wygeneruje przerwanie natychmiastowe
powodujâce przeskok do etykiety o numerze równym obliczonej wartoôci
wyraûenia (coô jakby obliczalne GOTO w BASIC-u). Przykîad:
SIGNAL i*800
/*tu moûe byê dalszy ciâg programu */
800:

SAY ’i równa sië 1’
END
1600:
SAY ’i równa sië 2’
END
(w zaleûnoôci od wartoôci i -- gdy i bëdzie równe 1 -- nastâpi skok do etykiety
800, a gdy i bëdzie równe 2 -- do etykiety 1600.
UWAGA: nie wolno wykonywaê skoków do wnëtrza zakresu DO ... END, ani do wnëtrza
zakresu SELECT ... END. Jeôli nie chcesz naruszyê ôrodowiska wywoîania,
wówczas bezpiecznie jest uûyê instrukcji SIGNAL wewnâtrz funkcji wewnëtrznej.
Przy korzystaniu z instrukcji moûna posîugiwaê sië zmiennymi systemowymi:
* SIGL -- przechowuje aktualny numer wiersza, do którego nastâpiî skok
po uûyciu instrukcji SIGNAL,
* RC -- przyjmuje wartoôê kodu bîëdu, który spowodowaî przerwanie (w
przypadku ERROR -- kod zwrotny ADOS, w przypadku SYNTAX -- numer kodu bîëdu
ARexxa). Przykîadowo:
SIGNAL ON ERROR
(umoûliwia przerwania w przypadku wystâpienia bîëdu typu ERROR),
SIGNAL OFF SYNTAX
(uniemoûliwia przerwanie w przypadku wystâpienia bîëdu skîadniowego), zaô
SIGNAL START
(powoduje skok do etykiety START:).

1.87 sourceline

SOURCELINE (wiersz úródîowy) *** FUNKCJA
SOURCELINE([l])
gdzie: l -- to liczba.
Przyjmuje jako wartoôê tekst wiersza programowego wystëpujâcego
jako l-ty w wykonywanym programie ARexxa. Jeôli opuôcisz parametr
l -- wówczas wartoôciâ funkcji bëdzie liczba wierszy w programie.
Przykîadowo:
/* prosty przyklad */
SAY SOURCELINE ()
(da w wyniku 2),
SAY SOURCELINE(1)
(da w wyniku wydruk "/* prosty przyklad */
SPACE (przestrzeï, tu: miejsce) **** FUNKCJA
SPACE (a$,n [,p$])

gdzie: a$ -- to îaïcuch, n -- liczba, p$ -- îaïcuch 1-znakowy.
Przekształca îaïcuch a$ w ten sposób, ûe pomiędzy kaûdym sîowem îaïcucha
umieszcza n spacji (lub znaków uûytych we wzorcu p$. Przykîad:
SAY SPACE('Commodore Amiga 500',2,'*,)
(da w wyniku 'Commodore**Amiga**500), zaô
SAY SPACE('COMMODORE AMIGA 500',0)
(utworzy îaïcuch 'COMMODOREAMIGA500').

1.88 storage

STORAGE (skîadowanie) **** FUNKCJA
STORAGE ([a$] [,b$] [,l] [,p$])
Parametrami sâ tu: a$,b$ -- îaïcuchy, l -- liczba, p$ -- îaïcuch 1-znakowy.
Funkcja uûyta bez argumentów poda aktualnie dostëpnâ iloôê
pamiëci. Zastosowanie parametru a$ (4-bajtowy adres) spowoduje
przekopiowanie danych z îaïcucha b$ -- do adresu a$. Parametr l
okreôla liczbë kopiowanych bajtów (defaultowo przyjëta jest
dîugoôê îaïcucha b$). Jeôli l jest wiëksze niú dîugoôê îaïcucha
wówczas jest on uzupeîniany na poczâtku zerami ('00'X) lub
znakami umieszczonymi we wzorcu p$. Wartoôciâ funkcji bëdzie
poprzednia zawartoôê pamiëci. UÛYWAJ TEJ FUNKCJI OSTROÛNIE.
Moûesz "niechcâcy" zniszczyê zawartoôê pamiëci. Wprawdzie moûna
odzyskaê jâ (dziëki temu, úe funkcja "pamiëta" starâ zawartoôê),
ale co bëdzie jeôli np. nowe dane w pamiëci spowodujâ zawieszenie
sië komputera? Jak wtedy odczytasz wartoôê funkcji? UWAGA:
Podczas wykonywania tej funkcji nie wolno przeîâczaê dostëpów, bo
moûe spowodowaê to problemy, wówczas gdy îaïcuch b$ jest bardzo
dîugi. Jeôli opuôcisz nazwë îaïcucha b$, wtedy w pamiëci zostanie
zapisane l zer lub l znaków uûytych w parametrze p$ -- poczynajâc
od komórki o adresie a$. Przykîad:
SAY STORAGE()
(wypisze iloôê wolnej pamiëci (np. 226500). Inny przykîad:
pam = STORAGE('0004 0000'X,'ROZWIAZANIE')
(w komórce 40000 i dalszych wpisze 'ROZWIAZANIE'. Stara zawartoôê
przepisanych komórek bëdzie zapamiëtana w zmiennej pam). Jeôli teraz
napiszesz:
SAY pam
wówczas otrzymasz w wyniku starâ zawartoôê tych komórek, przykîadowo:
'SOLUTION IS'
CALL STORAGE ('0004 0000'X,,32,'-')
(wypeîni komórkë 40000 i dalsze -- 32 znakami "-")

