AITECH®
Artificial Intelligence Laboratory

ZINTEGROWANY PAKIET SZTUCZNEJ INTELIGENCJI AITECHSPHINX® 4.5

KRZYSZTOF MICHALIK

PC-SHELL

SZKIELETOWY SYSTEM EKSPERTOWY

CZĘŚĆ 3
INSTRUKCJE JĘZYKA AITECHSPHINX

KATOWICE 2006

AITECH®
Artificial Intelligence Laboratory

ul. Kossutha 9, 40-844 KATOWICE
tel./fax: tel.: (0-32) 782-28-90
tel. kom. 0 502-99-27-28
e-mail: aitech@aitech.com.pl
WWW: http://www.aitech.com.pl

addFact

Składnia:

addFact(O, A, W);

gdzie: O – symbol lub zmienna typu char lub znak „_”
A – liczba lub zmienna typu char
W – liczba, łańcuch znakowy lub dowolna zmienna prosta lub znak „_”

Instrukcja *addFact* umożliwia utworzenie i dodanie faktu do bazy wiedzy – w sposób dynamiczny – podczas wykonywania programu z bloku sterowania. Fakt będzie zbudowany na podstawie wartości parametrów *O* (obiekt), *A* (atrybut), *W* (wartość). Jeśli obiekt lub wartość atrybutu nie występuje, to odpowiedni parametr jest zastąpiony znakiem „_”.

Przykład:

```
addFact( _, temperatura_ciała, 36.6 );
addFact( _, powierzchnia_kapelusza, "silnie pofałdowana" );
addFact( _, napięcie_w_obwodzie_U1, wysokie );
addFact( układ_1, wartość_progowa, 1.6 );
addFact( _, bóle_mięśni, _ );
```

Po wykonaniu powyższych instrukcji w bazie wiedzy pojawi się następujący zbiór nowych faktów:

```
temperatura_ciała = 36.6
powierzchnia_kapelusza = "silnie pofałdowana"
napięcie_w_obwodzie_U1 = "wysokie"
wartość_progowa( układ_1 ) = 1.6
bóle_mięśni
```

addFacts

Składnia:

addFacts(X1, X2, X3, X4);

gdzie: X1 – liczba lub zmienna typu *int*
X2 – identyfikator tablicy typu prostego lub znak „_”
X3 – identyfikator tablicy typu *char*
X4 – identyfikator tablicy typu prostego lub znak „_”

Instrukcja *addFacts* umożliwia dynamiczne – w trakcie pracy programu – utworzenie i dodanie zbioru faktów do bazy wiedzy.

Parametr *X1* określa liczbę faktów, które mają być utworzone i dodane do bazy wiedzy. Wartość *X1* nie może przekroczyć zadeklarowanego rozmiaru tablic *X2*, *X3* *X4*. Argument *X2* jest tablicą przechowującą identyfikatory obiektów (o ile występują). Jeśli obiekty nie są wykorzystywane, argument *X1* powinien przybrać wartość „_”. Argument *X3* jest tablicą przechowującą identyfikatory atrybutów. Należy podkreślić, że wszystkie te identyfikatory, o ile mają być wykorzystane w instrukcji *addFacts*, muszą być zadeklarowane w bloku faset. Argument *X4* jest tablicą przechowującą wartości atrybutów (o ile są używane).

Instrukcja *addFacts* syntetyzuje poszczególne fakty, „składając je” z informacji zawartych w tablicach *X2*, *X3*, *X4*, na odpowiadających sobie pozycjach.

Przykład:

```
int WART[ 3 ];
char ATR[ 3 ];
ATR[ 0 ] := atrybut1;
ATR[ 1 ] := atrybut2;
ATR[ 2 ] := atrybut3;
WART[ 0 ] := 1;
WART[ 1 ] := 2;
WART[ 2 ] := 3;
addFacts( 3, _ , ATR, WART );
```

W rezultacie wykonania powyższego fragmentu programu do bazy wiedzy dodane zostaną następujące fakty:

```
atrybut1 = 1
atrybut2 = 2
atrybut3 = 3
```

addSolution

Składnia:

addSolution(X);

gdzie: X – symbol *yes* lub *no* albo zmienna typu *char*

Instrukcja *addSolution* umożliwia automatyczne wprowadzenie do bazy wiedzy faktów będących konkluzjami wnioskowania uruchomionego przy pomocy instrukcji *goal*. Jeśli *X* przyjmie wartość *yes*, to konkluzje będą dodane do bazy wiedzy, jeśli wartością *X* będzie *no*, to fakty nie będą dodane. Jeśli instrukcja ta nie będzie użyta, to system nie dodaje konkluzji do bazy wiedzy, tj. zachowuje się tak, jakby domyślnie użyto instrukcji o postaci *addSolution*(*no*).

Typowym kontekstem użycia instrukcji *addSolution* może być aplikacja o architekturze tablicowej.

Przykład:

```
addSolution( yes );
goal( "konkluzja=X" );
// Bez wykorzystania instrukcji addSolution powyższy fragment
// należałoby zapisać w następujący sposób:
int LR;
char Tablica[ 10 ], Ob, Atr, War;
goal( "konkluzja=X" );
saveSolution( Tablica, LR );
for I = 0 to LR step 1
begin
   splitOAV( Tablica[ I ], Ob, Atr, War );
   addFact( _, Atr, War );
end;
```

appendMenu (2.2)

Składnia:

appendMenu(menu_id, tekst, funkcja);
appendMenu(menu_id, **separator**, _);
appendMenu(menu_id, **menu**, popup_menu_id);

gdzie: menu_id – zmienna typu *int*
tekst – zmienna typu *char* lub łańcuch tekstowy
funkcja – bezparametrowa funkcja, zdefiniowana w bloku sterowania
popup_menu_id – zmienna typu *int*

Instrukcja *appendMenu* służy do utworzenia nowej pozycji menu, która ma zostać przypisana do menu identyfikowanego przez parametr *menu_id*. Każda pozycja menu posiada związaną z nią funkcję, w której zdefiniowana jest akcja wykonywana po wybraniu danej opcji przez użytkownika.

Druga postać instrukcji *appendMenu* pozwala na dodanie do menu separatora. W tym przypadku drugi parametr stanowi symbol *separator*. Wartość trzeciego parametru nie jest istotna (podajemy „_”):

```
appendMenu( MenuId, separator, _ )
```

Ostatnia postać instrukcji *appendMenu* służy do dołączenia określonego wcześniej podmenu do już istniejącego menu (lub podmenu). Możliwe jest również dodanie standardowego podmenu „Okno”, zawierającego opcje umożliwiające zarządzanie oknami aplikacji. Menu to dodajemy poprzez podanie instrukcji w następującej postaci:

```
appendMenu( MenuId, menu, window );
```

Przykład wykorzystania instrukcji *appendMenu* zamieszczono w części poświęconej instrukcji *createMenu*.

Zobacz: *createMenu*, *createPopupMenu*, *fullMenu*

arraySize (2.2)

Składnia:

arraySize(array, type, size);

gdzie: array – identyfikator dowolnej tablicy
type – wartość 0, 1 lub 2
size – zmienna numeryczna typu *int*

Instrukcja *arraySize* jest instrukcją pomocniczą, służącą do pobierania rozmiaru tablicy. Jest ona wymagana do określania rozmiaru tablicy przekazanej do wnętrza funkcji, ponieważ w deklaracjach funkcji nie podaje się „z góry" rozmiaru tablic. Drugi parametr wywołania określa wymiar: *0* – obliczenie rozmiaru całej tablicy, niezależnie od jej rodzaju (macierz lub wektor), *1* – wyznaczenie ilości wierszy tablicy, *2* – wyznaczenie ilości kolumn tablicy.

Przykład:

```
function test( char T[] )
begin
   int S0, S1, S2;
   arraySize( T, 0, S0 );
   arraySize( T, 1, S1 );
   arraySize( T, 2, S2 );
   char Tekst;
   sprintf( Tekst, "Wymiary tablicy: %d : %d : %d", S0, S1, S2);
   messageBox( 0,0, "arraySize", Tekst );
end;
```

ascii

Składnia:

ascii(X, Y);

gdzie: X – znak lub zmienna typu *char*
Y – zmienna typu *int*

Instrukcja *ascii* zmienia znak reprezentowany przez *X* na wartość dziesiętną *Y* w kodzie ASCII.

Przykład:

```
int KD;
ascii( "a", KD );  // KD = 97
```

break

Składnia:

break;

Instrukcja *break* może być użyta w bloku instrukcji *menu*, *while* lub *for*. Wykonanie tej instrukcji powoduje wyjście z bieżącej instrukcji menu lub pętli i przejście do pierwszej instrukcji znajdującej się za nią.

c_ntos (2.1)

Składnia:

c_ntos(N, S, C);

gdzie: N – liczba lub zmienna numeryczna
S – zmienna typu *char*
C – znak typu *char*

Instrukcja *c_ntos* umożliwia przekształcenie wartości liczbowej reprezentowanej przez parametr *N* do postaci łańcucha znakowego i przypisanie go do zmiennej reprezentowanej przez *S*, przy czym różni się ona tym od standardowej instrukcji *ntos*, że jako separator w ułamkach dziesiętnych używany jest znak określony przez parametr *C*, (zamiast standardowo stosowanej kropki dziesiętnej). Instrukcja zapewnia kompatybilność pomiędzy różnymi systememi zapisu liczb zmiennoprzecinkowych. Ma to szczególne znaczenie np. w trakcie przesyłania liczby do arkusza kalkulacyjnego (np. MS-Excel). Przesłanie liczby jako łańcucha „1.34” spowoduje wstawienie go jako tekstu, natomiast przesłanie łańcucha „1,34” wstawi do komórki oczekiwaną przez nas liczbę (w polskiej wersji programu Excel).

Przykład:

```
char S;
float L;
L := -150;
c_ntos( 123.45, S, "," );  // S przyjmuje wartość "123,45"
c_ntos( L, S,",");         // S przyjmuje wartość "-150"
ntos( 123.45, S );         // S przyjmuje wartość "123.45"
```

Zobacz: *ntos*, *ston*

catchFact (2.15)

Składnia:

catchFact(dest, O, A, V, N);

gdzie: dest – zmienna typu *char*
O – symbol, łańcuch tekstowy, zmienna lub znak „_”
A – symbol, łańcuch tekstowy lub zmienna typu *char*
V – symbol, łańcuch tekstowy, liczba, zmienna lub znak „_”
N – liczba całkowita typu *int*

Instrukcja służy do pobierania faktów z bazy wiedzy. Pobrany fakt *O* zostaje zapisany jako łańcuch znakowy w zmiennej *dest*. Łańcuch ten może być następnie użyty jako parametr instrukcji *splitOAV*.

Fakty z bazy wiedzy są pobierane w zależności od podanych kryteriów w parametrach *O*, *A* oraz *W*. Wymagane jest podanie jedynie nazwy atrybutu (parametr *A*), natomiast obiekt (*O*) oraz wartość (*W*) mogą być pominięte poprzez podanie znaku „_”. Paramter *N* pozwala określić, który z kolei fakt, pasujący do podanego wzorca, ma zostać pobrany (numer określa pozycję względną, poczynając od 0).

Przykład 1:

```
// wyszukaj pierwszy fakt o atrybucie "grzyb" i wartości "kania"
catchFact( Dest, _, "grzyb", "kania", 0 );
// wartość Dest="" oznacza, ze fakt nie istnieje w bazie wiedzy
```

Przykład 2:

```
// Pobierz wszystkie fakty o atrybucie "grzyb"
I := 0;
while ( 1 == 1 )
begin
   catchFact( Dest, _, "grzyb", _, I );
   if ( Dest == "" )
   begin
       break;  // zakończ pętlę
   end
   else
   begin
       messageBox( 0, 0, "Fakt", Dest );
       I := I + 1;
   end;
end;
```

changeCategory (2.1)

Składnia:

changeCategory(cat);

gdzie: cat – łańcuch znakowy lub zmienna typu *char*

Instrukcja służy do uaktywnienia kategorii o nazwie określonej parametrem *cat*, co spowoduje przypisanie odpowiednich wartości zmiennym parametrycznym zdefiniowanym w tej kategorii. Podanie jako nazwa kategorii łańcucha pustego lub łańcucha „_” oznacza przywrócenie wszystkim zmiennym parametrycznym wartości domyślnych, określonych w bloku faset. Bliższe szczegóły dotyczące parametryzacji znajdują się w rozdziale „Parametryzacja baz wiedzy” (część 2. dokumentacji systemu PC-Shell pt. „Podręcznik inżyniera wiedzy”).

Zobacz: *setDefParamValue*

close

Składnia:

close(X);

gdzie: X – symbol, łańcuch znakowy lub zmienna typu *char*

Instrukcja *close* powoduje zamknięcie pliku identyfikowanego za pomocą nazwy *X*. W obecnej wersji systemu możliwe jest otwarcie i praca tylko z jednym plikiem jednocześnie dlatego parametr *X* jest w rzeczywistości ignorowany.

Przykład:

```
close( dane );
```

Zobacz: *open, fpnt, fseek, fgetc, fgets, fgetn, read*

closeAppWindow

Składnia:

closeAppWindow;

Instrukcja zamyka otwarte wcześniej okno aplikacji utworzonej za pomocą systemu PC-Shell.

closeChart (2.3)

Składnia:

closeChart(wykres);

gdzie: wykres – łańcuch lub zmienna typu *char*

Instrukcja zamyka i usuwa z pamięci wykres o nazwie *wykres* oraz zamyka okno widoku powiązane z danym wykresem.

Zobacz: *openChart*, *showChart*

closeSheet (2.2)

Składnia:

closeSheet(arkusz);

gdzie: arkusz – łańcuch lub zmienna typu *char*

Instrukcja zamyka arkusz o nazwie określonej parametrem *arkusz*. Przed zamknięciem arkusza zamykane są wszystkie jego „widoki”.

Zobacz: *getSheetRange, getSheetValue, openSheet, readSheet, setSheetRange, setSheetValue, showSheet, writeSheet*

closeWindow (2.1)

Składnia:

closeWindow(nazwa_okna);

gdzie: nazwa_okna – tytuł okna

Instrukcja zamyka okno, którego nazwa zaczyna się od tekstu *nazwa_okna*. Należy ostrożnie posługiwać się tą instrukcją, ponieważ może ona zamknąć okna których się nie spodziewamy. Instrukcja *closeWindow* zamyka w pierwszej kolejności okna („widoki”) otwarte przez aplikacje systemu PC-Shell, w dalszej kolejności próbuje zamknąć okna macierzyste innych aplikacji.

Przykład:

```
// sprawdzamy, czy jest uruchomiona aplikacja
// Menedżera Plików; jeżeli tak to ją zamykamy
int Ret;
isAppRunning( "Menedżer plików", Ret );
if ( Ret == 1 )
begin
   closeWindow( "Menedżer plików" );
end;
```

Zobacz: *showWindow*, *system*

confirmBox

Składnia:

confirmBox(X, Y, S1, S2, R);

gdzie: X, Y – liczba lub zmienna typu *int*
S1, S2 – symbol, łańcuch znakowy lub zmienna typu *char*
R – zmienna typu *int*

Instrukcja *confirmBox* umożliwia potwierdzanie lub zaniechanie pewnych działań poprzez pobranie odpowiedzi na pytanie skierowane do użytkownika. Użytkownik odpowiada za pomocą przycisków *OK* oraz *Anuluj*, które zwracają odpowiednio wartość *1* lub *0*, przypisaną do zmiennej oznaczonej tu przez *R*.

X i *Y* oznaczają współrzędne lewego górnego rogu okna. Symbole *S1* i *S2* przekazują teksty zawierające informacje dla użytkownika. Sugerowanym sposobem ich użycia jest przekazywanie za pomocą *S1* klasy wiadomości lub ogólnie sformułowanego pytania. Za pomocą *S2* można wtedy sprecyzować treść wiadomości.

Przykład:

```
int Odp;
Odp := 1;
while ( Odp == 1 )
begin
   // instrukcje...
   confirmBox( 0, 0, "Czy chcesz kontynuować?",
     "Naciśnij 'OK' jeśli tak, 'Anuluj' jeśli nie", Odp );
end;
```

continue

Składnia:

continue;

Instrukcja *continue* może być użyta wewnątrz instrukcji *menu* lub pętli związanych z instrukcjami *for* i *while*. Powoduje bezwarunkowe przejście na początek pętli, mimo nie zakończonej sekwencji instrukcji zawartych wewnątrz pętli lub – w przypadku instrukcji *menu* – powrót do obsługi menu.

Przykład:

```
// Jeśli zmienna I osiągnie wartość większą od 5,
// to instrukcja3 nie będzie wykonana.
while ( I < 10)
begin
   // instrukcja1;
   // instrukcja2;
   if ( I > 5)
   begin
       continue;
   end;
   // instrukcja3;
end;
```

Zobacz: *break*

createAppWindow

Składnia:

createAppWindow;

Instrukcja *createAppWindow* otwiera okno aplikacji systemu PC-Shell. Okno przykrywa okno główne systemu PC-Shell, wraz z jego menu i paskami narzędziowymi, tworząc własne tło do pracy aplikacji. Domyślnie na pasku tytułu okna umieszczony zostanie tekst: „Aplikacja systemu PC-Shell” (zmiana etykiety okna możliwa jest przy użyciu polecenia *setAppWinTitle*).

Zobacz: *setAppWinTitle*

createMenu (2.2)

Składnia:

createMenu(menu);

gdzie: menu – zmienna typu *int*

Instrukcja *createMenu* tworzy identyfikator nowe menu do użycia w instrukcji *fullMenu*. Po wykonaniu instrukcji *createMenu* w zmiennej *menu* zapamiętywany jest idetyfikator menu głównego. Teraz przy użyciu instrukcji *appendMenu* możemy dodawać pozycje menu lub utworzone podmenu.

