ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1989

Seria: TRANSPORT z.9 Nr kol. 952

Roman KONIECZNY

CHARAKTERYSTYKA ZASOBÓW JĘZYKA LOGLAN NA POTRZEBY MODELOWANIA SYSTEMÓW TRANSPORTOWYCH

CZĘŚĆ I: Składnia, struktura, obiekty, współprogramy ...

Streszczenie. Niniejszy artykuł dokonuje prezentacji języka LOGLAN pod kątem jego wykorzystania - jako narzędzia do modelowania systemów transportowych. Część I zawiera opis składni języka, charakterystykę modularnej struktury programu, definicję obiektu jako podstawowej jednostki fazy wykonania programu. Przedstawione zostały również typy danych oraz wyrażenia, a także zasady przerywania wykonywania instrukcji, zasady komunikacji z otoczeniem (tj. mechanizmy tworzenia plików dyskowych oraz elementy grafiki komputerowej), możliwość tworzenia abstrakcyjnych typów danych, itd.

LOGLAN jest językiem obiektowo-zorientowanym, oferującym środki tworzenia i usuwania obiektów. Pojęcie "obiekt" można interpretować jako pewien fragment pamięci - służący do przechowywania wartości atrybutów. Program wykonywany w LOGLANie jest zbiorem współdziałających obiektów. Obiektami mogą być: procedury, funkcje, bloki, klasy, procesy, współprogramy, tablice dynamiczne oraz pliki. Dostęp do zasobów obiektu dokonywany jest poprzez zmienne referencyjne.

W artykule zwrócono szczególną uwagę na bezpieczeństwo programowania oraz na elementy programowania równoległego - z użyciem techniki współprogramów. Całość uzupełniają dwa przykłady programów eksponujących istotne cechy języka.

1. Uwagi wstępne

W niniejszym artykule dokonano prezentacji języka LOGLAN pod kątem jego wykorzystania - jako narzędzia do modelowania dużych systemów transportowych (jak np. ruch pociągów w złożonych sieciach kolejowych, ruch kołowy w dużych aglomeracjach miejskich, ruch lotniczy, morski, wewnątrzzakładowy, a także połączenia tych systemów).

Język LOGLAN opracowany został w drugiej połowie lat siedemdziesiątych (wersja LOGLAN-77), a następnie rozszerzony na początku lat osiemdziesiątych (wersja LOGLAN-82) przez grupę pracowników Instytutu Informatyki Uniwersytetu Warszawskiego, pracujących pod kierownictwem prof. Andrzeja Salwickiego. W roku 1986 przystąpiono do dalszych prac nad rozbudową tego języka.

Idea języka LOGLAN wywodzi się od takich języków, jak SIMULA-67 oraz (częściowo) ALGOL-68, jednakże - z praktycznego punktu widzenia - LOGLAN stanowi znaczny postęp w stosunku do tych języków. Uwzględniając fakt, że LOGLAN w ciągu ostatnich lat został zaprezentowany w wielu publikacjach

krajowych, jak również zagranicznych, w artykule niniejszym przedstawiono wyłącznie najistotniejsze cechy języka, interesujące nie tylko dla naukowca-programistę programów symulacyjnych. Wśród cech tych można wymienić m.in.: modularną strukturę, przejrzystą składnię, udogodnienia w instrukcjach iteracyjnych, typy formalne, klasy, tablice dynamiczne, prefiksowanie, współprogramy, procesy, obsługę sytuacji wyjątkowych, klasę SIMULATION (na potrzeby symulacji dyskretnej), klasę LOGTEXT (do operacji na tekstach) i inne klasy opracowane w ostatnim czasie.