1.89 strip

STRIP (rozbierz) *** FUNKCJA
STRIP (a$ [,op] [,p$])
gdzie: a$ -- îaïcuch, p$ -- îaïcuch 1-znakowy, op -- argument wg opisu
poniûej.
Jeôli w funkcji STRIP uûyjesz wyîâcznie argumentu a$, wówczas zostanâ z
niego usuniëte spacje poprzedzajâce lub wystëpujâce na koïcu. Argument op
moûe przybieraê wartoôci:
* L -- co usunie spacje poprzedzajâce îaïcuch,
* T -- usunie spacje wystëpujâce na koïcu îaïcucha,
* B -- usunie i jedne i drugie spacje.
Jeôli uûyjesz argumentu p$, wówczas zamiast spacji zostanâ usuniëte
poczâtkowe i koïcowe znaki, jakie wystëpujâ we wzorcu p$.
Przykîad: (znak "_" tu oznacza spacje):
SAY STRIP('___usuwanie spacji________')
(da w wyniku 'usuwanie spacji'),
SAY STRIP('______usun spacje z przodu_______','L')
(da w wyniku 'usun spacje z przodu______'),
zaô
SAY STRIP('+++12+3++','B','+')
(da w wyniku '12+3'). Zauwaû, ûe znak + wewnâtrz îaïcucha nie zostaî usuniëty.

1.90 substr

SUBSTR (sub str[ing] -- podîaïcuch, tu: fragment îaïcucha) *** FUNKCJA
SUBSTR (a$,s [,l] [b$])
gdzie: a$ -- îaïcuch, s,l -- liczby, p$ -- îaïcuch 1-znakowy.
Tworzy îaïcuch zîoûony z fragmentu îaïcucha a$ zaczynajâcy sië od pozycji
s i o dîugoôci l (jeôli l jest opuszczone, wówczas defaultowo przyjëte
zostanie jako równe dîugoôci îaïcucha a$ liczonej od pozycji s do koïca).
Jeôli l jest wiëksze niú dîugoôê "reszty" îaïcucha a$, wówczas îaïcuch
zostanie uzupeîniony na koïcu spacjami. Jeûeli uûyîeô parametru p$, wtedy
spacje zostanâ zastâpione znakiem, jaki jest uûyty we wzorcu p$. Przykîad:
SAY SUBSTR('1234567',3)
(da w wyniku îaïcuch '34567'),
SAY SUBSTR('Commodore Amiga 500',10,11,'!')
(da w wyniku 'Amiga 500!!).

1.91 subword

SUBWORD (podsîowo -- tu: kilka sîów z îaïcucha) *** FUNKCJA
SUBWORD (a$,n [,l])
Parametrami funkcji sâ: a$ -- îaïcuch, n,l -- liczby.
Tworzy îaïcuch zîoûony z l sîów rozpoczynajâc od n-tego sîowa îaïcucha
a$. Jeôli opuôcisz l, wówczas pod uwagë (domyôlnie) brane sâ wszystkie
pozostaîe sîowa w îaïcuchu a$. Spacje poczâtkowe i koïcowe zostanâ
odrzucone. Przykîad:
SAY SUBWORD('Wlazl kotek na plotek i mruga',4)
(da w wyniku îaïcuch:'plotek i mruga), natomiast (przyjmujâc ûe znak "_"
oznacza spacjë):
SAY SUBWORD('____ladna_to_piosenka__niedluga_____________',3,2)
(da w wyniku 'piosenka niedluga')

1.92 symbol

SYMBOL (nazwa zmiennej) *** FUNKCJA
SYMBOL (a$)
gdzie: a$ -- to îaïcuch.
Sprawdza, czy argument a$ jest prawidîowâ nazwâ zmiennej ARexxa. W
zaleûnoôci od wyniku badania przyjmuje wartoôci:
* VAR -- gdy zmienna jest zainicjalizowana (tzn., pod którâ podstawiono
jakâô wartoôê),
* LIT -- gdy badana zmienna jest nie zainicjalizowana (juû zdefiniowana, ale
jeszcze nie podstawiona),
* BAD -- gdy nazwa sprawdzanej zmiennej (z punktu widzenia ARexxa) jest
nieprawidîowa. Jeôli np. podstawiîeô wartoôê 4 pod K, wówczas:
SAY SYMBOL('K')
(da w wyniku VAR), zaô
SAY SYMBOL('n')
(da w wyniku LIT), natomiast:
SAY SYMBOL('*2*')
(da w wyniku BAD).