Przykład:

```
// Struktura tworzonego menu:
//   Plik       Edycja
//     Otwórz     Wytnij
//     Zapisz     Kopiuj
//     ------     Wklej
//     Koniec

int MenuGłówne, Plik, Edycja;
createMenu( MenuGłówne );
createPopupMenu( Plik, "&Plik" );
appendMenu( Plik, "&Otwórz", otwórzFn );
appendMenu( Plik, "&Zapisz", zapiszFn );
appendMenu( Plik, separator, _ );
appendMenu( Plik, "&Koniec", koniecFn );
appendMenu( MenuGłówne, menu, Plik );
createPopupMenu( Edycja, "&Edycja" );
appendMenu( Edycja, "Wy&tnij", wytnijFn );
appendMenu( Edycja, "&Kopiuj", kopiujFn );
appendMenu( Edycja, "&Wklej",  wklejFn );
appendMenu( MenuGłówne, menu, Edycja );
fullMenu( MenuGłówne );
```

Zobacz: *appendMenu*, *createPopupMenu*, *fullMenu*

createPopupMenu (2.2)

Składnia:

createPopupMenu(podmenu, tytuł);

gdzie: podmenu – zmienna typu *int*
tytuł – zmienna typu *char* lub łańcuch tekstowy

Instrukcja tworzy nowe podmenu o nazwie (tytule) określonej parametrem *tytuł*. Do tak zainicjowanego podmenu możemy dołączać kolejne pozycje podmenu (przy użyciu instrukcji *appendMenu*). Mogą to być oczywiście kolejne, zagnieżdżone, podmenu. Po skonstruowaniu całego podmenu należy je dołączyć w odpowiednim miejscu do menu głównego przy użyciu instrukcji *appendMenu*.

Przykład wykorzystania instrukcji *createPopupMenu* zamieszczono w części poświęconej instrukcji *createMenu*.

Zobacz: *appendMenu*, *createMenu*, *fullMenu*

ddeConnect (2.1)

Składnia:

ddeConnect(kanał, usługa, temat);

gdzie: kanał – identyfikator kanału DDE (zmienna typu *int*)
usługa – określenie usługi DDE (zmienna typu *char*)
temat – określenie tematu DDE (zmienna typu *char*)

Instrukcja nawiązuje „konwersację” poprzez mechanizm dynamicznej wymiany danych (DDE) z serwerem, udostępniającym usługę o nazwie *usługa*, w zakresie „tematu” *temat*. W przypadku pozytywnego nawiązania konwersacji, w zmiennej *Kanał* zapamiętany zostaje identyfikator nawiązanej konwersacji (numer kanału), którym należy się posługiwać wywołując inne instrukcje dotyczące wymiany danych poprzez DDE. Nawiązując konwersację należy upewnić się, czy interesujący nas serwer jest uruchomiony. W przeciwnym razie, operacja nawiązania konwersacji zakończy się niepowodzeniem.

Każdą otwartą konwersację należy zakończyć przy użyciu instrukcji *ddeDisconect*.

Przykład:

```
int ID;
// nawiązanie konwersacji z arkuszem kalkulacyjnym MS-Excel
ddeConnect( ID, "Excel", "System" );
```

Zobacz: *ddeDisconect*, *ddeExecute*, *ddePoke*, *ddeRequest*

ddeDisconnect (2.1)

Składnia:

ddeDisconnect(kanał);

gdzie: kanał – identyfikator uprzednio otwartego kanału

Instrukcja zamyka nawiązaną konwersację i zwalnia wszelkie zasoby przez nią zarezerwowane. Po wywołaniu tej instrukcji nie wolno już wywoływać jakiejkolwiek instrukcji typu DDE w powiązaniu z kanałem *kanał*.

Zobacz: *ddeConnect*

ddeExecute (2.1)

Składnia:

ddeExecute(kanał, instrukcja);

gdzie: kanał – identyfikator otwartego kanału
instrukcja – nazwa instrukcji (polecenia) do wykonania

Instrukcja powoduje wykonanie polecenia *instrukcja* przez serwer DDE. Przyjęta konwencja zakłada, że podawane polecenia są zamknięte w nawiasach kwadratowych wraz z argumentami podanymi w nawiasach okrągłych. Najczęściej jeden łańcuch *instrukcja* może zawierać wiele poleceń, np. „[*New*(*1*)][*Close*(*0*)]".

Wykonanie polecenia może być czasochłonne i może kończyć się pojawieniem komunikatu o zajętości serwera. Użytkownik może wtedy powtórzyć operację lub jej zaniechać.

Przykład:

```
int ID;
// nawiązanie konwersacji z arkuszem MS-Excel o nazwie DANE.XLS
ddeConnect( ID, "Excel", "System" );
ddeExecute( ID, "[OPEN(\"DANE.XLS\")]" );
// wybranie obszaru W2K2-W2K2
ddeExecute( ID, "[SELECT(\"W2K2\",\"W2K2\")]" );
// zamknięcie bieżącego arkusza
ddeExecute( ID, "[CLOSE(0)]" );
// otwarcie nowego arkusza
ddeExecute( ID, "[NEW(1)]" );
// ...
ddeDisconnect( ID );
```

Zobacz: *ddeConnect*, *ddeDisconnect*, *ddePoke*, *ddeRequest*

ddePoke (2.1)

Składnia:

ddePoke(kanał, pozycja, dane);

gdzie: kanał – identyfikator otwartego kanału
pozycja – nazwa pozycji gdzie przesyłamy dane
dane – dane do przesłania do serwera

Instrukcja powoduje przesłanie danych zawartych w zmiennej *dane* do serwera DDE na pozycji określonej przez zmienną *pozycja*. Dane mogą być dowolnego typu prostego. Zostaną one automatycznie zamienione na łańcuch tekstowy i w tej postaci przesłane do serwera.

Przykład:

```
int ID;
// nawiązanie konwersacji z serwerem MS-Excel
// w celu otwarcia arkusza DANE.XLS
ddeConnect( ID, "Excel", "System" );
ddeExecute( ID, "[OPEN(\"DANE.XLS\")]" );
ddeDisconnect( ID );
// konwersacja mająca na celu wpisanie danych do arkusza
ddeConnect( ID, "Excel", "DANE.XLS" );
// wpisanie do komórki W1K1 odpowiedniego tekstu
ddePoke( ID, "W1K1", "Tekst z PC-Shell'a" );
// wpisanie liczby do komórki W1K2
ddePoke( ID, "W1K2", -1 );
// ...
ddeDisconnect( ID );
```

Zobacz: *ddeConnect*, *ddeDisconnect*, *ddeExecute*, *ddeRequest*

ddeRequest (2.1)

Składnia:

ddeRequest(kanał, pozycja, dane);

gdzie: kanał – identyfikator otwartego kanału
pozycja – nazwa pozycji, spod której pobieramy dane
dane – bufor, gdzie mamy umieścić dane

Instrukcja umożliwia pobranie danych z serwera z pozycji określonej parametrem *pozycja*. Pobrane dane, jeżeli zostaną pobrane, przesyłane są do zmiennej *dane*.

Przykład:

```
int ID, SYSTEM;
int DANE;
// nawiązanie konwersacji z serwerem MS-Excel
// w celu otwarcia arkusza DANE.XLS
ddeConnect( SYSTEM, "Excel", "System" );
ddeExecute( SYSTEM, "[OPEN(\"DANE.XLS\")]");
// konwersacja mająca na celu wpisanie danych do arkusza
ddeConnect( ID, "Excel", "DANE.XLS" );
// pobranie z komórki W1K1 danych do zmiennej DANE
ddeRequest( ID, "W1K1", DANE );
ddeDisconnect( ID );
// zamknięcie aktywnego arkusza
ddeExecute( SYSTEM, "[CLOSE(0)]" );
ddeDisconnect( SYSTEM );
```

Zobacz: *ddeConnect*, *ddeDisconect*, *ddeExecute*, *ddePoke*

delFacts (2.1)

Składnia:

delFact(O, A, V);

gdzie: O – symbol, łańcuch tekstowy, zmienna, lub znak „_”
A – symbol, łańcuch tekstowy lub zmienna
V – symbol, łańcuch tekstowy, liczba, zmienna lub znak „_”

Instrukcja *delFacts* służy do usuwania wybranych faktów z bazy wiedzy. W efekcie jej wykonania z bazy wiedzy usunięte zostaną wszystkie fakty, pasujące do podanej trójki *OAV*, przy czym symbol „_” oznacza „wszystkie wartości”. Jedynym obowiązkowym elementem jest nazwa atrybutu *A*.

Przykład:

```
// Początkowa zawartość bazy wiedzy:
//   atrybut1 = "a"
//   atrybut2 = 1
//   atrybut2 = 2
//   profil( huty ) = "ok"
//   profil( kopalnie ) = "ok"

delFacts( _, "atrybut1", "b" );
delFacts( _, "atrybut2", _ );
delFacts( huty, "profil", _ );

// Po wykonaniu powyższych instrukcji w bazie pozostaną jedynie
// następujące fakty:
//   atrybut1 = "a";
//   profil(kopalnie) = "ok";
```

Zobacz: *delNewFacts*

delNetwork

Składnia:

delNetwork(X);

gdzie: X – symbol lub zmienna typu *char*

Instrukcja *delNetwork* powoduje usunięcie symulatora sieci neuronowej, wskazanego za pomocą parametru *X* (nazwa źródła zadeklarowanego w bloku *sources*).

Przykład:

```
sources
   sieć:
       type neural_net
       file "siec1.def";
end;
control
   // ...
   delNetwork( sieć );
   // ...
end;
```

Zobacz: *initNetwork*, *runNetwork*

delNewFacts

Składnia:

delNewFacts;

Instrukcja powoduje usunięcie z bazy wiedzy wszystkich nowych faktów, o ile takie się w niej znajdują. Do kategorii nowych faktów zalicza się te fakty, które nie są częścią opisu w bloku *facts* bazy wiedzy.

dlgBindButton (4.0+)

W wersji 4.0 dodana instrukcja rozszerzona o obsługę dialogów definiowanych w programie Dialog Editor.

Składnia:

dlgBindButton(dlg_id, item_id, fn_name);

gdzie: dlg_id – zmienna typu *int*
item_id – identyfikator numeryczny typu *int* lub znakowy typu *char*
fn_name – identyfikator znakowy (nazwa funkcji)

Zaawansowana instrukcja, przypisująca przyciskowi (ang. *button*), określonemu parametrem *item_id*, zawartemu we wskazanym oknie dialogowym (*dlg_id*), funkcję o nazwie podanej w argumencie *fn_name*. W czasie, kiedy okno dialogowe jest aktywne, każdorazowe naciśnięcie danego przycisku spowoduje wywołanie wskazanej funkcji. Dokładniejszy opis budowy funkcji dialogowych znajduje się w rozdziale 5. „Podręcznika inżyniera wiedzy” (część 2. dokumentacji), w części poświęconej definiowaniu funkcji.

Przykład 1:

```
int DLG;
char Pensja;
dlgCreate( DLG, "custom.dll", "EMP_DLG" );
dlgBindStatic( DLG, 101, "Kowalski Jan" );
dlgBindEdit( DLG, 102, Pensja );
dlgBindButton( DLG, 201, przWyczyść );
dlgExecute( DLG );
```

Przykład 2: użycie dialogu zdefiniowanego w programie Dialog Editor

```
int DLG, Ret;
dlgCreate( DLG, "", "kwestionariusz.dlg" );
dlgBindEdit( DLG, "Imie", Imię );
dlgBindEdit( DLG, "Nazwisko", Nazwisko );
dlgBindEdit( DLG, "Ulica", Ulica );
dlgBindEdit( DLG, "Nr", NrDomu );
dlgBindEdit( DLG, "KodPocztowy", KodPocztowy );
dlgBindRadioButton ( DLG, "Plec", Plec );
dlgBindEdit( DLG, "Woj", Województwo );

dlgBindButton( DLG, "GetWoj", getWojewództwo );
dlgExecute( DLG, Ret );
```

Zobacz: *dlgCreate*, *dlgBindEdit*, *dlgBindComboBox*, *dlgBindCheckBox*, *dlgBindListBox*, *dlgBindRadioButton*, *dlgBindStatic*, *dlgExecute*

dlgBindCheckBox (4.0+)

W wersji 4.0 dodana instrukcja rozszerzona o obsługę dialogów definiowanych w programie Dialog Editor.

Składnia:

dlgBindCheckBox(dlg_id, item_id, var);

gdzie: dlg_id – zmienna typu *int*
item_id – identyfikator numeryczny typu *int* lub znakowy typu *char*
variable – zmienna numeryczna typu *int*

Instrukcja powoduje przypisanie przyciskowi typu *check-box* o identyfikatorze określonym przez parametr *item_id*, zawartemu we wskazanym oknie dialogowym (*dlg_id*), zmiennej *Variable*. Wyróżnia się trzy możliwe stany przycisku: „włączony” (*1*), „wyłączony” (*0*) oraz „nieaktywny” (*2*). Po uruchomieniu okna dialogowego (instrukcją *dlgExecute*) następuje ustawienie przycisku w stan uzależniony od wartości zmiennej *var*. Po zamknięciu okna przez naciśnięcie przycisku „OK” zmienna *var* przyjmie wartość opisującą stan przycisku w momencie zamknięcia okna.

Zobacz: *dlgCreate*, *dlgExecute*, *dlgBindEdit*, *dlgBindComboBox*, *dlgBindButton*, *dlgBindListBox*, *dlgBindRadioButton*, *dlgBindStatic*

dlgBindComboBox (4.0+)

W wersji 4.0 dodana instrukcja rozszerzona o obsługę dialogów definiowanych w programie Dialog Editor.

Składnia:

dlgBindComboBox(dlg_id, item_id, array, sel_str);

gdzie: dlg_id – zmienna typu *int*
item_id – identyfikator numeryczny typu *int* lub znakowy typu *char*
array – tablica znakowa (typu *char*)
sel_str – zmienna znakowa typu *char*

Instrukcja wiąże element typu „lista kombinowana" (ang. *combo-box*), określony parametrem *item_id*, zawarty we wskazanym oknie dialogowym (*dlg_id*), z tablicą *array*, przechowującej kolejne elementy listy. Zmiennej *sel_str* przypisany zostanie tekst wybrany z listy lub wpisany przez użytkownika.

Zobacz: *dlgCreate*, *dlgExecute*, *dlgBindEdit*, *dlgBindComboBox*, *dlgBindButton*, *dlgBindCheckBox*, *dlgBindListBox*, *dlgBindRadioButton*, *dlgBindStatic*

dlgBindEdit (4.0+)

W wersji 4.0 dodana instrukcja rozszerzona o obsługę dialogów definiowanych w programie Dialog Editor.

Składnia:

dlgBindEdit(dlg_id, item_id, var);

gdzie: dlg_id – zmienna typu *int*
item_id – identyfikator numeryczny typu *int* lub znakowy typu *char*
var – zmienna znakowa typu *char*

Instrukcja powoduje przypisanie elementowi typu „pole edycyjne” (ang. *edit-box*) o idetyfikatorze określonym parametrem *item_id*, zawartemu we wskazanym oknie dialogowym (*dlg_id*), zmiennej *var*. W momencie uruchomienia dialogu instrukcją *dlgExecute* nastąpi przepisanie do tego pola zawartości zmiennej *var*, natomiast po zamknięciu okna przez naciśnięcie przycisku „OK” – przepisanie do zmiennej wartości wpisanej do pola edycyjnego przez użytkownika.

Zobacz: *dlgCreate*, *dlgBindButton*, *dlgBindCheckBox*, *dlgBindComboBox*, *dlgBindListBox*, *dlgBindRadioButton*, *dlgBindStatic*, *dlgExecute*

dlgBindListBox (4.0+)

W wersji 4.0 dodana instrukcja rozszerzona o obsługę dialogów definiowanych w programie Dialog Editor.

Składnia:

dlgBindListBox(dlg_id, item_id, array, index);

gdzie: dlg_id – zmienna typu *int*
item_id – identyfikator numeryczny typu *int* lub znakowy typu *char*
array – tablica znakowa (typu *char*)
index – zmienna numeryczna typu *int*

Instrukcja wiąże element dialogu typu „lista" (ang. *list-box*) ze zmienną tablicą *array,* w której umieszczone są kolejne elementy listy. Zmienna *index* przechowuje indeks (począwszy od 0) wybranego elementu tablicy. W przypadku, gdy nie został wybrany żaden element listy, zmiennej zostanie przypisana wartość *-1*.

Zobacz: *dlgCreate*, *dlgExecute*, *dlgBindCheckBox*, *dlgBindStatic*, *dlgBindEdit*, *dlgBindComboBox*, *dlgBindButton*, *dlgBindRadioButton*

dlgBindRadioButton (4.0+)

W wersji 4.0 dodana instrukcja rozszerzona o obsługę dialogów definiowanych w programie Dialog Editor.

Składnia:

dlgBindRadioButton(dlg_id, item_id, var);

gdzie: dlg_id – zmienna typu *int*
item_id – identyfikator numeryczny typu *int* lub znakowy typu *char*
var – zmienna numeryczna typu *int*

Instrukcja powoduje przypisanie przyciskowi typu *radio-button* o identyfikatorze określonym parametrem *item_id*, zawartemu we wskazanym oknie dialogowym (*dlg_id*), zmiennej *var*. Możliwe są trzy stany tego typu przycisku: „włączony” (*1*), „wyłączony” (*0*) oraz „nieaktywny” (*2*). Podczas uruchomienia okna dialogowego (instrukcją *dlgExecute*) następuje ustawienie przycisku w stan uzależniony od wartości zmiennej *var*. Po zamknięciu okna dialogowego przez naciśnięcie przycisku „OK” zmienna *var* przyjmie wartość określającą stan przycisku w momencie zamknięcia okna.

Zobacz: *dlgCreate*, *dlgExecute*, *dlgBindStatic*, *dlgBindButton*, *dlgBindCheckBox*, *dlgBindEdit*, *dlgBindComboBox*, *dlgBindListBox*

dlgBindStatic (4.0+)

W wersji 4.0 dodana instrukcja rozszerzona o obsługę dialogów definiowanych w programie Dialog Editor.

Składnia:

dlgBindStatic(dlg_id, item_id, text);

gdzie: dlg_id – zmienna typu *int*
item_id – identyfikator numeryczny typu *int* lub znakowy typu *char*
text – łańcuch znakowy

Instrukcja powoduje przypisanie elementowi statycznemu (ang. *static*) o identyfikatorze określonym przez parametr *item_id*, zawartemu we wskazanym oknie dialogowym (*dlg_id*), tekstu zawartego w argumencie *text*. W momencie uruchomienia okna instrukcją *dlgExecute* nastąpi przepisanie do pola zawartości parametru *text*.