1.93 time

TIME (czas) FUNKCJA

TIME ([op])

gdzie: op -- nazwa opcji wg opisu poniûej.

Podaje czas zegara systemowego w formacie zaleûnym od uûytej opcji.

Moûliwe opcje:

* () [opcja "pusta"] -- czas bëdzie podany w formacie GG:MM:SS (godziny, minuty,sekundy),

* E -- poda czas, jaki minâî od ostatniego uûycia funkcji TIME lub od wyzerowania zegara w sekundach,

* H -- czas w caîych godzinach, jaki upîynâî od póînocy,

* M -- czas, jaki minâî od póînocy w minutach,

* S -- czas od póînocy w sekundach,

* R -- spowoduje zerowanie zegara.

Przypuôêmy, ûe w tej chwili jest godzina 2:05 po póînocy. Wówczas:

SAY TIME('H')

(da w wyniku 2),

SAY TIME('M')

(da w wyniku 125),

SAY TIME()

(da w wyniku 02:05:00),

SAY TIME('R')

wyzeruje zegar Amigi, zaô jeôli po wpisaniu tego ostatniego wiersza napiszesz:

SAY TIME('E')

(otrzymany wynik bëdzie zaleûaî od Twojego refleksu i szybkoôci pisania. U mnie byîo 1.04. [znam lepszych "koderów"]).

1.94 trace

TRACE (ôledzenie) **** FUNKCJA

TRACE (op)

gdzie: op -- dowolna nazwa îaïcuchowa lub przedrostkowa wedîug opisu poniûej. Funkcja TRACE ustawia tryb ôledzenia na opisany opcjâ. Opcje mogâ byê nastëpujâce:

OPCJE ALFABETYCZNE:

* ALL -- ôledzenie caîego programu,

* BACKGROUND -- wyîâcza ôledzenie zarówno programowe, jak i interaktywne,

* COMMANDS -- wszystkie rozkazy sâ ôledzone przed wysîaniem ich do programu zarzâdzajâcego. Wszystkie niezerowe kody zwrotne sâ wyôwietlane na ekranie,
* ERROR -- ôledzone sâ tylko te rozkazy, które generujâ niezerowy kod zwrotny. Ôledzenie nastëpuje po wykonaniu rozkazu i generacji niezerowego kodu zwrotnego,
* INTERMEDIATES -- ôledzony jest caîy program. Wszystkie rezultaty poôrednie sâ wyôwietlane na ekranie,
* LABELS -- ôledzone i wyôwietlane na ekranie sâ wszystkie etykiety z programu,
* NORMAL (default) -- ôledzone i wypisywane na ekranie sâ te rozkazy, które generujâ kod zwrotny przekraczajâcy dopuszczalny. Ôledzenie ma miejsce po wykonaniu rozkazu,
* OFF -- wyîâcza ôledzenie programowe. Ôledzenie interaktywne jest moûliwe,
* RESULTS -- ôledzony jest caîy program. Wszystkie wyniki -- poôrednie i koïcowe sâ wyôwietlane na ekranie,
* SCAN -- przeôledzony jest caîy program. Wykonanie programu jest zatrzymane. Ewentualnie znalezione bîëdy sâ wypisywane na ekranie.

OPCJE PRZEDROSTKOWE:

Przedrostek skîada sië z 3 znaków. Przy zastosowaniu tych opcji ôledzony jest caîy program. W zaleûnoôci od uûytego przedrostka na ekranie wypisywane sâ tylko:

+++ -- bîëdy skîadniowe,
>C> -- rozszerzenia nazw zmiennych zîoûonych,
>F> -- wyniki wywoîywanych funkcji,
>L> -- wyraûenia etykietowane,
>O> -- wyniki operacji dwuwartoôciowych,
>P> -- wyniki operacji przedrostkowych,
>U> -- nazwy zmiennych nie zainicjalizowanych,
>V> -- wartoôci zmiennych,
>>> -- wyniki wyraûeï,
>.> -- wartoôci zmiennych oznaczonych za pomocâ symbolu kropki (np. rdzeniowych).

Jeôli chcesz to wszystko sprawdziê -- wpisz sobie dowolny program (im dîuûszy i bardziej rozbudowany, tym lepiej), a nastëpnie uûyj TRACE (na poczâtek najlepiej jest zastosowaê TRACE RESULTS). Zamiast tej ostatniej instrukcji moûesz uûyê po prostu instrukcji opisanej niûej:

TRACE **** INSTRUKCJA

TRACE

Wîâcza ôledzenie interaktywne caîoôci programu bez wyôwietlania wartoôci poôrednich.

1.95 translate

TRANSLATE (przetîumacz) **** FUNKCJA
TRANSLATE (a$ [,b$] [,c$] [,p$])
gdzie: a$,b$,c$ -- to îaïcuchy, p$ -- îaïcuch 1-znakowy.
Tworzy macierz tîumaczeï i zamienia okreôlone parametrami znaki w
îaïcuchu a$. Jeôli îaïcuch a$ jest jedynym parametrem, wówczas
zostanie on zamieniony na îaïcuch skîadajâcy sië wyîâcznie z
duûych liter. Jeôli uûyjesz parametru, c$ wówczas macierz
tîumaczeï zostanie zmodyfikowana w ten sposób, ûe znaki z
îaïcucha a$, które wystëpujâ równieû w îaïcuchu c$, zostanâ
zamienione przez znaki wystëpujâce na tych samych pozycjach w
îaïcuchu b$. Jeôli îaïcuch b$ jest krótszy niû a$ -- znaki
powtarzajâce sië w îaïcuchach a$ i c$ zostanâ zastâpione spacjami
(lub w przypadku uûycia parametru p$ -- znakiem wystëpujâcym w
îaïcuchu p$). UWAGA: w tej funkcji îaïcuchy musisz podaê w
cudzysîowie. Îaïcuch wynikowy zawsze bëdzie miaî dîugoôê îaïcucha
a$ -- niezaleûnie od dîugoôci îaïcuchów b$ i c$. Przykîad:
SAY TRANSLATE("abcde","123","cbade","*")
(da w wyniku ’321**’),
SAY TRANSLATE("male litery")
(da w wyniku "MALE LITERY"), zaô:
SAY TRANSLATE("0110","10","01")
(da w wyniku "1001").

1.96 trim

TRIM (okrój) **** FUNKCJA
TRIM (a$)
Parametr a$ to dowolny îaïcuch.
Usuwa spacje wystëpujâce na koïcu îaïcucha a$. Jeûeli przyjmiemy, ûe znak
"_" oznacza spacjë, wówczas:
SAY
LENGTH(TRIM(’_____123_____’))
(da w wyniku 8).

1.97 trunc

TRUNC (trunc[ate] -- zaokrâgl przez odciëcie) **** FUNKCJA
TRUNC (l [,n])
przy czym: l -- to liczba, zaô n -- liczba caîkowita).
Funkcja "zaokrâgla" liczbë l przez odciëcie od niej okreôlonej
parametrem n cyfry. Jeôli opuôcisz parametr n, wtedy wszystkie
miejsca dziesiëtne liczby l zostanâ odciëte. Jeôli uûyjesz n --
wówczas liczba bëdzie miaîa tyle miejsc po przecinku, ile wynosi
n. W razie potrzeby wynik bëdzie uzupeîniony zerami na koïcu.
Przykîad:
SAY TRUNC(123.4567)
(da w wyniku 123), natomiast
SAY TRUNC(123.4567,6)
(da w wyniku 123.456700).

1.98 upper

UPPER ([translate to] upper[case] -- przeîóû na duûe litery) **** FUNKCJA
UPPER(a$)
gdzie: a$ -- to îaïcuch literowy.
Funkcja UPPER dziaîa identycznie jak funkcja TRANSLATE, jednak
jest o wiele szybsza jeôli îaïcuch a$ nie przekracza 32 liter.
UWAGA: zastosowanie cyfr w îaïcuchu a$ w tym przypadku powoduje
bîâd. Przykîad:
SAY TRANSLATE("amiga jest ok")
(da w wyniku "AMIGA JEST OK").

1.99 value

VALUE (wartoôê) **** FUNKCJA
VALUE (n$)
Parametr n$ -- jest nazwâ zmiennej.
Podaje wartoôê zmiennej o nazwie n$. Jeûeli np. zmienna k ma wartoôê 24,
wówczas:
SAY VALUE('k')
(da w wyniku 24).

1.100 verify

VERIFY (sprawdû) **** FUNKCJA
VERIFY (a$,b$ [’M’])
gdzie: a$, b$ -- sâ îaïcuchami.
Jeôli opuôcisz parametr M -- wówczas funkcja obliczy pozycjë
pierwszego znaku w îaïcuchu a$, który nie wystëpuje w îaïcuchu b$
(jeôli w îaïcuchu b$ bëdâ wystëpowaê wszystkie znaki wystëpujâce
w îaïcuchu a$ -- wówczas funkcja przyjmie wartoôê = 0). Przy
zastosowaniu parametru M -- funkcja obliczy pozycjë pierwszego
znaku w îaïcuchu a$, który wystëpuje w îaïcuchu b$ (lub 0 --
jeôli nie bëdzie ûadnego takiego znaku). Przykîad:
SAY VERIFY(’123456’,’0123456789’)
(da w wyniku 0),
SAY VERIFY(’123x56’,’0123456789’)
(da w wyniku 4), zaô
SAY VERIFY(’1234x6’,’uvxyz’,’M’)
(da w wyniku 5), natomiast:
SAY VERIFY(’123456’,’uvxyz’,’M’)
(da w wyniku 0).