Zobacz: *dlgCreate*, *dlgExecute*, *dlgBindEdit*, *dlgBindComboBox*, *dlgBindListBox*, *dlgBindButton*, *dlgBindRadioButton*, *dlgBindCheckBox*

dlgCreate (4.0+)

W wersji 4.0 dodana instrukcja rozszerzona o obsługę dialogów definiowanych w programie Dialog Editor.

Składnia:

dlgCreate(dlg_id, name, res_name);

gdzie: dlg_id – zmienna typu *int*
name – łańcuch znakowy
res_name – łańcuch znakowy lub identyfikator całkowity typu *int*

Instrukcja inicjuje tworzenie okna dialogowego o identyfikatorze *res_name* i zdefiniowanego (istniejącego) w bibliotece *name* lub zdefiniowanego w pliku programie Dialog Editor i zapisanego w pliku o nazwie określonej w parametrze *name*. W rezultacie wykonania instrukcji następuje zainicjowanie okna dialogowego i przypisanie jego identyfikatora zmiennej *dlg_id*. Sprawdzenie czy dane okno rzeczywiście istnieje we wskazanej bibliotece następuje dopiero w chwili wywołania instrukcji *dlgExecute*. Instrukcja *dlgCreate* służy jedynie zainicjowaniu odpowiednich buforów. Po zainicjowaniu możliwe jest powiązanie zmiennych bloku sterowania z elementami kontrolnymi zawartymi w danym oknie dialogowym (służą temu instrukcje typu *dlgBind…* – patrz niżej).

Przykład 1:

```
int DLG;
char Pensja;
dlgCreate( DLG, "custom.dll", "EMP_DLG" );
dlgBindStatic( DLG, 101, "Kowalski Jan" );
dlgBindEdit( DLG, 102, Pensja );
dlgExecute( DLG );
```

Przykład 2: wywołanie okna dialogowego utworzonego w programie Dialog Editor

```
int DLG,Ret;
dlgCreate( DLG, "", "wojewodztwo.dlg" );
dlgBindListBox( DLG, "Woj", _, Województwo );
dlgExecute( DLG, Ret );
```

Zobacz: *dlgExecute*, *dlgBindButton*, *dlgBindCheckBox*, *dlgBindRadioButton,*
dlgBindStatic, *dlgBindListBox*, *dlgBindEdit*, *dlgBindComboBox*

dlgExecute (4.0+)

W wersji 4.0 dodana instrukcja rozszerzona o obsługę dialogów definiowanych w programie DialogEditor.

Składnia:

dlgExecute(dlg_id, ret_val);

gdzie: dlg_id, ret_val – zmienne typu *int*

Instrukcja powoduje aktywowanie okna dialogowego zainicjowanego uprzednio instrukcją *dlgCreate*. Pierwszy parametr (*dlg_id*) identyfikuje okno, natomiast parametr *ret_val* jest zmienną, która przyjmuje wartość zależną od sposobu zamknięcia danego okna:

1 – przez naciśnięcie przycisku „OK”;

0 – przez naciśnięcie przycisku „Anuluj”.

Zobacz: *dlgCreate, dlgBindButton*, *dlgBindCheckBox*, *dlgBindRadioButton, dlgBindStatic*, *dlgBindEdit*, *dlgBindComboBox*, *dlgBindListBox*

exit

Składnia:

exit;

Instrukcja *exit* jest przeznaczona do bezwarunkowego opuszczenia instrukcji *menu* lub *fullMenu* i przejścia do wykonania instrukcji znajdującej się bezpośrednio za blokiem menu. Jeśli instrukcja *exit* będzie umieszczona wewnątrz instrukcji pętli (np. *for*, *while*), nawet zagnieżdżonych, to przerwie ich działanie i nastąpi wykonanie instrukcji znajdującej się bezpośrednio za instrukcją *menu*. Instrukcja może być umieszczona wewnątrz bloku menu lub wewnątrz definicji funkcji.

fgetc

Składnia:

fgetc(X);

gdzie: X – zmienna typu *char*

Instrukcja *fgetc* powoduje odczytanie jednego znaku z pliku otwartego za pomocą instrukcji *open* i wpisanie tego znaku do zmiennej *X*.

Przykład:

```
char Znak;
// ...
fgetc( Znak );
```

Zobacz: *fgetn*, *fgets*, *fput*, *open*, *close*

fgetn

Składnia:

fgetn(X);

gdzie: X – zmienna dowolnego typu numerycznego

Instrukcja *fgetn* powoduje odczytanie liczby z aktualnie otwartego pliku i przypisanie jej zmiennej *X*. Liczba powinna być zakończona spacją lub znakiem nowej linii.

Zobacz: *fgetc*, *fgets*, *fput*, *open*, *close*

fgets

Składnia:

fgets(X);

gdzie: X – zmienna typu *char*

Instrukcja *fgets* powoduje odczytanie symbolu lub łańcucha znakowego z aktualnie otwartego pliku i przypisanie tego napisu zmiennej *X*. Napis powinien być zakończony znakiem nowej linii.

Zobacz: *fgetc*, *fgetn*, *fput*, *open*, *close*

fileBox (2.2)

Składnia:

fileBox(X, Y, tytuł, typy_plików, nazwa_pliku);

gdzie: X, Y – liczba lub zmienna typu *int*
tytuł – łańcuch znakowy typu *char*
typy_plików – łańcuch znakowy typu *char*
nazwa_pliku – zmienna znakowa typu *char*

Instrukcja *fileBox* służy do wybrania lub pobrania nazwy pliku w standardowym oknie typu *Otwórz* (*FileOpen*). Parametry *X*, *Y* określają pozycję okna na ekranie (podanie wartości *0*, *0* spowoduje wyświetlenie okna w środku ekranu). Kolejny parametr (*tytuł*) określa tekst, jaki pojawi na pasku tytułu okna wyboru pliku. Parametr *typy_plików* określa typy plików wyświetlane na liście „Typ pliku”. Lista musi być zbudowana według schematu:

```
tekst1|schemat1
```

gdzie *tekst1* to tekst pojawiający się na liście, a *schemat1* to maska wyboru plików. Możliwe jest zdefiniowanie większej ilości pozycji, przy czym każda pozycja jest oddzielona od siebie znakiem „|”, np.:

```
"Pliki baz wiedzy (*.bw)|*.bw|Wszystkie pliki (*.*)|*.*"
```

Ostatni parametr (*nazwa_pliku*) stanowi zmienna, do której zostanie wpisana nazwa pliku wybranego przez użytkownika. Nazwa pliku jest zapamiętana w postaci pełnej ścieżki (oznaczenie napędu + ścieżka dostępu).

Przykład:

```
char S;
S := "";
fileBox( 0, 0, "Wybór pliku",
  "Bazy wiedzy (*.bw)|*.bw|Wszystkie pliki (*.*)|*.*", S );
```

for

Składnia:

for Z := W1 **to** W2 **step** W3 *instrukcja_złożona*

gdzie: Z – zmienna numeryczna
W1, W2, W3 – liczby lub zmienne typu numerycznego

Instrukcja *for*, podobnie jak instrukcja *while* umożliwia tworzenie pętli programowych. Jest najczęściej wykorzystywana w sytuacjach, gdy „z góry" znamy liczbę powtórzeń pętli. Stanowi wygodne narzędzie do cyklicznego indeksowania tablic w zadanym przedziale wartości.

Działanie instrukcji *for* rozpoczyna się od przypisania do zmiennej *Z* wartości reprezentowanej przez *W1*, a następnie sprawdzenia czy *Z<=W2*. Jeśli relacja jest spełniona, to wykonywana jest *instrukcja_złożona*. Po wykonaniu tej instrukcji następuje automatyczne dodanie liczby reprezentowanej przez *W3* do zmiennej *Z* i przejście do ponownego sprawdzenia wspomnianej relacji. Jeśli relacja nie jest spełniona, to znaczy zmienna *Z* przekroczy wartość *W2*, to następuje przejście do wykonania instrukcji znajdującej się bezpośrednio za instrukcją *for*. Działanie instrukcji *for* można zatem zdefiniować za pomocą instrukcji *while* w następujący sposób:

```
Z := W1;
while( Z <= W2 )
begin
   // instrukcje...
   Z := Z + W3;
end;
```

gdzie instrukcje są zbiorem instrukcji należących do bloku *instrukcja_złożona* związanego z instrukcją *for*.

Typ zmiennej *Z* jest typem decydującym o zakresie typów liczb lub zmiennych *W1*, *W2* oraz *W3*.

Podobnie jak w przypadku instrukcji *while*, dozwolone jest użycie instrukcji *for* wewnątrz innej instrukcji *for* lub *while*.

Przykład:

```
// Przypisanie wartości 0 (zero) kolejnym elementom tablicy TAB
int I, TAB[ 10 ];
for I := 0 to 9 step 1
begin  TAB[ I ] := 0; end;
// Przykład zagnieżdżonego użycia instrukcji for
// (program przypisuje kolejnym elementom tablicy T
// wartości będące iloczynem indeksu wierszy i kolumn
// kolejnych jej elementów)
int X, I, J, T[ 5, 10 ];
for I := 0 to 4 step 1
begin
   for J := 0 to 9 step 1
   begin
       T[ I, J ] := I * J;
   end;
end;
```

forward (4.1)

Składnia:

forward;

forward(TABLICA_FAKTÓW);

gdzie: *TABLICA_FAKTÓW* – tabela typu **char**.

Instrukcja *forward* powoduje uruchomienie procesu wnioskowania w przód. Jako rezultat (jeżeli nie zablokowano okna rozwiązań instrukcją *solutionWin*) wyświetlane jest okno rozwiązań zawierające listę wszystkich rozwiązań wygenerowanych przez algorytm wnioskowania w przód. Rozwiązania są dodawane lub nie do bazy wiedzy jako fakty w zależności od ustawienia instrukcją *addSolution*. Możliwe jest również przechwycenie wszystkich rozwiązań instrukcją *saveSolution* do tablicy (należy jedynie pamiętać o zdefiniowaniu odpowiednio dużej tablicy).

Druga forma wywołania wymaga podania jako parametru tablicy w której podane są fakty dodawane na czas wnioskowania do bazy wiedzy. Wywołanie w tej formie nie wymaga wcześniejszego dodawania faktów do bazy wiedzy i automatycznie zapewnia usunięcie tych faktów po zakończeniu wnioskowania.

Zobacz: *goal, solve, saveSolution*

fput

Składnia:

fput(X);

gdzie: X – liczba, symbol, łańcuch znakowy lub zmienna

Instrukcja *fput* powoduje wprowadzenie do bieżąco otwartego pliku wartości reprezentowanej przez *X*.

Przykład:

```
open( "dane", w );
// (symbol "\n" oznacza znak nowej linii).
fput( "\nłańcuch znakowy" );
fput( dowolny_symbol );
fput( 12345.67 );
fput( M[ 12 ] );
close( dane );
```

Zobacz: *fgetc*, *fgetn*, *fgets*, *open*, *close*

freadFacts

Składnia:

freadFacts(X);

gdzie: X – symbol, łańcuch znakowy lub zmienna typu *char*

Instrukcja *freadFacts* powoduje odczytanie pliku określonego parametrem *X*, a następnie umieszczenie zawartych w nim faktów na początku bazy wiedzy. Fakty zapisane są w pliku *X* w sposób jawny (kolejne wartości faktów w postaci trójek *OAV*). Blok faktów zakończony jest słowem kluczowym *end*.

Przykładowa postać pliku *fakty.fct*:

```
wskaźnik_płynności = 2;
zabezpieczenie = "hipoteka";
zabezpieczenie = "samochód o wartości > 20.000 zł";
dochód = 1000;
ilość_mieszkańców = 4;
end;
```

Przykład:

```
freadFacts( "c:\\AitechSPHINX\\bw\\fakty.fct" );
```

Zobacz: *addFact*, *addFacts*, *catchFact*

freeSource

Składnia:

freeSource(X);

gdzie: X – symbol lub zmienna typu *char*

Instrukcja *freeSource* powoduje zwolnienie aktywnego źródła wiedzy, w tym usunięcie z bazy wiedzy reguł i/lub faktów pochodzących z tego źródła. Po zwolnieniu danego źródła, może ono być ponownie załadowane do bazy wiedzy i uaktywnione za pomocą instrukcji *getSource*.

Przykład wykorzystania instrukcji *freeSource* zamieszczono w części poświęconej instrukcji *getSource*.

fromClipboard (2.15)

Składnia:

fromClipboard(dest);

gdzie: dest – dowolna zmienna prosta

Instrukcja służy pobraniu zawartości schowka (jeżeli jest nie pusta i jest w formacie tekstowym) i zapamiętaniu jej w zmiennej *dest*. Jeżeli zmienna *dest* jest typu *char* – następuje proste zapamiętanie zawartości schowka jako tekstu, jeżeli natomiast argument instrukcji jest zmienną numeryczną – nastąpi próba konwersji tekstu zawartego w schowku do postaci liczbowej. Jeżeli tekst zawarty w schowku nie jest liczbą wstawiona zostanie wartość *0*.

Instrukcją tą należy posługiwać się ostrożnie, ponieważ może ona na przykład wstawić do zmiennej tekstowej bardzo duży blok tekstu.

Przykład:

```
int ID;
char STR;
long L;
STR := "12000";
toClipboard( STR );
if ( RETURN == 1 )
begin
   fromClipboard( L );  // L powinno zawierać 12000
end;
```

Zobacz: *toClipboard*

fsaveExplan

Składnia:

fsaveExplan(X, Y, Z);

gdzie: X – symbol, łańcuch znakowy lub zmienna typu *char*
Y – symbol lub zmienna typu *char*
Z – liczba lub zmienna typu *int*

Instrukcja *fsaveExplan* powoduje wyprowadzenie do pliku określonego przez nazwę lokalną *X* tekstu wyjaśnień typu „jak?”, wygenerowanych podczas konsultacji. Instrukcja ta powinna być zatem poprzedzona w ciągu wykonywanych instrukcji użyciem instrukcji *goal* lub *solve*.

Argument *Y* jest identyfikatorem atrybutu, dla którego wyjaśnienia mają być wyprowadzone do pliku.

Argument *Z* ustala tryb wyjaśnień i w obecnej wersji systemu nie jest wykorzystany (należy przypisać mu wartość *1*).

Instrukcję można traktować jako narzędzie do budowy powiązań pomiędzy systemem PC-Shell a innymi aplikacjami, w szczególności napisanymi dla Windows.

Przykład:

```
fsaveExplan( "how.exp", diagnoza, 1 );
fsaveExplan( jak, decyzja_kredytowa, 1 );
```

fsaveSolution

Składnia:

fsaveSolution(X);

gdzie: X – symbol, łańcuch znakowy lub zmienna typu *char*

Instrukcja *fsaveSolution* powoduje zapisanie do pliku określonego przez nazwę lokalną *X* zbioru konkluzji, wygenerowanych w związku z ostatnio potwierdzoną hipotezą. Instrukcja ta, podobnie jak w przypadku instrukcji *fsaveExplan*, powinna być zatem poprzedzona w ciągu wykonywanych instrukcji użyciem instrukcji *goal* lub *solve*.

Instrukcję można traktować jako narzędzie do budowy powiązań pomiędzy systemem PC-Shell a innymi aplikacjami, w szczególności napisanymi dla Windows.

Przykład:

```
fsaveSolution( "rozw.sol" );
fsaveSolution( rozwiaz );
```

fseek (2.15)

Składnia:

fseek(offset, direction);

gdzie: offset – liczba lub zmienna całkowita
direction – liczba *0*, *1*, lub *2*

Instrukcja służy do ustawienia żądanej pozycji w otwartym pliku. Argument *offset* definiuje o ile bajtów przesunąć się od pozycji określonej w drugim argumencie (*direction*). Kolejno: wartość *0* oznacza pozycjonowanie bezwzględne od początku pliku, *1* – pozycjonowanie względne od bieżącej pozycji, *2* – pozycjonowanie bezwzględne od końca pliku. W ostanim przypadku *offset* powinien być liczbą ujemną.

Przykład:

```
open( "file.txt", "r" );
fgets( STR );
fseek( 10, 0 );  // przesuń o 10 bajtów od początku pliku
fgetc( C );
fseek( 2, 1 );   // pomiń 2 bajty
fgetc( C );
fseek( -2, 2 );  // ustaw pozycję na 2 bajty przed końcem pliku
```

Zobacz: *open*, *close*, *fput*, *fgetc*, *fgetn*, *fgets*, *readMat*

ftoi (2.1)

Składnia:

ftoi(src, dest);

gdzie: src – liczba rzeczywista typu *float* lub *double*
dest – zmienna typu całkowitego (*int* lub *long*)

Instrukcja służy do konwersji liczb zmiennoprzecinkowych na całkowite. Konwersja polega na przeniesieniu do zmiennej *dest* części całkowitej liczby *src*.

Konwersje w drugą stronę (liczby całkowitej na rzeczywistą) odbywają się automatycznie.

Przykład:

```
int I;
float F;
F := 10.45;
ftoi( F, I );  // w zmiennej I znajduje się liczba 10
```

fullMenu (2.2)

Składnia:

fullMenu(menu);

gdzie: menu – zmienna typu *int*

Instrukcja *fullMenu* powoduje uruchomienie okna aplikacji systemu (jeżeli nie zostało uruchomione instrukcją *createAppWindow*) i dodaje do niego menu zdefiniowane przy użyciu instrukcji *createMenu*, *createPopupMenu* i *appendMenu*. Następnie system przechodzi w stan oczekiwania na wybór przez użytkownika jednej z opcji systemu. Wybranie dowolnej opcji powoduje wywołanie powiązanej z nią funkcji (zobacz instrukcja *appendMenu*). Wyjście z menu odbywa się przez umieszczenie w funkcjach instrukcji *mainMenu*.

Przykład wykorzystania instrukcji *fullMenu* zamieszczono w części poświęconej instrukcji *createMenu*.