1.101 when

WHEN (jeôli) *** INSTRUKCJA
WHEN w1 [THEN] [;] [w2]
(gdzie: w1, w2 -- to wyraûenia).
Instrukcja WHEN jest bardzo podobna do instrukcji IF, ale moûe
wystëpowaê tylko w zakresie wyznaczonym instrukcjâ SELECT.
Ponadto w trakcie sprawdzania warunku przyjmuje wartoôci "true"
(1) lub "false" (0) i w zaleûnoôci od nich -- po sprawdzeniu
skacze do instrukcji END (gdy = 1) lub do nastëpnego wiersza (gdy
= 0). Po instrukcjach WHEN musi wystëpowaê OTHERWISE. Przykîad:
SELECT
WHEN i<j THEN SAY ’i jest mniejsze od j’
WHEN i>j THEN SAY ’i jest wiëksze od j’
OTHERWISE SAY ’i rowna sie j’
END
(komentarz chyba zbyteczny).

1.102 word

WORD (sîowo) *** FUNKCJA

WORD (a$,n)

gdzie: a$ -- to îaïcuch, zaô n -- liczba.

Funkcja przyjmuje wartoôê n-tego sîowa îaïcucha a$. Jeôli w îaïcuchu jest

mniej niû n sîów -- wówczas tworzy îaïcuch "pusty". Przykîad:

SAY WORD('Wlazl kotek na plotek',2)

(da w wyniku 'kotek').

1.103 wordindex

WORDINDEX (pozycja sîowa) **** FUNKCJA

WORDINDEX (a$,n)

Parametrami sâ tu: a$ -- îaïcuch, n -- liczba.

Oblicza pozycjë, na której w îaïcuchu a$ wystëpuje pierwszy znak

n-tego sîowa îaïcucha a$. Jeôli liczba sîów îaïcucha jest

mniejsza od n -- wówczas funkcja przyjmuje wartoôê = 0. UWAGA:

Przy obliczaniu pozycji spacje sâ równieû brane pod uwagë.

Przykîad:

SAY WORDINDEX('Wlazl kotek na plotek',3)

(da w wyniku 13), natomiast:

SAY WORDINDEX('Wlazl kotek na plotek',6)

(da w wyniku 0).

1.104 wordlenght

WORDLENGHT (dîugoôê sîowa) **** FUNKCJA

WORDLENGHT (a$,n)

gdzie: a$ -- to îaïcuch, n -- liczba.

Podaje dîugoôê n-tego sîowa w îaïcuchu a$. Jeôli n jest wiëksze

niû liczba sîów w îaïcuchu a$, wówczas przyjmuje wartoôê 0.

Przykîad:

SAY WORDLENGTH ('WLAZL KOTEK NA PLOTEK,3)

(da w wyniku 2), zaô:

SAY WORDLENGTH ('WLAZL KOTEK NA PLOTEK,7)

(da w wyniku 0).

1.105 words

WORDS (liczba sîów) *** FUNKCJA
WORDS (a$)
Parametr a$ jest îaïcuchem.
Podaje liczbë sîów w îaïcuchu a$. Przykîad:
SAY WORDS(’ile tez tu moze byc slow’)
(da w wyniku 6).

1.106 writech

WRITECH (write ch[aracters] -- pisz znaki) *** FUNKCJA
WRITECH (a$,b$)
gdzie: a$, b$ -- to îaïcuchy.
Zapisuje îaïcuch a$ do zbioru o nazwie b$ -- przyjmujâc jako wartoôê liczbë
zapisanych znaków (spacje sâ wliczane). Przykîad:
SAY WRITECH(’moj_zbior’,’dane pomocnicze’)
(zapisze îaïcuch "moj_zbior" do zbioru o nazwie "dane pomocnicze" i
przyjmie przy tym wartoôê 9).

1.107 writeln

WRITELN (write [and add "new]l[i]n[e character] -- pisz i doîâcz kod
[RETURN]) *** FUNKCJA
WRITELN (a$,b$)
przy czym: a$, b$ -- to îaïcuchy.
Funkcja WRITELN dziaîa identycznie jak opisany powyûej WRITECH, z
tym ûe do îaïcucha b$ dodaje kod koïca wiersza (CR/LF=RETURN)
wliczany do liczby pisanych znaków. Przykîad:
SAY WRITELN(’moj_zbior’,’dane pomocnicze’)
(zapisze îaïcuch "moj_zbior" do zbioru "dane pomocnicze i przyjmie
wartoôê 10).

1.108 x2c

X2C ([convert from he]x [to] c[haracter string] -- przeksztaîê
îaïcuch heksadecymalny w znakowy) *** FUNKCJA
X2C (a$)
gdzie: a$ -- to îaïcuch heksadecymalny.

Przekształca îaïcuch heksadecymalny w jego odpowiednik znakowy
(w postaci spakowanej -- bez spacji). Jeôli wartoôê heksadecymalna mieôci sië
w zakresie kodów ASCII -- wówczas îaïcuch zostanie przekształcony w
jego odpowiednik ASCII (uwaga wg kodów hex). Przykîad:
SAY X2C('12AB')
(da w wyniku '12AB'), zaô
SAY X2C('40 FF')
(da w wyniku '40FF'),
SAY X2C('62')
(da w wyniku 'b').

1.109 x2d

X2D ([convert from he]x [to] d[ecimal number] -- przekształê îaïcuch
heksadecymalny w liczbë dziesiëtnâ) *** FUNKCJA
X2D (a$ [,n])
gdzie: a$ -- to îaïcuch heksadecymalny, n-liczba.
Przekształca îaïcuch heksadecymalny w liczbë dziesiëtnâ dodajâc n miejsc
po przecinku (domyôlnie -- 0). Przykîad:
SAY X2D('0A')
(da w wyniku 10), zaô
SAY X2D('0A',3)
(da w wyniku 10.000).

1.110 xrange

XRANGE ([he]x [numbers] range -- zakres liczb heksadecymalnych) *** FUNKCJA
XRANGE ([x1] [,x2])
Parametry: x1, x2 -- to liczby heksadecymalne z zakresu 00 do FF (przy czym
x1 musi byê mniejsze niû x2).
Tworzy îaïcuch zawierajâcy wszystkie liczby heksadecymalne
znajdujâce sië pomiëdzy x1 a x2 wîâcznie. Defaultowo x1=00,
x2=FF. UWAGA: Do zakresu liczy sië tylko pierwszy znak z x1 i x2, np.:
XRANGE ('1F'X,'F3'X)
(utworzy îaïcuch '101112..0F', a nie jak mogîoby sië wydawaê '1F2021...F3').
Funkcjë XRANGE moûna teû wykorzystywaê do tworzenia îaïcucha znakowego
zawierajâcego litery A...F, np.:
XRANGE('A','F')
(utworzy îaïcuch 'ABCDEF').
Jeôli opuôcisz parametry x1 i x2 -- wówczas zostanie stworzony -- utworzy
caîy îaïcuch '00010203....FBFCFDFEFF).

1.111 allocmem

ALLOCMEM (allocate memory -- przealokuj pamiëê)
ALLOCMEM (l [,a$])
gdzie: l -- to liczba, a$ -- îaïcuch 4-bajtowy.
Alokuje blok pamiëci o dîugoôci l z obszaru wolnej pamiëci systemowej i oblicza jego adres jako îaïcuch 4-bajtowy. Opcjonalny parametr a$ (atrybut) musi byê standardowâ flagâ alokacji dla pamiëci EXEC. Defaultowo przyjmowana jest pamiëê typu PUBLIC. Zalecam uûywaê tego rozkazu jedynie przy alokowaniu pamiëci na potrzeby programów zewnëtrznych, w przeciwnym razie mogâ wyjôê komplikacje.
Przykîad:
SAY C2X(ALLOCMEM(1000))
(da w wyniku adres 50000 (przykîadowo)).

1.112 closeport

CLOSEPORT (zamknij port)
CLOSEPORT (a$)
gdzie: a$ -- îaïcuch.
Zamyka port komunikatów o nazwie podanej przez îaïcuch a$. Port taki musi byê wczeôniej przealokowany (np. przez wywoîanie rozkazu OPENPORT) z tego samego programu ARexxa. Wszystkie komunikaty z zamykanego portu (poza tymi, które zostaîy potraktowane rozkazem REPLY) w momencie zamykania portu otrzymajâ kod zwrotny bîëdu = 10. Przykîad:
CALL CLOSEPORT moj_port
(zamknie (otwarty wczeôniej) port komunikatów o nazwie "moj_port).

1.113 freemem

FREEMEM (free mem[ory] -- wolna pamiëê)
FREEMEM (a$,l)
Parametrami rozkazu sâ: a$ -- îaïcuch 4-bajtowy, l - liczba.
Zwalnia blok pamiëci o podanej dîugoôci rozpoczynajâcy sië od adresu a$ do systemu. Wartoôê adresu moûna otrzymaê wywoîujâc bezpoôrednio ALLOCMEM. UWAGA: jeôli pamiëê zostaîa wczeôniej przealokowana przez GETSPACE -- nie wolno uûywaê rozkazu FREEMEM. Obliczana jest równieû wartoôê logiczna operacji (1 -- gdy sië uda, 0 -- gdy nie). Przykîad:
SAY FREEMEM('00042000'X,32) (da w wyniku (np.) 1, zaô blok pamiëci zaczynajâcy sië od adresu hexadecymalnego 42000 zostanie zwolniony. Uwaga: Nieumiejëtne zwalnianie pamiëci pozwoli Ci co prawda "szpanowaê" tym, ûe masz jej "wiëcej niû ustawa (i zamontowane koôci) przewiduje", ale moûe wprowadziê duûe zamieszanie przy obsîudze innych, niearexxowskich programów).