Zobacz: *appendMenu*, *createMenu*, *createPopupMenu*

function (2.2)

Składnia:

function nazwa_funkcji [(*lista_arguemntów*)]
begin
  instrukcje
end;

Deklaracja *function* służy do deklarowania funkcji w bloku sterowania. Może ona wystąpić jedynie na początku bloku *control*, przed innymi instrukcjami. Argument *nazwa_funkcji* musi być nazwą symboliczną (zaczynającą się od małej litery), po której może występić lista parametrów, a następnie – ujęta w bloku *begin…end* –treść funkcji, stanowiąca jej implementację. Bliższe szczegóły na temat definiowania funkcji znajdują się w rozdziale 5. „Podręcznika inżyniera wiedzy”.

Przykład:

```
function wyświetlKomunikat( char Tekst )
begin
   messageBox( 0, 0, "Komunikat", Tekst );
end;

function test( int X, int Y )
begin
   if ( X < Y )
   begin
       return 1;
   end
   return 0;
end;

function obliczX( int X, long &Wynik )
begin
   Wynik := X * 10;
end;
```

getActiveWindow (2.2)

Składnia:

getActiveWindow (nazwa_okna);

gdzie: nazwa_okna – łańcuch lub zmienna typu *char*

Instrukcja zapamiętuje w zmiennej *nazwa_okna* nazwę aktywnego okna.

Przykład:

```
char Okno, Typ;
getActiveWindow( Okno );
if ( Okno <> "" )
begin
   getWindowType( Okno, Typ );
   if ( Typ == "sheet" )
   begin
       writeSheet( Okno, "" );
   end;
end;
```

Zobacz: *getWindowType*

getDate (2.2)

Składnia:

getDate(day, month, year);

gdzie: day, month, year – zmienne typu *int*

Instrukcja *getDate* zapamiętuje w zmiennych systemową datę.

Przykład:

```
int Dz, Mies, Rok;
char T;
getDate( Dz, Mies, Rok );
sprintf( T, "Dzisiejsza data: %02d-%02d-%04d", Dz, Mies, Rok );
messageBox( 0, 0, "", T );
```

Zobacz: *getTime*

getProfile (2.1)

Składnia:

getProfile(sect, key, dest, def, filename);

gdzie: sect, key – łańcuch znakowy lub zmienna typu *char*
dest – zmienna typu *int* lub *char*
def – literał lub zmienna o typie identycznym jak zmienna dest
filename – łańcuch znakowy lub zmienna typu *char*

Instrukcja służy do odczytywania ustawień zapisanych w plikach konfiguracyjnych typu *.ini* systemu Windows. Są to pliki tekstowe, w których poszczególne elementy konfiguracyjne są zgrupowane wewnątrz tzw. sekcji. Każdy element sekcji jest złożony z dwóch składników – nazwy, czyli tzw. klucza oraz właściwej wartości klucza. Przykładowa zawartość pliku konfiguracyjnego może wyglądać następująco:

```
[ Keyboard ]// nazwa sekcji
Keyb = eng.dll     // element o nazwie (kluczu) Keyb i wartości 'eng.dll'
Repeat = 10
```

Instrukcja *getProfile* służy właśnie do odczytania wartości klucza z sekcji *sect* i nazwie *key*. Wartość klucza jest zapisywana do zmiennej *dest*, w przypadku braku w podanej sekcji podanego klucza do zmiennej tej wpisywana jest wartość domyślna podana jako parametr *def*. Ostatnim parametrem jest fizyczna ścieżka do pliku konfiguracyjnego. Należy zauważyć, że instrukcja czyta zarówno wartości numeryczne, jak i łańcuchowe, typ odczytywanych danych jest określany na podstawie parametrów *dest* i *def*, wymagana jest tutaj oczywiście identyczność typów obu wartości.

Przykład:

```
getProfile( "Keyboard", "Keyb", Value, "", "baza.ini" );
```

Zobacz: *writeProfile*, *changeCategory*

getSheetActiveCell (2.3)

Składnia:

getSheetActiveCell(skoroszyt, arkusz, W, K)*;*

gdzie: skoroszyt, arkusz – łańcuchy tekstowe lub zmienne typu *char*
W, K – zmienne typu *int*

Instrukcja *getSheetActiveCell* pobiera współrzędne aktywnej komórki z podanego skoroszytu i arkusza (określonych odpowiednio parametrami *skoroszyt* oraz *arkusz*) i zapamiętuje w postaci indeksów w zmiennych *W* (wiersz) oraz *K* (kolumna). Wiersze i kolumny numerowane są od wartości 1 tzn. komórka B3 ma odpowiednio współrzędne 3, 2.

Zobacz: *getSheetActiveRange*

getSheetActiveRange (2.3)

Składnia:

getSheetActiveRange(skoroszyt, arkusz, zakres);

gdzie: skoroszyt, arkusz – łańcuchy tekstowe lub zmienne typu *char*
zakres – zmienna typu *char*

Instrukcja *getSheetActiveRange* pobiera ze skoroszytu określonego przez parametr *skoroszyt* i arkusza określonego parametrem *arkusz* wskazany zakres danych i zapamiętuje go w postaci formuły adresowej (na przykład "A1:C7"). Zakres może być wybrany za pomocą myszki lub klawiatury, a pobrana formuła adresowa może zostać użyta na przykład w celu utworzenia połączenia pomiędzy arkuszem a wykresem za pomocą instrukcji *linkChart2Sheet*.

Zobacz: *getSheetActiveCell*, *linkChart2Sheet*

getSheetRange (2.2)

Składnia:

getSheetRange(arkusz, str_arkusza, W1, K1, W2, K2, zmienna);

gdzie: arkusz – łańcuch tekstowy lub zmienna typu *char*
str_arkusza – łańcuch tekstowy lub zmienna typu *char* (param. nadmiarowy)
W1, K1, W2, K2 – zmienne lub stałe typu *int*
zmienna – tablica lub wektor dowolnego typu prostego

Instrukcja pobiera z arkusza o nazwie *arkusz* wartości z zakresu od *W1*, *K1* do *W2*, *K2* i umieszcza w tablicy *zmienna*. Wartości *Wn*, *Kn* oznaczają odpowiednio numer wiersza i numer kolumny. Tablica *zmienna* wypełniana jest wierszami.

Przykład:

```
int Wskaźniki[ 10 ];
getSheetRange( "Bilans", "", 1, 2, 7, 2, Wskaźniki );
// Wykonanie instrukcji spowoduje wpisanie
// do wektora wartości od wiersza 1 do 7 z kolumny nr 2
```

Zobacz: *closeRange*, *getSheetValue*, *openSheet*, *readSheet*, *setSheetRange*, *setSheetValue*, *showSheet*, *writeSheet*

getSheetValue (2.2)

Składnia:

getSheetValue(arkusz, str_arkusza, W, K, zmienna);

gdzie: arkusz – łańcuch znakowy lub zmienna typu *char*
str_arkusza – łańcuch znakowy lub zmienna typu *char* (param. nadmiarowy)
W1, K1, W2, K2 – zmienne lub stałe typu *int*
zmienna – zmienna dowolnego typu prostego

Instrukcja pobiera zawartość komórki w wierszu *W* i kolumnie *K* z arkusza *arkusz* i zapamiętuje ją w zmiennej *zmienna*.

Przykład:

```
float Wskaźnik szybki;
char Nazwisko;
getSheetValue( "Bilans", "", 4, 2, Wskaźnik szybki );
getSheetValue( "Bilans", "", 1, 1, Nazwisko );
```

Zobacz: *closeSheet*, *getSheetRange*, *openSheet*, *readSheet*, *setSheetRange*, *setSheetValue*, *showSheet*, *writeSheet*

getSource

Składnia:

getSource(X);

gdzie: X – symbol lub zmienna typu *char*

Instrukcja *getSource* należy do grupy instrukcji umożliwiających symulowanie niektórych elementów architektury tablicowej. Jej podstawowym zadaniem jest załadowanie i uaktywnienie źródła wiedzy typu *kb* o identyfikatorze *X*. Argument *X* określa lokalną nazwę źródła, określoną przez inżyniera wiedzy w bloku źródeł wiedzy. W związku z tym każde użycie źródeł wiedzy wymaga ich zadeklarowania w bloku *sources*. Jednorazowo dozwolone jest użycie do 10 różnych źródeł wiedzy typu *kb*.

Podstawową korzyścią z korzystania ze źródeł wiedzy jest możliwość podziału dużej bazy wiedzy na mniejsze części, na ogół wyodrębnione według tematyki. Jednocześnie architektura systemu PC-Shell umożliwia kooperację tych źródeł, poprzez rozwiązywanie określonych podproblemów i przekazywanie sobie rezultatów, aż do znalezienia ostatecznego rozwiązania.

Nie jest dozwolone użycie instrukcji *getSource* w odniesieniu do już aktywnego źródła wiedzy (wcześniej musi być ono zwolnione za pomocą instrukcji *freeSource*).

Przykład:

```
sources
   zr1:
       type kb
       file "zrodlo1.bw";
   zr2:
       type kb
       file "zrodlo2.bw";
end;

control
   getSource( zr1 );
   // ...
   freeSource( zr1 );
   getSource( zr2 );
   // ...
   freeSource( zr2 );
end;
```

getTime (2.2)

Składnia:

getTime(hour, min, sec, hun);

gdzie: hour (godz.), min (min.), sec (sek.), hun (setne sek.) – zmienne typu *int*

Instrukcja *getTime* zapamiętuje w zmiennych czas systemowy.

Przykład:

```
int Godz, Min, Sek, SSek;
char S;
getTime( Godz, Min, Sek, SSek );
sprintf( S, "Czas: %02d:%02d:%02d.%02d", Godz, Min, Sek, SSek);
messageBox( 0, 0, "", S );
```

Zobacz: *getDate*

getWindowType (2.2)

Składnia:

getWindowType (nazwa_okna, typ_okna);

gdzie: nazwa_okna – łańcuch znakowy lub zmienna typu *char*
typ_okna – zmienna typu *char*

Instrukcja służy do sprawdzenia typu okna określonego jako pierwszy parametr (*nazwa_okna*). Typ okna jest identyfikowany przez odpowiedni symbol (obecnie dostępne są jedynie okna arkuszy identyfikowane przez symbol *sheet* oraz okna wykresów identyfikowane symbolem *chart*).

Przykład:

```
char Okno, Typ;
getActiveWindow( Okno );
if ( Okno <> "" )
begin
   getWindowType( Okno, Typ );
   if ( Typ == "sheet" )
   begin
       writeSheet( Okno, "" );
   end;
   if ( Typ == "chart" )
   begin
       writeChart( Okno, "" );
   end;
end;
```

Zobacz: *getWindowType*

goal

Składnia:

goal(X);

gdzie: X – łańcuch znakowy lub zmienna typu *char*

Instrukcja *goal* inicjuje proces wnioskowania wstecz, którego zadaniem jest potwierdzenie danego celu (hipotezy). Podczas wnioskowania pojawia się okienko dialogowe o tej samej postaci jak w przypadku interakcyjnego trybu pracy z systemem. Parametr *X* określa treść celu.

Przykład:

```
// Przykłady poprawnego użycia instrukcji goal.
// Wszystkie podane sposoby użycia instrukcji goal
// są semantycznie równoważne.
char OAW, HIPO[ 5 ];
OAW := "diagnoza=X";
HIPO[ 0 ] := "diagnoza=X";
goal( "diagnoza=X" );
goal( OAW );
goal( HIPO[ 0 ] );
```

goto

Składnia:

goto etykieta;

gdzie: etykieta – symbol

Instrukcja *goto* umożliwia bezwarunkową zmianę sterowania i przejście od bieżąco wykonywanej instrukcji do instrukcji stojącej bezpośrednio po etykiecie wskazanej w instrukcji *goto*. Etykiety są symbolami zakończonymi znakami dwukropka.

Skok nie może następować do lub poza zakres objęty działaniem bloku, w którym jest bezpośrednio zawarta instrukcja *goto*.

Przykład:

```
control
   // ...
   goto skok_2;
   skok_1: goto skok_3;
   skok_2: goto skok_1;
   skok_3:
   // ...
end;
```

if

Składnia:

if (*warunek*) *instrukcja_złożona*;

if (*warunek*)
  instrukcja_złożona_1;
else
  instrukcja_złożona_2;

Instrukcja *if* wykorzystywana jest do warunkowej zmiany przebiegu sterowania. Umożliwia podejmowanie decyzji o wykonaniu lub niewykonaniu określonego zbioru instrukcji, zależnie od spełnienia zadanego w instrukcji warunku.

Postać 1:
Jeśli warunek jest spełniony, następuje wykonanie instrukcji *instrukcja_złożona*. W przeciwnym wypadku wykonana będzie instrukcja znajdująca się bezpośrednio za instrukcją *if*.

Postać 2:
Jeśli warunek jest spełniony, następuje wykonanie instrukcji *instrukcja_złożona_1*. W przeciwnym wypadku wykonana zostaje instrukcja *instrukcja_złożona_2*.

Dopuszczalne jest zagnieżdżanie instrukcji *if*.

Przykład:

```
// Przykład użycia instrukcji if (pierwsza postać)
if ( Ocena == "dostateczna" )
begin
   messageBox( 0, 0, "Ocena", "Nie przyznawać stypendium" );
end;
if ( Ocena == "bardzo dobry" )
begin
   messageBox( 0, 0, "Ocena", "Przyznać stypendium" );
end;

// Przykład użycia zagnieżdżonych instrukcji if
char S1, S2;
if ( DELTA < 0 )
begin
   messageBox( 0, 0, "Rozwiązanie", "Brak rozw. rzeczywistych");
end
else
begin
   if ( DELTA == 0 )
   begin
       S1 := "X = ";
       X := - B / ( 2 * A );
       ntos( S2, X );
       strcat( S1, S2 );
       messageBox( 0, 0, "Rozwiązanie", S1 );
   end
   else
   begin
       X1 := ( - B - sqrt( DELTA ) ) / ( 2 * A );
       X2 := ( - B + sqrt( DELTA ) ) / ( 2 * A );
       S1 := "X1 = ";
       ntos( S2, X1 );
```

```
        strcat( S1, S2 );
        strcat( S1, "   X2 = " );
        ntos( S2, X1 );
        strcat( S1, S2 );
        messageBox( 0, 0, "Rozwiązanie", S2 );
    end;
end;
```

initNetwork

Składnia:

initNetwork(X);

gdzie: X – symbol lub zmienna typu *char*

Instrukcja *initNetwork* powoduje wygenerowanie i zainicjowanie symulatora sieci neuronowej. Parametr *X* określa nazwę źródła wiedzy typu *neural_net*. Definicja zadeklarowanej sieci znajduje się w pliku określonym w bloku *sources*, utworzonym za pomocą systemu NEURONIX.

Deklaracja źródła wiedzy w postaci symulatora sieci neuronowej musi wystąpić w bloku *sources*.

Przykład:

```
sources
   siec_1:
       type neural_net
       file "siec_2.npr";
end;

control
   initNetwork( siec_1 );
   // ...
end;
```

isAppRunning (2.1)

Składnia:

isAppRunning(tytuł_okna, odp);

gdzie: tytuł_okna – nazwa okna
odp – zmienna typu *int*

Instrukcja *isAppRunnig* sprawdza, czy jest aktualnie otwarte okno mające w tytule nazwę *nazwa_okna*. Nazwa nie musi być dokładną nazwą okna, ważne jest tylko podanie stałej części nazwy okna np. Microsoft Excel. Jeżeli okno istnieje w systemie, wartość zmiennej *odp* ustawiana jest na *1*, w przeciwnym razie zmienna *odp* otrzyma wartość *0*.

Przykład:

```
// sprawdzamy, czy jest uruchomiona aplikacja MS-Excel;
// jeżeli nie to ją uruchamiamy
int Ret;
isAppRunning( "Microsoft Excel", Ret );
if ( Ret == 0 )
begin
   system( "C:\\EXCEL\\EXCEL" );
end;
```

Zobacz: *closeWindow*, *showWindow*, *system*

linkChart2Sheet (2.3)

Składnia:

linkChart2Sheet(wykres, arkusz, zakres, typ);

gdzie: wykres, arkusz, zakres – łańcuchy tekstowe lub zmienne typu *char*
typ – wartość 0, 1 lub 2 (jawnie lub jako zmienna)

Instrukcja *linkChart2Sheet* wiąże wykres określony parametrem *wykres* z danymi zawartymi w skoroszycie o nazwie zawartej w zmiennej *arkusz*. Kolejny parametr (*zakres*) określa źródło danych i powinien zostać podany zgodnie z formułą adresową o przykładowej postaci *Arkusz1!A1:C7*, przy czym jeżeli skoroszyt zawiera tylko jeden arkusz – nazwa arkusza może być pominięta (w tym przypadku formuła dotyczy pierwszego arkusza). Ostatni argument (*typ*) określa tryb połączenia z arkuszem: wartość *0* oznacza zamknięcie połączenia, *1* – połączenie wykresu z arkuszem (przy czym następuje jedynie aktualizacja wartości danych), *2* – połączenie oraz sformatowanie wykresu, uaktualnienie ilości wierszy i ilość kolumn na wykresie zgodnie z podanym zakresem. Przykład łączenia wykresu z arkuszem znajduje się w przykładowej bazie wiedzy o nazwie *link.bw*.

Zobacz: *setChartArray*, *setChartData*

mainMenu

Składnia:

mainMenu;

Instrukcja *mainMenu* powoduje bezwarunkowe zakończenie wykonania programu i przejście do głównego menu systemu PC-Shell.

makeOAV

Składnia:

makeOAV(O, A, V, T);

gdzie: O – symbol, zmienna typu *char* lub znak „_”
A – symbol lub zmienna typu *char*
V – liczba, łańcuch znakowy, zmienna numeryczna lub znakowa lub znak „_”
T – zmienna typu *char*

Instrukcja *makeOAV* umożliwia utworzenie łańcucha znakowego będącego trójką obiekt-atrybut-wartość (OAW). Parametr *O* jest identyfikatorem obiektu i może być pominięty (należy wtedy użyć znaku podkreślenia „_”). Parametr *A* jest identyfikatorem atrybutu i musi wystąpić. Parametr *V* reprezentuje wartość atrybutu w danej trójce OAW.

Właściwym kontekstem użycia tej instrukcji może być utworzenie trójki OAW dla późniejszego użycia jej w instrukcji *goal*. Łańcuch OAW stanowiący rezultat zapisywany jest w zmiennej *T*.

Z punktu widzenia semantyki instrukcja stanowi „odwrotność” instrukcji *splitOAV*.

Przykład:

```
// Użycie instrukcji makeOAV dla tworzenia - nie określonych
// "z góry" - hipotez
char ATTR, VAL, OAV;
seditBox( 0, 0, "Dane", "Identyfikator atrybutu:", 0, ATTR );
seditBox( 0, 0 ,"Dane", "Wartość atrybutu:", 1, VAL );
makeOAV( _, ATTR, VAL, OAV );
goal( OAV );
```

menu

Składnia:

menu tekst_0
1. tekst_1
2. tekst_2
n. tekst_n
case 1:
instrukcje_1
case 2:
instrukcje_2
case *n*:
instrukcje_n
end;

Instrukcja *menu* jest złożoną instrukcją sterującą, tworzącą hierarchiczny system menu oraz organizującą w strukturalny sposób proces sterowania. Łańcuchy znakowe, oznaczone jako *tekst_0*, …, *tekst_n*, określają nazwy menu oraz poszczególnych opcji menu. W obecnej wersji maksymalna liczba wariantów, czyli *n* równa się 10. Wyrażenia *case 1*, …, *case n* definiują procedury związane z każdym wariantem. Cały proces sterowania (kontroli wywołań, powrotów na wyższy poziom itp.) realizowany jest automatycznie przez system PC-Shell. Tworzenie złożonego systemu menu nie wymaga zatem dodatkowych czynności programistycznych ze strony użytkownika.

Działanie systemu menu, programowanego w systemie PC-Shell, polega na wykonaniu następujących czynności:

1. oczekiwaniu na wybranie przez użytkownika jednego z wariantów zawartych w okienku bieżącego menu;
2. wykonaniu zbioru instrukcji związanych z danym wariantem, tj. następujących po słowie *case* z numerem wybranego wariantu;
3. po wykonaniu wspomnianego zbioru instrukcji powrót do ponownego wykonania bieżącego menu.

Wyjście z bieżącego menu może wystąpić po wykonaniu jednej z następujących instrukcji: *exit*, *quit* lub *break*.

Przykład:

```
menu "Główne &zadania"
   1. "Ocena sytuacji &finansowej klienta banku"
   2. "Decyzje &kredytowe"
   3. "&Wyjście"
   case 1:
       goal( "sytuacja_finansowa=Syt_fin" );
       delNewFacts;
   case 2:
       goal( "decyzja=Decyzja" );
       delNewFacts;
   case 3:
       exit;
end;
```

messageBox

Składnia:

messageBox(X, Y, S1, S2);

gdzie: X, Y – liczba lub zmienna typu *int*
S1, S2 – symbole, łańcuchy znakowe lub zmienne typu *char*

Instrukcja *messageBox* powoduje wyprowadzenie informacji (komunikatu) reprezentowanego przez *S2* w formie okna z przyciskiem „OK”. Tekst *S1* pojawia się jako tytuł okna. *X* i *Y* oznaczają współrzędne punktu ekranu, w którym ma się pojawić lewy, górny róg okna. Napis jest wyświetlany do momentu naciśnięcia dowolnego klawisza lub „kliknięcia” myszką na przycisku.

Jeśli współrzędne są równe *0* – okno komunikatu wyświetlone zostanie w środku ekranu.

Przykład:

```
messageBox( 0, 0, "Tytuł okna", "Treść wiadomości" );
```

neditBox

Składnia:

neditBox(X, Y, W, Z, S, R);

gdzie: X, Y – liczby lub zmienne typu *int*
W, Z – liczby lub zmienne dowolnego typu numerycznego
R – zmienna typu *float*
S – symbol, łańcuch znakowy lub zmienna typu *char*

Instrukcja *neditBox* jest narzędziem do wprowadzania informacji liczbowej. Powoduje ona wyświetlenie okna zawierającego pole edycji liczb, tekst wiadomości dla użytkownika oraz przyciski „OK" oraz „Anuluj". Argumenty *X* i *Y* reprezentują współrzędne lewego, górnego rogu okna, natomiast *W* i *Z* określają dozwolony zakres wprowadzanych wartości (*W* – ograniczenie dolne, *Z* – ograniczenie górne), co umożliwia prostą kontrolę poprawności danych, w sposób automatyczny, bez konieczności programowania. Parametr *S* reprezentuje tekst będący informacją dla użytkownika, opisującą polecenie lub znaczenie wprowadzanej danej. Początkowa zawartość pola edycji zależy od wartości zmiennej *R* (jeżeli wartość zawiera się w dozwolonym zakresie – zostanie wpisana w polu edycyjnym; w przeciwnym razie pole będzie puste). Po wprowadzeniu przez użytkownika liczby jej wartość zostanie przypisana zmiennej reprezentowanej przez *R*. Typy zmiennych *W*, *Z* oraz *R* muszą być zgodne (tj. tego samego typu lub typu zgodnego z typem zmiennej *R*).

Przykład:

```
float Liczba;
neditBox( 0, 0, 1, 10 , "Podaj liczbę iteracji", Liczba );
```

nsheetBox

Składnia:

nsheetBox(X, Y, N, S, T, A, W);

gdzie: X, Y, N, S – liczby lub zmienne typu *int*
T, A – tablica typu *char*
W – tablica typu numerycznego

Instrukcja *nsheetBox* tworzy arkusz do wprowadzania danych liczbowych. Dane wprowadzane są w formie trzykolumnowej tabeli, zawierającej kolejno:

1. liczbę porządkową,
2. tekstowy opis danych, najczęściej podzbiór atrybutów bazy wiedzy,
3. wprowadzane dane.

Wartości *X*, *Y* ustalają współrzędne arkusza. Argument *N* określa liczbę pozycji (wierszy) arkusza danych. Jeśli wartością parametru S będzie 1, to poza wymienionymi trzema kolumnami pojawi się pozycja zawierająca bieżącą sumę wprowadzanych danych. Jej wartość jest aktualizowana po wprowadzeniu każdej danej.

T reprezentuje trójelementową tablicę służącą do przekazania trzech napisów, będących odpowiednio tytułami okna arkusza oraz drugiej i trzeciej kolumny.

Tablice *A* i *W* muszą zawierać tyle samo elementów. Tablica *A* przekazuje opis poszczególnych pozycji (wierszy) danych, natomiast tablica *W* przechowuje wprowadzone dane. Jeśli przed użyciem instrukcji *nsheetBox* tablica *W* zawierała już dane, to zostaną one przekazane do arkusza i pojawią się w trzeciej kolumnie.

Naciśnięcie klawisza „OK” powoduje zamknięcie okna i zapamiętanie wprowadzonych danych w tablicy *W*.

Przykład:

```
float Wart[ 3 ];
char Tytuł[ 3 ], NazPoz[ 3 ];
Tytuł[ 0 ] := "ARKUSZ WPROWADZANIA DANYCH";
Tytuł[ 1 ] := "Nazwa";
Tytuł[ 2 ] := "Wartość";
Npoz[ 0 ] := atrybut1;
Npoz[ 1 ] := atrybut2;
Npoz[ 2 ] := atrybut3;
nsheetBox( 0, 0, 3, 1, Tytul, NazPoz, Wart );
```

ntos

Składnia:

ntos(N, S);

gdzie: N – liczba lub zmienna numeryczna
S – zmienna typu *char*

Instrukcja *ntos* umożliwia przekształcenie wartości liczbowej reprezentowanej przez parametr *N* do postaci łańcucha znakowego i przypisanie go do zmiennej *S*.

Przykład:

```
char S;
float L;
L := -150;
ntos( 123.45, S );  // S przyjmuje wartość "123.45"
ntos( L, S );       // S przyjmuje wartość "-150"
```

Zobacz: *c_ntos*

oleCreateObject (4.0)

Składnia:

oleCreateObject(V, Name);

gdzie: V –zmienna typu *variant*
Name – znak, łańcuch znakowy lub zmienna typu *char*

Instrukcja *oleCreateObject* powoduje utworzenie nowego obiektu OLE Automation o nazwie klasy zdefiniowanej w parametrze *Name*. Po prawidłowym utowrzeniu w zmiennej *V* zapamiętywany jest identyfikator tego obiektu.

Przykład:

```
// Utworzenie obiektu MS-Word
variant WordApplication;
oleCreateObject( WordApplication, "Word.Application" );
```

Zobacz: *oleFunction, oleProcedure, olePropertyGet, olePropertySet*

oleFunction (4.0)

Składnia:

oleFunction(V, FnName, Dst, ...);

gdzie: V –zmienna typu *variant*
FnName – znak, łańcuch znakowy lub zmienna typu *char*
Dst –zmienna dowolnego typu, w szczególności *variant*
pozostałe parametry dowolnego typu i w dowolnej ilości

Instrukcja *oleFunction* powoduje wywołanie funkcji serwera OLE Automation. Pierwszy parametr jest uchwytem utworzonego wcześniej obiektu, *FnName* to nazwa funkcji, *VDst* – zmienna do której będzie wpisany wynik wykonania funkcji. Pozostałe parametry są parametrami wywoływanej funkcji i zależne są od składni funkcji serwera OLE Automation

Przykład:

```
// Utworzenie obiektu MS-Word
variant WordApplication, WordDocuments, WordDocument;
oleCreateObject( WordApplication, "Word.Application" );
olePropertyGet( WordApplication, "Documents", WordDocuments );

// Dodanie nowego - pustgo dokumentu
oleFunction( WordApplication, "Add", WordDocument );
```

Zobacz: *oleCreateObject, oleProcedure, olePropertyGet, olePropertySet*

oleProcedure (4.0)

Składnia:

oleProcedure(V, FnName,...);

gdzie: V –zmienna typu *variant*
FnName – znak, łańcuch znakowy lub zmienna typu *char*
pozostałe parametry dowolnego typu i w dowolnej ilości

Instrukcja *oleProcedure* powoduje wywołanie procedury serwera OLE Automation. Pierwszy parametr jest uchwytem utworzonego wcześniej obiektu, *FnName* to nazwa procedury. Pozostałe parametry są parametrami wywoływanej procedury i zależne są od składni procedury serwera OLE Automation.

Przykład:

```
// Utworzenie obiektu MS-Word
variant WordApplication, WordDocuments, WordDocument,
        WordSelection;
oleCreateObject( WordApplication, "Word.Application" );
olePropertyGet( WordApplication, "Documents", WordDocuments );

// Dodanie nowego - pustgo dokumentu
oleFunction( WordApplication, "Add", WordDocument );
olePropertyGet( WordApplication, "Selection", WordSelection );
oleProcedure( WordSelection, "TypeText", "Tekst wstawiany do
dokumentu programu Microsoft Word" )
```

Zobacz: *oleCreateObject, oleFunction, olePropertyGet, olePropertySet*

olePropertyGet (4.0)

Składnia:

olePropertyGet(V, PropName, Dst, ...);

gdzie: V –zmienna typu *variant*
PropName – znak, łańcuch znakowy lub zmienna typu *char*
Dst –zmienna dowolnego typu, w szczególności *variant*
pozostałe parametry dowolnego typu i w dowolnej ilości

Instrukcja *olePropertyGet* powoduje pobranie wartości właściwości obiektu serwera OLE Automation. Pierwszy parametr jest uchwytem utworzonego wcześniej obiektu, *PropName* to nazwa właściwości, *VDst* – zmienna do której będzie wpisana wartość właściwości, może to być zmienna dowolnego typu. Pozostałe parametry są dodatkowymi parametrami wywoływanej właściwośc i zależne są od serwera OLE Automation.

Przykład:

```
// Utworzenie obiektu MS-Word
variant WordApplication, WordDocuments, WordDocument,
        WordSelection;
oleCreateObject( WordApplication, "Word.Application" );
olePropertyGet( WordApplication, "Documents", WordDocuments );

// Dodanie nowego – pustgo dokumentu
oleFunction( WordApplication, "Add", WordDocument );
olePropertyGet( WordApplication, "Selection", WordSelection );
oleProcedure( WordSelection, "TypeText", "Tekst wstawiany do
dokumentu programu Microsoft Word" )
```

Zobacz: *oleCreateObject, oleFunction, oleProcedure, olePropertySet*

olePropertySet (4.0)

Składnia:

olePropertySet(V, PropName, Val, ...);

gdzie: V –zmienna typu *variant*
PropName – znak, łańcuch znakowy lub zmienna typu *char*
Val –wartość lub zmienna dowolnego typu
pozostałe parametry dowolnego typu i w dowolnej ilości

Instrukcja *olePropertySet* powoduje ustawienie wartości właściwości obiektu serwera OLE Automation. Pierwszy parametr jest uchwytem utworzonego wcześniej obiektu, *PropName* to nazwa właściwości, *Val* – wartość jaką ma przyjąć podana właściwość, może to być wartość numeryczna, tekstowa lub zmienna typu variant. Pozostałe parametry są dodatkowymi parametrami wywoływanej właściwości i zależne są od serwera OLE Automation.

Przykład:

```
variant WordApplication;
oleCreateObject( WordApplication, "Word.Application" );
// Spowodowanie że aplikacja Microsoft Word jest widoczna
olePropertySet(WordApplication,"Visible", 1);
```

Zobacz: *oleCreateObject, oleFunction, oleProcedure, olePropertyGet*

open

Składnia:

open(X, Y);

gdzie: X – symbol lub zmienna typu *char*
Y – znak, łańcuch znakowy lub zmienna typu *char*

Instrukcja *open* powoduje otwarcie pliku o nazwie określonej przez parametr *X* w trybie określonym przez parametr *Y*. Nazwa pliku może być uzupełniona o ścieżkę dostępu.

Dozwolone są następujące tryby otwarcia:

r - do czytania (ang. *read*),
w - do pisania (ang. *write*),
a do rozszerzania (ang. *append*).

Przykład:

```
// Otwarcie pliku o nazwie "dane" do wprowadzania (czytania):
open( dane, r );
```

Zobacz: *close*, *fput*, *fseek*, *fgetc*, *fgets*, *fget*, *readMat*

openChart (2.3)

Składnia:

openChart(wykres, szablon);

gdzie: wykres, szablon – łańcuch tekstowy lub zmienna typu *char*

Instrukcja tworzy nowy wykres o nazwie *Wykres* w pamięci, ustawiając jego początkową postać zgodnie z zawartością pliku szablonu zapisanego w pliku o nazwie *PlikSzablonu*. Aby utworzyć widok wykresu należy po utworzeniu wykresu wywołać instrukcję showChart.

Przykład:

```
openChart( "Wykres płynności", "" );
showChart( "Wykres płynności", 0 );
```

Zobacz: *closeChart*, *showChart*

openSheet (2.2)

Składnia:

openSheet(arkusz, plik_szablonu);

gdzie: arkusz, plik_szablonu – łańcuchy znakowe lub zmienne typu *char*

Instrukcja tworzy nowy arkusz o nazwie określonej parametrem *arkusz* i definiuje jego postać początkową na podstawie wzorca zapisanego w pliku o nazwie zawartej w zmiennej *plik_szablonu*. Arkusz zostaje utworzony w systemie i od tego momentu jest on dostępny pod nazwą określoną przez argument *arkusz*. W celu wyświetlenia arkusza na ekranie należy wywołać instrukcję *showSheet*, która powoduje utworzenie „widoku" arkusza w postaci okna. Zamknięcie widoku przez użytkownika nie powoduje usunięcia arkusza z pamięci, jest on nadal obecny w systemie do momentu wywołania instrukcji *closeSheet* lub zamknięcia okna aplikacji.

Obecna wersja systemu dopuszcza, aby wzorzec określony w pliku *plik_szablonu* zapisany był w formacie FormulaOne (*.vts*) lub MS-Excel 5.0 (*.xls*).

Uwaga! Arkusze mogą być tworzone jedynie w momencie istnienia okna aplikacji systemu PC-Shell, które zarządza arkuszami.

Przykład:

```
openSheet( "Bilans", "bilans.vts" );
showSheet( "Bilans", 1 );
```

Zobacz: *closeSheet*, *getSheetRange*, *getSheetValue*, *readSheet*, *setSheetRange*, *setSheetValue*, *showSheet*, *writeSheet*

paramWindow (2.2)

Składnia:

paramWindow;

Instrukcja *paramWindow* powoduje wyświetlenie okna parametryzacji, umożliwiającego definiowanie kategorii zmiennych parametrycznych.

playSound (2.3)

Składnia:

playSound(Plik);

gdzie: *Plik* - łańcuch lub zmienna typu **char**

Instrukcja odtwarza dźwięk zapisany w pliku *Plik*. Dopuszczalny format pliku dźwiękowego to pliki typu wave.

Przykład:

```
playSound( "tada.wav" );
```

precision

Składnia:

precision(X, Y);

gdzie: X, Y – liczby lub zmienne typu *int*

Instrukcja *precision* ustala dokładność z jaką prezentowane są przez system wartości liczbowe. Dotyczy to między innymi liczb pojawiających się w wyjaśnieniach, podczas przeglądania bazy wiedzy w opcji przeglądania (ang. *browser*), a także w zapytaniach systemu podczas konsultacji. Standardowo założona jest dokładność do dwóch miejsc po przecinku.

Parametr *X* określa minimalną szerokość pola w znakach, natomiast parametr *Y* ustala liczbę znaków (miejsc) po przecinku (kropce dziesiętnej).

quit

Składnia:

quit;

Instrukcja *quit* powoduje bezwarunkowe opuszczenie systemu PC-Shell.

random (2.1)

Składnia:

random(R, X);

gdzie: R – liczba lub zmienna typu *int*
X – zmienna typu *int*

Instrukcja generuje liczbę pseudolosową z przedziału od 0 do *R*-1 i zapamiętuje ją w zmiennej *X*.

readChart (2.3)

Składnia:

readChart(wykres, nazwa_pliku);

gdzie: wykres, plik – łańcuchy znakowe lub zmienne typu *char*

Instrukcja *readChart* wczytuje do wykresu o nazwie zawartej w zmiennej *wykres* zawartość pliku określonego parametrem *nazwa_pliku*. Plik musi być w formacie Visual Formula Chart (**.vtc*) – format ten opisuje definicję wykresu. Instrukcja *readChart* służy między innymi do wczytania plików zapamiętanych instrukcją *writeChart*.