1.114 getarg

GETARG (get arg[ument -- pobierz argument)
GETARG (a$ [,n])
gdzie: a$ -- to îaïcuch 4-bajtowy, n -- liczba.
Pobiera rozkaz, nazwë funkcji lub îaïcuch z pakietu o adresie a$. A$ musi byê
4-bajtowym adresem pakietu komunikatów uzyskanym przez bezpoôrednie wywoîanie
GETPKT. Liczba n oznacza szczelinë, z której ma byê pobrany argument mniejszy
lub równy wartoôci licznika argumentów w pakiecie. UWAGA: Rozkazy i nazwy
funkcji znajdujâ sië zawsze w szczelinie 0, pozostaîe szczeliny w pakiecie
numerowane sâ od 1 do 15. Przykîad:
SAY GETARG('0005 0000',0)
(da w wyniku (np.) 'allocmem').

1.115 getpkt

GETPKT (get p[ac]k[e]t -- pobierz pakiet)
GETPKT (a$)
gdzie: a$ -- îaïcuch.
Sprawdza port komunikatów o nazwie a$. Port ten musi byê wczeôniej otwarty przez
bezpoôrednie wywoîanie OPENPORT z wykonywanego programu w ARexxie. Zostaje
obliczony 4-bajtowy adres pierwszego pakietu w porcie lub '0000 0000', gdy w
porcie nie ma ûadnego pakietu. Przykîad:
SAY GETPKT('Moj_port')
(da w wyniku (n.p.) '0005 0000').

1.116 openport

OPENPORT (otwórz port)
OPENPORT(a$)
gdzie: a$ -- îaïcuch.
Tworzy port komunikatów o nazwie a$. Jeûeli port zostaî utworzony -- przyjmuje
wartoôê "true" (1), w przeciwnym wypadku (np. gdy port o podanej nazwie juû
zostaî otworzony lub bit sygnalny nie moûe byê przealokowany) -- przyjmuje
wartoôê "false" (0). Stworzony port komunikatów jest linkowany do struktury
danych ogólnych programu. Port moûna zamknâê przez CLOSEPORT lub przez wyjôcie z
programu. Przykîad:
SAY OPENPORT('Moj_Port')
(da w wyniku (przykîadowo) 1).

1.117 reply

REPLY (powtórz)

REPLY (a$,l)

gdzie: a$ -- to îaïcuch 4-bajtowy, l -- liczba caîkowita.

Zwraca pakiet komunikatów do "miejsca nadania" ustawiajâc w polu wynikowym pakietu jako pierwszy wynik -- wartoôê l. Wynik wtórny zostanie wyzerowany.

Przykîad:

CALL REPLY ’0005 0000’,20

(wygeneruje zwrotny kod bîëdu = 20)

1.118 showdir

SHOWDIR (show dir[ectory] -- pokaû katalog

SHOWDIR (a$ [,SK])

Parametrami sâ tu: a$ -- îaïcuch, SK -- sîowo kluczowe wedîug opisu poniûej.

Podaje zawartoôê katalogu o nazwie oznaczonej îaïcuchem a$. Parametr SK wskazuje, co ma byê pokazane. Moûe to byê:

* ALL -- caîy katalog,

* F -- tylko nazwy zbioru,

* D -- tylko podkatalogi. Domyôlnie ustawiony jest parametr F.

Przykîad:

SAY SHOWDIR ("df1:c,’ALL’)

(da w wyniku, przykîadowo): copy ed Shell rx ts tc).

1.119 showlist

SHOWLIST (pokaû listë)

SHOWLIST (<’D’ | ’L’ | ’W’ | ’R’ | ’P’> [,a$])

gdzie: a$ -- to îaïcuch, a pozostaîe parametry sâ opisane dalej.

Jeôli w rozkazie pominiesz parametr a$, wówczas zostanie przeszukana odpowiednia lista (w zaleûnoôci od parametru):

* D -- lista urzâdzeï,

* L -- lista bibliotek,

* P -- lista portów,

* R -- lista Ready (gotów),

* W -- lista Wait (oczekujâcy),

zaô znajdujâce sië na liôcie nazwy utworzâ îaïcuch, w którym nazwy te bëdâ oddzielone spacjami. Jeôli uûyjesz dodatkowo parametru a$, wtedy bëdzie

poszukiwana nazwa identyczna z a$. W przypadku gdy znajduje sië na danej liôcie
-- zostanie przyjëta wartoôê "true" (1), w przeciwnym wypadku -- "false" (0).
UWAGA: Podczas przeszukiwania nie wolno zamieniaê dostëpów. Przykîad:
SAY SHOWLIST('P')
(da w wyniku (przykîadowo) REXX
Moj_Port Twoj_Port
a jeôli teraz sprawdzisz listë portów przez:
SAY SHOWLIST('P','REXX'
to otrzymasz w wyniku 1.