Zobacz: *closeChart*, *openChart*, *showChart*, *writeChart*

readMat

Składnia:

readMat(X);

gdzie: X – identyfikator tablicy (bez indeksu)

Instrukcja *readMat* powoduje odczytanie danych z bieżącego pliku typu tekstowego i przypisanie ich kolejnym elementom tablicy *X*. Dane powinny być typu zgodnego z deklaracją typu tablicy, a ich liczba nie większa od rozmiaru tablicy. Dane muszą być rozdzielone spacjami lub znakami nowej linii.

Plik musi być wcześniej otwarty za pomocą instrukcji *open*. W przypadku błędu podczas czytania pliku wartość zmiennej *RETURN* jest ustawiana na 0, bez wyświetlenia komunikatu.

Przykład:

Zawartość pliku "dane": 111 222 333 444

```
// Deklaracja tablicy
int Tab[ 4 ];
// Użycie instrukcji odczytu danych z pliku do tablicy
open( dane, r );
readMat( TAB );
close( dane );
```

Po wykonaniu podanego ciągu instrukcji wartości poszczególnych elementów tablicy będą następujące:

TAB[0] = 111, TAB[1] = 222, TAB[2] = 333, TAB[3] = 444.

readSheet (2.2)

Składnia:

readSheet(arkusz, nazwa_pliku);

gdzie: arkusz – łańcuch lub zmienna typu *char*
nazwa_pliku – łańcuch lub zmienna typu *char*

Instrukcja wczytuje do arkusza o nazwie *arkusz* zawartość pliku *nazwa_pliku*. Akceptowane pliki to pliki w formacie arkusza FormulaOne (**.vts*) oraz pliki w formacie MS-Excel 5.0 (**.xls*). W przypadku pozostałych typów plików nastąpi próba przeczytania pliku jako pliku tekstowego (kolejne wiersze z pliku zostaną wpisane do kolejnych wierszy arkusza w pierwszej kolumnie).

Przykład:

```
readSheet( "Bilans", "bilans.vts" );
readSheet( "Płace", "place.xls" );
```

Zobacz: *closeSheet*, *getSheetRange*, *getSheetValue*, *openSheet*, *setSheetRange*, *setSheetValue*, *showSheet*, *writeSheet*

return (2.2)

Składnia:

return ret_val;

gdzie: ret_val – wartość numeryczna.

Instrukcja *return* powoduje natychmiastowe wyjście z funkcji, dlatego może być użyta jedynie wewnątrz funkcji. Jako rezultat przekazuje wartość *ret_val*, która zostaje przypisana globalnej zmiennej systemowej *RETURN*.

Zobacz: *function*

run

Składnia:

run;

Instrukcja *run* ma w zasadzie charakter deklaratywny. Jej użycie powoduje, że program będzie wykonywany bezpośrednio po załadowaniu bazy wiedzy (bez konieczności użycia opcji *Program* z głównego menu systemu PC-Shell). Instrukcja *run* może pojawić się w dowolnym miejscu programu.

runNetwork

Składnia:

runNetwork(net, X, Y);

gdzie: net – nazwa sieci
X, Y – tablice rekordów typu *NeuralNet* lub tablice typu *float* lub *double*

Instrukcja *runNetwork* powoduje uruchomienie symulatora sieci neuronowej zainicjowanego wcześniej za pomocą instrukcji *initNetwork*. Parametr *X* określa wektor wejściowy, natomiast *Y* oznacza wektor wyjściowy do sieci neuronowej. Począwszy od wersji 2.2 możliwe jest zainicjowanie dwu lub więcej sieci na raz. Sieci identyfikowane są za pomocą nazwy zadeklarowanej w bloku *sources*.

Rozróżnia się dwa typy wywołań sieci. Pierwszy z nich (zalecany, wykorzystujący nowe możliwości symulatora Neuronix) wykorzystuje wektory (tablice) rekordów *NeuralNet*. W tym przypadku należy odpowiednio nazwać wszystkie wejścia i przypisać im odpowiednie wartości wejściowe (liczbowe lub symboliczne). Podobnie w wektorze wyjściowym muszą być nazwane wyjścia. Budowa rekordu *NeuralNet* jest następująca:

```
record NeuralNet
begin
  char Name;       // nazwa wejścia lub wyjścia
  double DValue;   // wartość numeryczna
  char Symbol;     // wartość symboliczna
end;
```

Rekord ten jest predefiniowany w systemie. Typy wartości muszą odpowiadać typom określonym w symulatorze sieci neuronowych w trakcie tworzenia sieci. W momencie wywołania ilość wejść musi odpowiadać rzeczywistej ilości wejść sieci (wszystkie wejścia muszą być określone) natomiast ilość wyjść może być mniejsza niż w rzeczywistej sieci.

Drugi typ wywołania, służący zapewnieniu kompatybilności z poprzednią wersją, dopuszcza podanie na wejściu i wyjściu tablic numerycznych (*float* lub *double*) bez podania nazw wejść i wyjść. W tym przypadku należy pamiętać, że wektory te muszą swoimi rozmiarami odpowiadać rozmiarom określonym w momencie tworzenia sieci. Należy również pamiętać, że wejścia i wyjścia w systemie Neuronix są sortowane według nazw wejść i wyjść – i tak posortowane odpowiadają kolejnym pozycją wektorów wejściowych lub wyjściowych.

Przykład:

Pierwszy typ wywołania sieci:

```
record NeuralNet SinWe[ 3 ];  // wejścia
record NeuralNet SinWy[ 1 ];  // wyjście
SinWe[ 0 ].Name := "y-1";
SinWe[ 0 ].DValue := 1;
SinWe[ 1 ].Name := "y-2";
SinWe[ 1 ].DValue := 1;
SinWe[ 2 ].Name := "y-3";
SinWe[ 2 ].DValue := 0.0;
SinWy[ 0 ].Name := "y";
initNetwork( netSin );  // zainicjowanie sieci
runNetwork( netSin, SinWe, SinWy ); // wykonanie obliczeń
delNetwork( netSin );   // usunięcie sieci
// wyprowadzenie wyniku
char SSinWy;
precision( 10,8 );
ntos( SinWy[ 0 ].DValue, SSinWy );
```

```
messageBox( 0, 0, "Wartosc wyjscia Sinus", SSinWy );
```

Drugi typ wywołania sieci:

```
float WEJŚCIE[ 10 ], WYJŚCIE[ 5 ], I;
for I := 0 to 9 step 1
begin
   neditBox( 0, 0, "Dane", "Element wektora", WEJŚCIE[ I ] );
end;
initNetwork( siec_1 );
runNetwork( siec_1, WEJŚCIE, WYJŚCIE );
for I := 0 to 4 step 1
begin
   fput( WYJŚCIE[ I ] );
end;
```

saveExplan (2.15)

Składnia:

saveExplan(dest, O, A, V, N);

gdzie: dest – zmienna typu *char*
O – symbol, zmienna typu *char* lub znak „_”
A – symbol lub zmienna typu *char*
V – liczba, łańcuch znakowy, zmienna numeryczna, tekstowa lub znak „_”
N – liczba całkowita lub zmienna typu *int*

Instrukcja powoduje zapamiętanie w zmiennej *dest* tekstu wyjaśnień typu „jak?”, wygenerowanych w czasie wnioskowania. Argumenty *O*, *A*, *V* określają wartości trójki obiekt-atrybut-wartość (OAW), której mają dotyczyć wyjaśnienia. Argumenty *O* oraz *V* mogą być nieokreślone (znak „_”) co oznacza „dla dowolnej wartości”. Poszukiwanie wyjaśnień polega na przeglądaniu faktów w bazie wiedzy i sprawdzaniu czy kolejny fakt pasuje do podanego wzorca. Ostatni argument (*N*) określa, dla którego kolejnego faktu zapamiętać wyjaśnienia. Podanie wartości *-1* oznacza „dla wszystkich” – wyjaśnienia są w tym przypadku łączone jedną całość.

W przypadku, kiedy brak jest faktów oraz wyjaśnień „jak?” dla podanego wzorca, zmienna *dest* zawierać będzie pusty łańcuch.

Uwaga! Po instrukcjach *goal* oraz *addSolution*(*no*) użycie instrukcji *saveExplan* nie spowoduje pobrania żądanego tekstu wyjaśnień „jak?”, ponieważ rozwiązanie wraz z wyjaśnieniami nie zostanie w takim przypadku dodane do bazy wiedzy. Aby uzyskać tekst wyjaśnień należy używać instrukcji *solve* lub przed instrukcją *goal* wywołać instrukcję *addSolution*(*yes*).

Przykład:

```
saveExplan( STR, _, "grzyby", _, -1 );
```

Zobacz: *fsaveExplan*, *saveWhatIs*

saveSolution

Składnia:

saveSolution(X, Y);

gdzie: X – identyfikator tablicy typu *char* (bez indeksu)
Y – zmienna typu *int*

Instrukcja *saveSolution* umożliwia zachowanie w tablicy *X* rozwiązania problemu, znalezionego przez system podczas konsultacji. Rozwiązanie może być wygenerowane w rezultacie użycia instrukcji *goal*. Do tablicy reprezentowanej tu przez parametr *X* wpisywane są, w postaci łańcuchów znakowych wszystkie rozwiązania danego problemu, a zmienna *Y* przyjmuje wartość równą liczbie tych rozwiązań. Deklarując rozmiar tablicy *X* należy przewidzieć maksymalną liczbę rozwiązań, które mogą zostać wygenerowane dla danego problemu.

Przykład:

```
// Przechowanie rozwiązań w tablicy ROZW
char ROZW[ 10 ];
int LR;
goal( "organizm=ORG" );
saveSolution( ROZW, LR );
```

saveWhatIs (2.15)

Składnia:

saveWhatIs(Dest, O, A, W);

gdzie: Dest – - zmienna typu **char**
O - symbol lub zmienna typu **char** lub znak " _ ",
A - symbol lub zmienna typu **char**,
V - liczba, łańcuch znakowy, zmienna numeryczna, łańcuchowa lub znak "_".

Instrukcja służy do zapamiętania w zmiennej *Dest* tekstu wyjaśnień ***Co to?*** Dla trójki OAW określonej odpowiednio w argumentach *O, A, V*. Należy tutaj zaznaczyć, że podanie wartości nieokreślonych w argumentach *O* lub *V* musi być *zsynchronizowane* ze sposobem zbudowania wyjaśnień ***Co to?*** w programie CAKE.

Przykład :

```
// w zmiennej STR będą zapamiętane wyjaśnienia WhatIs
//  dla trójki grzyb = "pieczarka" (jeżeli są określone) :
saveWhatIs( STR, _, "grzyb", "pieczarka" );
```

Zobacz : *saveExplan*

seditBox

Składnia:

seditBox(X, Y, S1, S2, T, R);

gdzie: X, Y – liczba lub zmienna typu *int*
T – liczba *0* lub *1*
R – zmienna typu *char*
S1, S2 – symbol, łańcuch znakowy lub zmienna typu *char*

Instrukcja *seditBox* służy do wprowadzania informacji tekstowej. Powoduje ona wyświetlenie okna zawierającego pole edycji tekstu, napis, będący wiadomością dla użytkownika oraz przyciski „OK” i „Anuluj”. Parametry *X* i *Y* reprezentują współrzędne lewego, górnego narożnika okna. Argument *S1* określa tekst, będący tytułem okna tworzonego przez instrukcję *seditBox*, natomiast *S2* – tekst zawierający informację dla użytkownika, dotyczącą polecenia lub znaczenia wprowadzanej danej.

Parametr *T* oznacza tryb pracy:

T=0 – zapobieżenie wprowadzeniu przez użytkownika pustego łańcucha (błąd);

T=1 – akceptowanie pustego łańcucha tekstowego.

Po wprowadzeniu tekstu jego treść zostanie przypisana zmiennej *R*.

Przykład:

```
char STR;
seditBox( 0, 0, "Dane", "Nazwa pliku:", 0, STR );
seditBox( 0, 0, "Dane", "Pełna ścieżka dostępu:", 1, STR );
```

setAppWinTitle

Składnia:

setAppWinTitle(X);

gdzie: X – łańcuch znaków lub zmienna typu *char*

Instrukcja *setAppWinTitle* ustala tytuł okna aplikacji systemu PC-Shell. Tekst tytułu jest przekazywany za pomocą argumentu *X*. Jeśli instrukcja nie wystąpi, otwarte wcześniej okno aplikacji (zob. instrukcja *createAppWindow*) zostanie automatycznie opatrzone etykietą „Aplikacja systemu PC-Shell”.

Przykład:

```
setAppWinTitle( "Klasyfikacja grzybów" );
```

setChartArray (2.3)

Składnia:

setChartArray(wykres, tablica_danych);

gdzie: wykres – łańcuch znakowy lub zmienna typu *char*
tablica_danych – tablica dowolnego typu numerycznego

Instrukcja *setChartArray* powoduje ustawienie poszczególnych punktów wskazanego wykresu zgodnie z rozmiarem tablicy oraz przepisuje wartości z tablicy bezpośrednio do wykresu. Ustawiana jest ilość wierszy i ilość kolumn oraz odpowiednie wartości punktów na wykresie.

Przykład:

```
double Tablica[ 2, 3 ];
Tablica[ 0, 0 ] := 1;
Tablica[ 0, 1 ] := 2;
Tablica[ 0, 2 ] := 4;
Tablica[ 1, 0 ] := 1;
Tablica[ 1, 1 ] := 4;
Tablica[ 1, 2 ] := 3;
setChartArray( "Wykres", Tablica );
```

Zobacz: *setChartData*

setChartData (2.3)

Składnia:

setChartData(wykres, dana, wartość);

gdzie: wykres, dana – łańcuch znakowy lub zmienna typu *char*
wartość – stała lub zmienna typu *char* lub typu numerycznego

Instrukcja *setChartData* jest instrukcją służącą do programowej zmiany ustawień istniejącego w pamięci wykresu. Za jej pomocą można określić szereg opcji (patrz tabela poniżej). Drugi z parametrów (*dana*) określa typ danych który mamy zamiar zmienić, a trzeci parametr *Wartość* oznacza odpowiednią nową wartość danej. Poniżej w tabeli przedstawiamy dopuszczalne typy danych i dopuszczalne dla nich wartości.

Nazwa danej	Opis	Dopuszczalne wartości
Title	Tekst tytułu wykresu	Dowolny łańcuch tekstowy (pusty usuwa pole)
Footnote	Tekst pola stopki wykresu	Dowolny łańcuch tekstowy (pusty usuwa pole)
Rows	Ilość wierszy na wykresie	Wartość liczbowa całkowita
Cols	Ilość kolumn na wykresie	Wartość liczbowa całkowita
Chart Type	Typ wykresu	Jeden z poniższych łańcuchów tekstowych: *3d Bar* *2d Bar* *3d Line* *2d Line* *3d Area* *2d Area* *3d Step* *2d Step* *3d Combination* *2d Combination* *3d Horizontal Bar* *2d Horizontal Bar* *3d Clustered Bar* *3d Pie* *2d Pie* *3d Doughnut* *2d XY* *2d Polar* *2d Radar* *2d Bubble* *2d HiLo* *2d Gantt* *3d Gantt* *3d Surface* *2d Contour* *3d Scatter* *3d XYZ*
Show Legend	Określa czy ma być wyświetlone pole legendy	1 – legenda 0 – bez legendy
X Axis Title *Y Axis Title* *Z Axis Title* *Second Y Axis Title*	Określa tekst oraz czy ma być wyświetlone pole określające tytuł jednej z kategorii	Dowolny łańcuch tekstowy (pusty usuwa pole)
xxxx Font	Rodzaj czcionki dla wybranego pola. *xxxx* może być jednym z poniższych: *Title*	Czcionka zakodowana w postaci łańcucha tekstowego (patrz niżej)

	Footnote *Legend* *X Axis Title* *Y Axis Title* *Z Axis Title* *Second Y Axis Title* *Rows* *Cols*	
[R,C]	Wartość danej w wierszu *R* i kolumnie *C*	Wartość liczbowa
Rx	Tekst nagłówka wiersza *x*	Łańcuch tekstowy
Cx	Tekst nagłówka kolumny *x*	Łańcuch tekstowy

W przypadku instrukcji *setChartData* zastosowany został specjalny format kodowania rodzaju czcionki. Format ten opiera się na zasadzie kodowania parametrów czcionki w postaci łańcucha tekstowego postaci:

"nazwa_czcionki, rozmiar_czcionki, atrybuty"

gdzie:

- nazwa_czcionki – nazwa dowolnej czcionki zainstalowanej w systemie;
- rozmiar_czcionki – rozmiar czcionki wyrażony w punktach;
- atrybuty – atrybuty czcionki (B – pogrubienie, I – kursywa, U – podkreślenie, S – przekreślenie, C:rrggbb – kolor tekstu, gdzie rrggbb określa kolor na podstawie zapisanych szestnastkowo wartości trzech barw podstawowych modelu RGB: czerwonego, zielonego, niebieskiego).

Zgodnie z powyższym łańcuch "Arial CE,12,B,C:FF0000" będzie opisywał czcionkę Arial CE o rozmiarze 12 punktów, wytłuszczoną, o kolorze czerwonym.

Przykład:

```
setChartData( "Wykres", "Rows", 2 );   // ilość wierszy
setChartData( "Wykres", "Cols", 3 );   // ilość kolumn
setChartData( "Wykres", "[1,1]", 1 );  // wartość punktu 1,1
setChartData( "Wykres", "[1,2]", 2 );
setChartData( "Wykres", "[1,3]", 4 );
setChartData( "Wykres", "[2,1]", 1 );
setChartData( "Wykres", "[2,2]", 4 );
setChartData( "Wykres", "[2,3]", 3 );
setChartData( "Wykres", "ShowLegend", 0 );  // legenda
setChartData( "Wykres", "Legend Font", "Arial CE,16,B" );
  // rodzaj czcionki dla tekstu legendy
```

Zobacz: *setChartArray*

setDefParamValue (2.1)

Składnia:

setDefParamValue(var);

gdzie: var – zmienna parametryczna typu *float* lub *char*

Instrukcja powoduje przypisanie zmiennej parametrycznej *var* wartości domyślnej, zdefiniowanej w bloku faset.

Zobacz: *changeCategory*

setDialogPos (2.2)

Składnia:

setDialogPos(dlg_id, X, Y, W, H);

gdzie: dlg_id – zmienna typu *int*
X, Y, W, H – liczby całkowite typu *int*

Instrukcja służy ustaleniu pozycji wyświetlenia okna dialogowego utworzonego za pomocą instrukcji *dlgCreate*. Parametr przekazany w instrukcji *dlgCreate* należy podać jako parametr wywołania (*dlg_id*). Kolejne parametry określają pozycję oraz rozmiary okna. Podanie wartości *0* jako parametru *X* lub *Y* spowoduje wycentrowanie okna na ekranie; podanie wartości *0* dla parametrów *W* i *H* spowoduje przyjęcie wartości domyślnych (określone przy projektowaniu okna).

Przykład:

```
int DLG, RET;
dlgCreate( DLG, "dlgs.dll", 5 );  //przygotowanie okna
setDialogPos( DLG, 0, 0, 0, 0 );  //centrowanie okna na ekranie
dlgExecute( DLG, RET );           //uaktywnienie okna
```

setSheetRange (2.2)

Składnia:

setSheetRange(arkusz, str_arkusza, W1, K1, W2, K2, zmienna);

gdzie: arkusz – łańcuch tekstowy lub zmienna typu *char*
str_akusza – łańcuch tekstowy lub zmienna typu *char* (param. nadmiarowy)
W1, K1, W2, K2 – zmienna lub stała typu całkowitego (*int*)
zmienna – tablica lub wektor dowolnego typu prostego

Instrukcja wpisuje do arkusza o nazwie określonej parametrem *arkusz* wartości od zakresu W1, K1 do W2,K2 z tablicy (wektora) stanowiącej argument *zmienna*. Wartości *Wn*, *Kn* oznaczają odpowiednio: numer wiersza i numer kolumny.

Przykład:

```
// Przepisanie wartości z kolumny 2 do kolumny 3
int Wskaźniki[ 10 ];
getSheetRange( "Bilans", "", 1, 2, 7, 2, Wskaźniki );
setSheetRange( "Bilans", "", 1, 3, 7, 3, Wskaźniki );
```

Zobacz: *showSheet, closeSheet*, *getSheetRange*, *getSheetValue*, *openSheet*, *readSheet*, *writeSheet*, *setSheetValue*

setSheetValue (2.2)

Składnia:

setSheetValue(arkusz, str_arkusza, W, K, wartość);

gdzie: arkusz – łańcuch tekstowy lub zmienna typu *char*
str_arkusza – łańcuch tekstowy lub zmienna typu *char* (param. nadmiarowy)
W1, K1, W2, K2 – zmienna lub stała typu całkowitego (*int*)
wartość – wartość lub zmienna dowolnego typu prostego

Instrukcja zapisuje wartość, określoną parametrem *wartość*, we wskazanej komórce (*W* – wiersz, *K* – kolumna) arkusza określonego przez parametr *arkusz*.

Przykład:

```
setSheetValue( "Płace", "", 1, 2, "Kowalski Jan" );
setSheetValue( "Płace", "", 1, 3, 1300 );
```

Zobacz: *showSheet, closeSheet*, *getSheetRange*, *getSheetValue*, *openSheet*, *readSheet*, *writeSheet*, *setSheetRange*

setSysText (2.2)

Składnia:

setSysText(element, tekst);

gdzie: e*lement*, *tekst* – zmienne lub łańcuchy znakowe

Dopuszczalne wartości parametru *element* to *problem* oraz *notConfirmed.*

Instrukcja służy do zmiany domyślnych ustawień wartości systemu PC-Shell. Możliwe do zmiany są dwa parametry: *problem* oraz *notConfirmed*. Parametr *problem* określa postać tekstu wyświetlanego w oknie konsultacji oraz w oknie rozwiązania. Domyślnie system PC-Shell wyświetla w tym miejscu tekst postawionej hipotezy, natomiast dzięki instrukcji *setSysText* można go zmienić na dowolny tekst, przybliżający w sposób opisowy cel wnioskowania.

Druga możliwa wartość zmieniana w systemie to wartość *notConfirmed,* która określa tekst, który pojawi się na liście w oknie rozwiązań. Domyślnie wyświetlany jest tekst „Hipoteza niepotwierdzona”.

Wartości tekstu w obu przypadkach mogą zawierać znaki sterujące sposobem wyświetlania tekstu. Dokładne objaśnienie znaków sterujących zamieszczono w rozdziale 3 „Podręcznika inżyniera wiedzy” (część 2. dokumentacji).

Przykład:

```
setSysText( problem, "Określenie profilu klienta" );
setSysText( notConfirmed, "[B[1Skonsultuj z przełożonym[0[b" );
solve( źródło1, "decyzja = X" );
```

setWindowPos (2.3)

Składnia:

setWindowPos(wnd, X, Y, W, H, option);

gdzie: wnd – symbol znakowy
X, Y, W, H, option – zmienne lub liczby typu *int*

Instrukcja *setWindowPos* pozwala na określenie pozycji standardowych okien pojawiających się w trakcie konsultacji z systemem ekspertowym oraz począwszy od wersji 2.3 do ustawiania okien typu arkusz kalkulacyjny lub wykres.

Pierwsza składnia zakłada, że parametr (*wnd*) określa typ okna, którego początkową pozycję chcemy ustawić. Może to być jeden z poniższych symboli:

ask	okno konsultacji;
how	okno wyjaśnień typu „jak?”;
metaphor	okno wyjaśnień typu *metafora*;
picture_ask	okno grafiki, wyświetlane w trakcie konsultacji;
picture_solution	okno grafiki, wyświetlane z oknem rozwiązań;
solution	okno rozwiązań;
video_ask	okno animacji wyświetlane w trakcie konsultacji;
video_solution	okno animacji wyświetlane z oknem rozwiązań;
what_is	okno wyjaśnień typu „co to jest?”;
why	okno wyjaśnień typu „dlaczego?”.

W przypadku ustawiania okien arkuszowych i wykresów pierwszy parametr to nazwa okna użyta m.in. w instrukcji **openSheet** lub **openChart**.

Kolejne parametery określają początkową pozycję górnego, lewego narożnika okna (*X*, *Y*) oraz szerokość (*W*) i wysokość (*H*) okna. Dane interpretowane są jako piksele (punkty ekranu).

Ostatni parametr (*option*) określa stan początkowy okna:

0 – okno w postaci normalnej;
1 – okno w postaci zmaksymalizowanej (pełny ekran);
2 – okno w postaci zminimalizowanej (ikona);
3 – przywrócenie okna do postaci normalnej.

Przykład:

```
setWindowPos( ask, 20, 20, 400, 300, 0 );
solve( źr1, "problem=X" );
```

sheetBindButton (4.0)

Składnia:

sheetBindButton(sheetName, button_name, fn_name);

gdzie: sheetName, button_name, fn_name – symbol znakowy

Instrukcja **sheetBindButton** umożliwia przypisanie do isntiejącego na arkuszu obiektu typu przycisk akcji zdefiniowanej w funkcji o nazwie *fn_name*. Aby przycisk był dostępny musi on posiadać zdefiniowaną w jego właściwościach nazwach poprzez którą przycisk jest identyfikowany (*button_name*).

showChart (2.3)

Składnia:

showChart(wykres, tryb);

gdzie: wykres – łańcuch znakowy lub zmienna typu *char*
tryb – zmienna lub liczba typu *int*

Instrukcja otwiera okno (widok) zawierające wykres zdefiniowany przez parametr *wykres*. Drugi z parametrów (*tryb*) określa tryb otwarcia: 1 – do edycji, 0 – bez możliwości edycji. Aby widok mógł zostać otwarty, należy wcześniej utworzyć wykres za pomocą instrukcji *openChart*. Instrukcja *showChart* może być wywoływana wielokrotnie (na przykład w celu otwarcia widoku wykresu zamkniętego przez użytkownika). Aby programowo zamknąć widok należy użyć instrukcji *closeWindow*. Również instrukcja *closeChart*, usuwająca wykres z pamięci, zamyka widok wykresu.

Zobacz: *openChart*, *closeChart*

showPicture (2.3)

Składnia:

showPicture(rysunek, plik, opcje);

gdzie: rysunek, plik – łańcuch tekstowy lub zmienna typu *char*
opcje – zmienna lub liczba typu *int*

Instrukcja tworzy okno o nazwie *Rysunek* i wyświetla w nim rysunek (mapę bitową) zapisaną w pliku *Plik*. Ostatni parametr określa sposób wyświetlania rysunku w oknie:
0 - rysunek dopasywany (rozciągany) jest do rozmiaru okna,
1 - rysunek centrowany jest w poziomie,
2 - rysunek centrowany jest w pionie,
3 - rysunek centrowany wewnątrz okna.

Przykład:

```
showPicture( "Rysunek", "rys1.bmp", 0 );
```

showSheet (2.2)

Składnia:

showSheet(arkusz, opcje);

gdzie: arkusz – łańcuch tekstowy lub zmienna typu *char*
opcje – zmienna lub liczba typu *int*

Instrukcja tworzy okno z „widokiem” arkusza, identyfikowanego przez argument *arkusz*, otwartego wcześniej za pomocą instrukcji *openSheet*. Możliwe jest tworzenie więcej niż jednego „widoku” jednego arkusza. Parametr *opcje* określa, czy na danym „widoku” można wykonywać operacje edycji arkusza (formatu, właściwości): *1* – edycja możliwa, *0* – brak możliwości edycji.

Zobacz: *closeSheet*, *getSheetRange*, *getSheetValue*, *openSheet*, *readSheet*, *writeSheet*, *setSheetRange*, *setSheetValue*

showVideo (2.3)

Składnia:

showVideo(Video, Plik, Opcje);

gdzie: *Video, Plik* - łańcuch lub zmienna typu **char**,
Opcje- zmienna numeryczna lub liczba.

Instrukcja tworzy okno o nazwie *Video* i wyświetla w nim animację wideo zapisaną w pliku *Plik*. Parametr *Opcje* określa czy po otwarciu okna należy uruchomić automatycznie animację (1) lub nie (0).

Przykład:

```
showVideo( "Animacja", "ostr4.avi", 0 );
```

showWindow (2.1)

Składnia:

showWindow(tytuł_okna, tryb);

gdzie: tytuł_okna – tytuł okna
tryb – kod określający tryb wyświetlania okna

Instrukcja służy do manipulacji sposobem wyświetlania okien, mających w swojej nazwie tekst określony przez parametr *tytuł_okna*.

Parametr *tryb* określa sposób wyświetlania okien:

0 – uaktywnij okno w bieżącej postaci;
1 – uaktywnij okno w trybie pełnoekranowym;
2 – zminimalizuj okno do ikony, uaktywnij okno PC-Shell'a;
3 – przywróć okno do zwykłej postaci;
4 – ukryj okno (aby przywrócić okno należy wykorzystać *tryb 0*).

Przykład:

```
// zminimalizuj okno menedżera plików
showWindow( "Menedżer plików", 2 );
```

Zobacz: *closeWindow*, *isAppRunning*

slistBox

Składnia:

slistBox(X, Y, tytuł, tekst, tekst_tabl, licznik, odp);

gdzie: X, Y – pozycja okna na ekranie
tytuł – tytuł okna
tekst – tytuł listy
tekst_tabl – tablica łańcuchów tekstowych
licznik – ilość elementów w tablicy *tekst_tabl*
odp – zmienna typu *int*

Instrukcja *slistBox* wyświetla listę łańcuchów tekstowych spośród których użytkownik może dokonać wyboru. Po naciśnięciu przycisku „OK” indeks wybranego znajdzie się w zmiennej *odp*.

Przykład:

```
char TEKST[ 4 ];
int RET;
TEKST[ 0 ] := "Drukarka";
TEKST[ 1 ] := "Monitor";
TEKST[ 2 ] := "Klawiatura";
TEKST[ 3 ] := "Stacja dysków";
slistBox( 0, 0, "Produkty","Wybierz produkt:", TEKST, 4, RET );
if ( RETURN == 1 )
begin
   messageBox( 0, 0, "Twój wybór: ", TEKST[ RET ] );
end;
```

solutionWin

Składnia:

solutionWin(X);

gdzie: X – symbol, łańcuch znakowy lub zmienna typu *char*

Instrukcja *solutionWin* umożliwia wyłączenie okna rozwiązania problemu. Dodatkowo wyłączany jest komunikat o potwierdzeniu postawionej hipotezy. Instrukcja ta może być stosowana wtedy, gdy inżynier wiedzy sam buduje interfejs użytkownika, w tym procedurę prezentacji wyników. Typowym zastosowaniem tej instrukcji jest aplikacja wykorzystująca architekturę tablicową. W takim przypadku instrukcja ta zapobiega zatrzymywaniu procesu wnioskowania przez okno rozwiązań po rozwiązaniu każdego problemu cząstkowego.

Parametr X może być jedną z dwóch wartości:
no – okno rozwiązań nie będzie wyświetlane;
yes – okno rozwiązań będzie wyświetlane.

Przykład:

```
solutionWin( yes );
```

solve

Składnia:

solve(S1, S2);

gdzie: S1 – symbol lub zmienna typu *char*
S2 – łańcuch znakowy lub zmienna typu *char*

Instrukcja *solve* należy do grupy instrukcji związanych z architekturą tablicową. Użycie tej instrukcji spowoduje następujący ciąg działań:

1. Uaktywnienie źródła wiedzy, wskazanego przez *S1*;
2. Rozpoczęcie wnioskowania wstecz (podobnie jak w przypadku instrukcji *goal*), z celem (hipotezą) określonym przez *S2*;
3. Zwolnienie uaktywnionego źródła wiedzy.

S1 jest nazwą źródła wiedzy, zadeklarowaną w bloku *sources*.
S2 zawiera tekst celu (hipotezy) podobnie jak w instrukcji *goal*.

Zgodnie z powyższym, instrukcja *solve* zastępuje ciąg instrukcji:

```
getSource( ... );
goal( ... );
freeSource( ... );
```

Instrukcja ta umożliwia zatem tworzenie bardziej oszczędnego (krótszego) kodu programu dla aplikacji tablicowych.

Przykład:

```
sources
   zr1:
       type kb
       file "zr1.zw";
   zr2:
       type kb
       file "zr2.zw";
end;

control
   // ...
   solve( zr1, "decyzja=DECYZJA" );
   solve( zr2, "diagnoza=DIAGNOZA" );
   // ...
end;
```

splitOAV

Składnia:

splitOAV(T, O, A, V);

gdzie: T – łańcuch znakowy lub zmienna typu *char*
O, A, V – zmienne typu *char*

Instrukcja *splitOAV* umożliwia rozbicie trójki OAW (obiekt-atrybut-wartość) na elementy składowe i przypisanie ich odpowiednio do parametrów (*O* – obiekt, *A* – atrybut, *V* – wartość). Z punktu widzenia semantyki tej instrukcji, można ją traktować jako „odwrotność" instrukcji *makeOAV*.

Właściwym kontekstem użycia instrukcji może być, między innymi, powiązanie jej z instrukcją *saveSolution*, w celu wyodrębnienia wymaganych wartości rozwiązania.

Przykład:

```
// Procedura rozbija rozwiązanie na elementy O, A, W
// i zapisuje je do pliku
char OB, ATR, WAR, ROZW[ 10 ];
float LE, I;
goal( "sytuacja_finansowa_firmy=SYT_FIN" );
saveSolution( ROZW, LE );
LE := LE - 1;
open( plik, w );
for I := 0 to LE step 1
begin
   splitOAV( ROZW[ I ], OB, ATR, WAR );
   fput( OB );
   fput( ATR );
   fput( WAR );
end;
close( plik );
```

sprintf (2.15)

Składnia:

sprintf(buffer, format, …);

gdzie: buffer – zmienna typu *char*
format – łańcuch znakowy lub zmienna typu *char*
pozostałe parametry dowolnego typu i w dowolnej ilości

Instrukcja powoduje utworzenie łańcucha znakowego w zmiennej *buffer* o formacie podanym w parametrze *format*. Jest ona analogiczna do funkcji standardowej języka C, z podanymi poniżej ograniczeniami. Parametry, począwszy od trzeciego, są alternatywne i mogą być dowolnego typu prostego (ilość ich jest nieograniczona, zależy jedynie od postaci łańcucha formatującego).

Dopuszczalne wzorce w łańcuchu formatującym to:

%d, *%i*	liczba dziesiętna typu *int*
%ld, *%li*	liczba dziesiętna typu *longint*
%u	liczba dziesiętna bez znaku typu *int*
%lu	liczba dziesiętna bez znaku typu *longint*
%x	liczba szesnastkowa typu *int* (małe litery)
%lx	liczba szesnastkowa typu *longint* (małe litery)
%X	liczba szesnastkowa typu *int* (duże litery)
%lX	liczba szesnastkowa typu *longint* (duże litery)
%f	liczba rzeczywista typu *float*
%lf	liczba rzeczywista typu *double*
%s	łańcuch znakowy
%c	pierwszy znak z łańcucha znakowego
%%	znak „%”

Jeżeli typ, parametr i wzorzec nie pasują do siebie (np. *%s* i zmienna typu *int*) w miejsce *%s* nie zostaną wstawione żadne dane. Również w przypadku zbyt małej ilość parametrów, zmiennym nie zostaną przypisane żadne dane.

Przykład:

```
int I;
long L;
float F;
double D;
char Dest, S;
I := 3;
L := 5;
F := 4.56;
D := 3.14;
S := "ma kota";
sprintf( Dest, "I=%d  L=%ld  F=%f  D=%lf", I, L, F, D );
// Dest = "I=3  L=5  F=4.56  D=3.14"
sprintf( Dest, "Ala %s i Adam %s", S, S );
// Dest = "Ala ma kota i Adam ma kota"
I := -3;
sprintf( Dest, "%% I=%i  I(bez znaku)=%u %%", I, I );
// Dest = "% I=-3  I(bez znaku)=65533 %"
```

sqlBind (2.1)

Składnia:

sqlBind(var);

gdzie: var – zmienna dowolnego typu prostego lub zmienna rekordowa

Instrukcja służy do utworzenia bufora pośredniczącego w wymianie danych pomiędzy bazą danych a systemem PC-Shell. Wymiana ta odbywa się w ten sposób, że przed wykonaniem zapytania zawierającego instrukcję *SELECT* należy utworzyć tzw. bufor transferowy, w którym każdej kolumnie wybranej za pomocą instrukcji *SELECT* odpowiada kolejna zmienna znajdująca się w buforze. Należy przy tym pamiętać o zgodności typów pomiędzy kolumnami a zmiennymi (w niektórych przypadkach system PC-Shell dokonuje automatycznej konwersji np. liczb z bazy danych do łańcucha znakowego itp.). Więcej informacji na temat dopuszczalnych konwersji można znaleźć w części „Dostęp do baz danych – wprowadzenie”.