1.120 statef

STATEF (stat [us of] e[xternal] f[ile] -- stan zbioru zewnëtrznego)
STATEF (a$)
gdzie: a$ -- îaïcuch (tu musi byê umieszczony w cudzysîowie).
Podaje informacje dotyczâce zbioru o nazwie a$ -- w postaci îaïcucha
o formacie: <DIR|FILE> a b c d
gdzie:
* DIR wystëpuje w przypadku, gdy a$ jest nazwâ katalogu,
* FILE -- gdy a$ jest nazwâ zbioru,
natomiast pozostaîe parametry oznaczajâ:
* a -- dîugoôê zbioru,
* b -- iloôê zajmowanych bloków na dysku,
* c -- rodzaj protekcji,
* d -- komentarz (jeôli jest wpisany do katalogu).
Przykîad:
SAY STATEF("libs:rexxsupport.library)
(da w wyniku (przykîadowo) 'FILE 1880 4 RWED ').

1.121 waitpkt

WAITPKT (wait [for message from p[ac]k[e]t -- czekaj na komunikat
z pakietu)
WAITPKT (a$)
gdzie: a$ -- to îaïcuch.
Czeka na komunikat z pakietu znajdujâcego sië w porcie o nazwie a$. Port musi
byê wczeôniej otwarty przez bezpoôrednie wywoîanie za pomocâ OPENPORT z
aktualnie wykonywanego programu ARexxa. Jeôli pakiet znajduje sië w porcie,
wówczas zostanie przyjëta wartoôê "true" (1). Praktycznie WAITPKT jest uûywane

do chwilowego wstrzymania pracy programu (wbrew pozorom -- nieraz okazuje sië to
przydatne). UWAGA: po uûyciu WAITPKT naleûy uûyê GETPKT, aby przywróciê program
do "stanu normalnego". Jeûeli dany pakiet bëdzie nadal potrzebny -- moûna go
odtworzyê przez REPLY. Jeôli to nie zostanie zrobione, wówczas przy wychodzeniu
z programu pakietowi przypisany bëdzie zwrotny kod bîëdu = 10.
Przykîad:
WAITPKT "Moj_port"
(jeôli w porcie "Moj_port" nie byîo nic -- program zostanie wstrzymany,
zaô rozkaz przyjmie wartoôê logicznâ 0).

1.122 hhu

HHU
HHU
Ustawia globalnâ flagë zatrzymania -- co powoduje, ûe wszystkie aktywne programy
ARexxa otrzymujâ zewnëtrzne zlecenie HALT. To z kolei uaktywnia przerwania typu
HALT. Po otrzymaniu zlecenia HALT przez programy -- flaga zostaje automatycznie
wyzerowana.

1.123 rx

RX
RX a$ [b$]
gdzie: a$, b$ -- to îaïcuchy.
Powoduje uruchomienie programu ARexxowskiego o nazwie a$. Jeôli chcesz skróciê
czas przeszukiwania katalogu, musisz w nazwie uûyê ôcieûki. Jeôli uûyjesz
parametru b$, wówczas îaïcuch b$ zostanie przekazany do programu. Przykîad:
RX "df0:REXX:bonieq"
(uruchamia program o nazwie "bonieq")

1.124 rxset

RXSET
RXSET a$ b$
gdzie: a$, b$ -- to îaïcuchy.
Dodaje argumenty (a$ -- nazwa, b$ -- wartoôê) do listy bufora (Clip List).
Jeôli na liôcie znajduje sië juô nazwa a$, wówczas jej wartoôê zostanie
zastâpiona przez îaïcuch b$. Jeôli jako b$ uûyjesz îaïcucha pustego --
wówczas nazwa a$ zostanie usuniëta z listy. Przykîad:
RXSET jump ’200’
(dopisuje parë "jump" i "200" do listy Clip List), zaô
RXSET jump ”
(usunie "jump" z tej listy.).

1.125 tc

TC

TC

Wyîâcza ôledzenie. UWAGA: przed wyîâczeniem ôledzenia musisz odîâczyê wszystkie rozkazy dotyczâce konsoli. Otwarte ostatnio okno ôledzenia znika z ekranu.

1.126 tg

TG

TG

Wîâcza ôledzenie globalne. Ôledzenie z poszczególnych aktywnych programów jest przeîâczane na nowo otwartâ konsolë. Okno konsoli moûe byê dowolnie powiëkszane i umieszczane na ekranie.

1.127 tt

TT

TT

Zeruje flagë ôledzenia globalnego -- co powoduje, ûe we wszystkich aktywnych programach ARexxa ôledzenie bëdzie wyîâczone.

1.128 tu

TU

TU

Ustawia flagë ôledzenia globalnego, co powoduje, ûe wszystkie aktywne programy ARexxa przechodzâ do trybu ôledzenia krok po kroku (wyôwietlanie wyników poôrednich i oczekiwanie po kaûdym wykonanym rozkazie). Zaleca sië stosowanie przy zapëtleniu programu lub przy innych tego typu "przyjemnoôciach".