Zobacz: *sqlFetch*, *sqlInitBinding*, *sqlQuery*

sqlDone (2.1)

Składnia:

sqlDone;

Instrukcja kończy współpracę (zwalnia wszystkie powiązania) systemu PC-Shell z bazami danych. Możliwe jest wielokrotne rozpoczynanie i kończenie pracy z systemami zarządzania bazami danych.

Zobacz: *sqlInit*

sqlFetch (2.1)

Składnia:

sqlFetch(result);

gdzie: result – zmienna typu *int*

Instrukcja powoduje pobranie kolejnych danych, uzyskanych w wyniku wykonania instrukcji *SELECT* z bazy danych do bufora transferowego. Poprawne wykonanie instrukcji powoduje ustawienie zmiennej *result* na *1*; brak lub wyczerpanie danych powoduje ustawienie wartości zmiennej *result* na *0*.

Zobacz: *sqlInitBinding*, *sqlBind*, *sqlQuery*

sqlInit (2.1)

Składnia:

sqlInit(S, X);

gdzie: S – łańcuch znakowy lub zmienna typu *char*
X – liczba lub zmienna typu *int*

Łańcuch *S* określa parametry połączenia z odpowiednim typem systemu zarządzającego bazami danych.

Format składni jest następujący:

```
DSN=data-source-name; UID[n]=userID; PWD[n]=password; DBQ=database-qualifier;
[UIDn=userID; PWDn=password…]
```

gdzie:

- *data-source-name* – nazwa bazy danych, określona w systemie zarządzania ODBC (np. *DSN=Pliki dBase*);
- *userID* – nazwa użytkownika korzystającego z systemu;
- *password* – hasło dostępu dla danego użytkownika (*NULL*, gdy hasło nie jest wymagane);
- *database-qualifier* – identyfikator bazy danych, najczęściej nazwa bazy danych. (parametr ma znaczenie tylko w przypadku systemów korzystających z nazw identyfikacyjnych).

Argument *X* określa, czy zarządzaniem transakcjami zajmuje się automatycznie system PC-Shell (gdy *X=0*), czy też blok sterowania, poprzez wywołania instrukcji *sqlTransact*. W trybie automatycznym system PC-Shell przyjmuje, że każde wywołanie instrukcji *sqlQuery* stanowi pojedynczą transakację.

Zobacz: *sqlDone*, *sqlTransact*

sqlInitBinding (2.1)

Składnia:

sqlInitBinding;

Instrukcja powoduje zainicjowanie bufora transferu danych z baz danych do systemu PC-Shell, a dokładniej – wyczyszczenie poprzedniego bufora, a następnie przygotowywuje do ewentualnego utworzenia kolejnego bufora (przy użyciu instrukcji *sqlBind*).

Zobacz: *sqlBind*, *sqlFetch*, *sqlQuery*

sqlQuery (2.1)

Składnia:

sqlQuery(query);

gdzie: query – łańcuch znakowy lub zmienna typu *char*

Instrukcja służy do wysyłania zapytania (w języku SQL) o treści zawartej w argumencie *query* do systemu zarządzania bazą danych (DBMS). Wynik zapytania typu *SELECT* należy pobrać za pomocą instrukcji *sqlFetch* (po uprzednim przygotowaniu bufora na dane). Przygotowanie bufora realizują instrukcje *sqlInitBinding* oraz *sqlBind*.

Przykład:

```
sqlQuery("SELECT * FROM PRAC");
sqlQuery("INSERT INTO PRAC (NAZW,IMIE) VALUES ('Nowak','Jan')");
```

Zobacz: *sqlBind*, *sqlFetch*

sqlTransact (2.1)

Składnia:

sqlTransact(type);

gdzie: type – symbol *commit* lub *rollback*

Instrukcja umożliwia programowe sterowanie transakcjami. W pierwszej kolejności należy zainicjować dostęp do baz danych przy użyciu instrukcji *sqlInit* z drugim argumentem o wartości *0*.

Wywołanie instrukcji z parametrem *commit* powoduje zatwierdzenie wszystkich wykonanych komend SQL (zatwierdzenie transakcji), natomiast wywołanie instrukcji z parametrem *rollback* powoduje anulowanie wszystkich poprzednich komend SQL. Każde wywołanie instrukcji *sqlTransact* wyznacza punkt zakończenia transakcji i rozpoczęcie nowej.

Jeśli dostęp do baz danych został zainicjowany jako dostęp automatyczny, każde wywołanie instrukcji *sqlQuery* traktowane jest jako pojedyncza transakcja zakończona poleceniem *commit*.

Zobacz: *sqlInit*, *sqlQuery*

ston

Składnia:

ston(S, N);

gdzie: S – symbol, łańcuch znakowy, lub zmienna typu *char*
N – zmienna typu numerycznego

Instrukcja *ston* umożliwia zamianę łańcucha znaków na odpowiadającą mu wartość liczbową.

Przykład:

```
char STR;
int N;
float F;
STR := "-100";
ston( "123.45" , F );  // F przyjmie wartość 123.45
ston( STR, N );        // N przyjmie wartość -100
```

strcat

Składnia:

strcat(S1, S2);

gdzie: S1 – zmienna typu *char*
S2 – łańcuch znaków lub zmienna typu *char*

Instrukcja umożliwia konkatenację (połączenie) dwóch łańcuchów znakowych poprzez dołączenie do końca łańcucha tekstowego zawartego w zmiennej *S1*, łańcucha znaków reprezentowanego przez parametr *S2*. Rezultat – połączone teksty – umieszczony zostanie w zmiennej *S1*.

Przykład:

```
// po wykonaniu poniższego fragmentu programu
// zmienna S1 przyjmie wartość "Sztuczna inteligencja"
char S1, S2;
S1 := "Sztuczna";
S2 := " inteligencja";
strcat( S1, S2 );
```

strChange (2.15)

Składnia:

strChange(str, substr_old, substr_new);

gdzie: string – zmienna typu *char*
substr_old – łańcuch lub zmienna typu *char*
substr_new – łańcuch znakowy (może być pusty) lub zmienna typu *char*

Instrukcja służy do wyszukania w łańcuchu zawartym w zmiennej *str* wszystkich wystąpień ciągu znaków *substr_old* i ich zamianie na ciąg *substr_new*. Ostatni argument (*substr_new*) może być łańcuchem pustym co spowoduje usunięcie z łańcucha *str* wszystkich wystąpień łańcucha *substr_old*.

Przykład:

```
S := "Ala ma kota. Ala ma psa.";
strChange( S, "Ala", "Adam" );
// po wykonaniu instrukcji strChange w zmiennej S znajdzie się
// następujący ciąg znaków: Adam ma kota. Adam ma psa.
S := "grzyb = \"pieczarka\"";  // grzyb = "pieczarka"
strChange( S, "\"", "\"\"" );  // podwojenie znaków cudzysłowu
// po wykonaniu instrukcji strChange w zmiennej S znajdzie się
// następujący ciąg znaków: grzyb = ""pieczarka""
// (może mieć to znaczenie w przypadku korzystania z mechanizmu
// DDE i przesyłania poleceń na przykład do programu MS-Word)
```

Zobacz: *strcat*

strLen (2.2)

Składnia:

strLen(łańcuch, długość);

gdzie: łańcuch – łańcuch znakowy lub zmienna typu *char*
długość – zmienna typu *int*

Instrukcja *strLen* przekazuje do zmiennej *długość* długość łańcucha znaków podanego jako parametr *łańcuch*.

Przykład:

```
int Dł;
char S;
S := "";
strLen( S, Dł );
// po wykonaniu powyższej instrukcji Dł będzie równe 0
S := "ala ma kota";
strLen( S, Dł );
// po wykonaniu powyższej instrukcji Dł będzie równe 11
```

system

Składnia:

system(X);

gdzie: X – łańcuch znaków lub zmienna typu *char*

Instrukcja *system* umożliwia uruchomienie innej aplikacji (programu) w środowisku Windows, bez opuszczania systemu PC-Shell. W ten sposób aplikacja systemu PC-Shell może powierzyć wykonanie pewnych specjalistycznych obliczeń innemu, („zewnętrznemu”) programowi.

Parametr *X* jest łańcuchem znaków, zawierającym nazwę programu, który należy uruchomić i, ewentualnie, dodatkowe argumenty, zależne od uruchamianego programu. Jeśli poza nazwą programu występują dodatkowe argumenty, powinny one być rozdzielone spacjami.

Instrukcja *system* zwraca wartość *RETURN* równą *1* w przypadku pomyślnego wykonania lub *0*, jeśli podczas próby uruchomienia programu wystąpił błąd.

Przykład:

```
// instrukcja uruchomi program "notepad.exe" (Notatnik)
// z argumentem w postaci nazwy pliku "plik.txt"
system( "notepad.exe plik.txt" );
```

testEOF

Składnia:

testEOF(X);

gdzie: X – zmienna typu *int*

Instrukcja *testEOF* umożliwia sprawdzenie, czy został ustawiony wskaźnik końca pliku. Jeśli tak, to wartość zmiennej *X* będzie różna od zera, w przeciwnym przypadku – będzie równa zero. Instrukcja w pewnych sytuacjach powinna występować po użyciu którejś z instrukcji czytania danych z pliku, zwłaszcza wtedy, gdy liczba danych w pliku nie jest znana „z góry” i czytanie odbywa się aż do napotkania znaku końca pliku.

toClipboard (2.15)

Składnia:

toClipboard(data);

gdzie: data – liczba, łańcuch lub dowolna zmienna prosta

Instrukcja powoduje wstawienie do „schowka” systemowego (ang. *clipboard*) danych, określonych jako argument *data*, w postaci tekstowej. Instrukcja może służyć do wymiany danych pomiędzy aplikacjami, jak również w celu chwilowego zapamiętania danych (w celu późniejszego ich odtworzenia należy skorzystać z instrukcji *fromClipboard*). Należy jednak zaznaczyć, że instrukcja *toClipboard* niszczy, w momencie wywołania, poprzednią zawartość „schowka”.

Przykład:

Wykonanie poniższego fragmentu programu spowoduje wstawienie do „schowka” tekstu wyjaśnień „jak?” dla atrybutu „grzyb" i przekazanie go, za pośrednictwem mechanizmu DDE, do edytora MS-Word.

```
int ID;
char HOW;
ddeConnect( ID, "WinWord", "System" );
ddeExecute( ID, "[FileNew]" );
saveExplan( HOW, _, "grzyb", _, -1 );
toClipboard( HOW );
ddeExecute( ID, "[EditPaste]" );  // wstaw dane ze "schowka"
```

Zobacz: *fromClipboard*

vignette

Składnia:

vignette(X1, X2, X3);

gdzie: X1, X2, X3 – łańcuch znaków lub zmienna typu *char*

Instrukcja *vignette* tworzy, w sposób niezwykle efektywny, winietę dla aplikacji użytkownika. Pojawienie się winiety powoduje zatrzymanie pracy programu zawartego w bloku sterowania. Wznowienie pracy programu nast.ąpi po naciśnięciu przycisku „OK”, umieszczonego w oknie winiety.

Winieta zawiera trzy pola, w których wyświetlane są, przekazane za pośrednictwem argumentów *X1*, *X2* oraz *X3*, informacje tekstowe.

Znaczenie poszczególnych parametrów jest następujące:

X1 – nazwa aplikacji (tekst jest wyróżniony wielkością czcionki i kolorem);

X2 – dowolny, obszerniejszy tekst, umieszczany poniżej nazwy aplikacji; zaleca się by tekst zawierał m.in. następujące informacje:

- rozwinięcie nazwy systemu;
- dane o twórcy lub producencie systemu;
- nazwę licencjobiorcy;

X3 – krótki tekst, umieszczany w dolnej części okna winiety, zawierający na przykład informacje o autorze aplikacji.

Przykład:

```
vignette( "KREDYT 2.1", "System do analizy wniosków kredytowych
\nProducent: Firma X\nLicencja: Bank Y", "(C) 1999, Firma X" );
// dwuznak '\n' umożliwia przeniesienie dalszej części
// tekstu do następnego wiersza
```

while

Składnia:

while (*warunek*) *intrukcja_złożona*

Instrukcja *while* służy do tworzenia pętli programowych. Wykonanie pętli *while* rozpoczyna się od sprawdzenia prawdziwości warunku. Jeśli wyrażenie *warunek* jest prawdziwe, wykonana zostaje instrukcja *intrukcja_złożona*, po czym następuje powrót do początku pętli (sprawdzenia warunku). W przeciwnym wypadku (wyrażenie *warunek* fałszywe), sterowanie zostaje przekazane do pierwszej instrukcji znajdującej się za instrukcją *while*.

Wyrażenie *warunek* złożone jest z dwóch wartości, rozdzielonych jednym z operatorów relacji („==”, „<”, „>”, „<=”, „>=” lub „<>”). Wartości mogą występować w formie stałych liczbowych, symboli, łańcuchów znakowych lub zmiennych dowolnego typu.

Dozwolone jest zagnieżdżanie instrukcji *while*, tj. umieszczanie instrukcji wewnątrz innej instrukcji *while* lub *for*.

Przykład:

```
// Program przypisuje kolejnym elementom (od 1 do 10)
// tablicy TAB liczby równe ich indeksom
float TAB[ 10 ];
I := 1
while ( I <= 10 )
begin
   TAB[ I ] := I;
   I := I + 1;
end;
```

winHelp (2.2)

Składnia:

winHelp(nazwa_pliku, rodzaj, dane);

gdzie: nazwa_pliku – łańcuch znakowy
rodzaj – wartość symboliczna (*index*, *context*, *key* lub *quit*)
dane – wartość numeryczna typu int lub łańcuch znakowy typu *char*

Instrukcja *winHelp* wyświetla, określony za pomocą parametru *nazwa_pliku*, plik pomocy (typu .*hlp*). W zależności od parametru *rodzaj* udostępniane są:

index – indeks (spis treści) pliku pomocy;
context – pomoc kontekstowa dotycząca tematu, identyfikowanego przez parametr *dane* (liczba lub łańcuch tekstowy);
key – pomoc dotycząca podanego słowa kluczowego (parametr *dane*);
quit – zakończenie pracy aplikacji z pomocą.

Bliższe informacje na temat sposobu tworzenia i wykorzystania plików pomocy znaleźć można w odrębnych opracowaniach książkowych oraz dokumentacji SDK dla systemu Windows.

Przykład:

```
// wyświetlenie spisu treści pliku "winfile.hlp"
winHelp( "winfile.hlp", index, 0 );
// wyświetlenie informacji na temat instrukcji "winhelp"
winHelp( "instr.hlp", key, "winhelp" );
```

writeChart (2.3)

Składnia:

writeChart(wykres, nazwa_pliku);

gdzie: wykres, nazwa_pliku – łańcuchy znakowe lub zmienne typu *char*

Instrukcja *writeChart* umożliwia zapamiętanie wykresu określonego przez parametr *wykres* w pliku o nazwie zawartej w zmiennej *nazwa_pliku*. Możliwe jest zapisanie wykresu w postaci pliku binarnego (format **.vtc*) lub w postaci graficznej w jednym z następujących formatów:

- *bmp* („mapa bitowa”);
- *gif* (skompresowany plik graficzny);
- *jpg* (skompresowany plik graficzny).

Pliki zapamiętane w jednym z powyższych formatów mogą być wykorzystane przez inne aplikacje.

W przypadku podania jako drugiego parametru (*nazwa_pliku*) łańcucha pustego (" ") system podejmie próbę zapisania wykresu pod ostatnio użytą nazwą (dotyczy tylko plików typu **.vtc*) – jeżeli nie została ona jeszcze określona, na ekranie pojawi się okno umożliwiające wprowadzenie nazwy pliku.

Przykład:

```
writeChart( Wykres1, "Wykr1.vtc" );
writeChart( Wykres1, "Wykr1.bmp" );
```

writeProfile (2.1)

Składnia:

writeProfile(sect, key, value, filename);

gdzie: sect, key – łańcuchy znakowe lub zmienne typu *char*
value – literał lub zmienna typu *int* lub *char*
filename – łańcuch znakowy lub zmienna typu *char*

Instrukcja *writeProfile* zapisuje do pliku konfiguracyjnego (typu *.ini*) o nazwie określonej parametrem *filename*, w sekcji *sect*, dla klucza *key*, wartość określoną w argumencie *value*. Dokładniejsze omówienie struktury pliku inicjalizacyjnego zamieszczono w części poświęconej instrukcji *getProfile*.

Przykład:

Poniższa instrukcja spowoduje umieszczenie w pliku „test.ini” w sekcji [System] wiersza „Time=12”

```
writeProfile( "System", "Time", "12", "test.ini" );
```

Zobacz: *getProfile*, *changeCategory*

writeSheet (2.2)

Składnia:

writeSheet(arkusz, nazwa_pliku);

gdzie: arkusz – łańcuch tekstowy lub zmienna typu *char*
nazwa_pliku – łańcuch tekstowy lub zmienna typu *char*

Instrukcja zapisuje w podanym pliku (*nazwa_pliku*) zawartość arkusza o nazwie określonej parametrem *arkusz*. Jeżeli wartość *nazwa_pliku* jest wartością pustą, arkusz zostanie zapamiętany pod ostatnio użytą nazwą. Plik zapisywany jest domyślnie w formacie FormulaOne (*.vts*). Aby zapisać plik w formacie MS-Excel nazwa pliku musi zawierać odpowiednie rozszerzenie (*.xls*).

Przykład:

```
writeSheet( "Bilans", "bilans.vts" );
writeSheet( "Płace", "place.xls" );
```

Zobacz: *closeSheet*, *getSheetRange*, *getSheetValue*, *openSheet*, *readSheet*, *setSheetRange*, *setSheetValue*, *showSheet*