2. Składnia

Język LOGLAN ma pascalo-podobną składnię, nie odbiegającą od przyzwyczajeń programistów. W celu zwiększenia przejrzystości zapisu oraz bezpieczeństwa programowania wprowadzono dodatkowe nawiasy syntaktyczne:

- w instrukcji warunkowej IF dodano nawias zamykający FI;

np.

```
IF  A>0  THEN CALL PROCEDURA1
         ELSE CALL PROCEDURA2  FI;
```

- w instrukcji pętli DO dodano OD;

np.

```
DO
   (ciąg instrukcji)
   (..........)
   OD;
```

- w instrukcji wyboru CASE dodano ESAC;

np.

```
CASE  WA
      WHEN L1: I1;
      WHEN L2: I2;
      ...
      WHEN Ln: In;
OTHERWISE  I;
ESAC,
```

gdzie: WA jest wyrażeniem typu INTEGER lub CHARACTER, L1..Ln są stałymi (lub listami stałych) tego samego typu, a I1..In oraz I są ciągami instrukcji.

Na dodatkowe podkreślenie zasługuje fakt, że LOGLAN nie posiada instrukcji GOTO. Brak tej instrukcji należy traktować jako zaletę tego języka.

3. Modularna struktura

Podstawowym pojęciem w języku jest pojęcie obiektu, ma ono jednak charakter dynamiczny i odnosi się do czasu wykonania programu; natomiast podstawową kategorią syntaktyczną jest moduł, będący tekstowym wzorem, według którego tworzone są obiekty.

W LOGLAN-ie rozróżnia się kilka rodzajów modułów: bloki, procedury, funkcje, klasy, współprogramy (coroutine) i procesy (process). Klasy pełnią szczególną rolę w języku. W pewnym uproszczeniu można klasę widzieć jako połączenie rekordu dynamicznego (jak w Pascalu) z procedurą. Klasy używane są zarówno do budowy struktur danych, jak też jako środek do modularyzacji i dla definiowania innych modułów.

Wszystkie moduły mają następującą strukturę:

```
<nagłówek modułu>
<lista deklaracji>
<lista handlerów>
BEGIN
<lista instrukcji>
 LAST WILL
 <awaryjna lista instrukcji>
END
```

Każda z powyższych list może być pusta. "Awaryjne" instrukcje po LAST-WILL wykonywane są jedynie w przypadku nienormalnego zakończenia pracy modułu, np. spowodowanego błędem.

Nagłówek modułu określa rodzaj modułu, w przypadku procedury, funkcji i klasy definiuje nazwę modułu, ewentualną listę parametrów, a dla funkcji dodatkowo typ wyniku. Dla bloku nagłówek redukuje się słowa BLOCK. Blok jest traktowany jak instrukcja będąca połączeniem generacji obiektu bloku z jego równoczesną deklaracją. Deklaracja rozpoczyna się od słowa kluczowego - odpowiednio VAR, CONST, SIGNAL lub UNIT, po którym następują specyfikacje deklarowanych atrybutów.

Możliwe jest zagnieżdżenie modułów - lista deklaracji modułu może zawierać deklaracje innych modułów: podobnie wystąpienie instrukcji bloku w module oznacza zagnieżdżenie tego bloku w otaczającym module.

Każdy moduł w LOGLAN-ie może zostać sparametryzowany. Istnieją trzy zasadnicze rodzaje parametrów: zmienne, procedury lub funkcje oraz typy. Specyfikacja parametrów rozpoczyna się od jednego ze słów: INPUT, OUTPUT, INOUT, PROCEDURE, FUNCTION lub TYPE. Brak specyfikacji rodzaju oznacza domyślnie parametr INPUT.

4. Typy i wyrażenia

Jako typ może wystąpić nazwa typu pierwotnego: INTEGER, REAL, BOOLEAN, CHARACTER (skr. CHAR), STRING, FILE, SEMAPHORE, nazwa klasy (procesu, współprogramu) lub nazwa parametru TYPE (tzw. typu formalnego). Zamiast nazwy klasy może wystąpić słowo kluczowe PROCESS lub COROUTINE

Typy klasowe (również COROUTINE i PROCESS), formalne i tablicowe (tzn. postaci ARRAYOF ...) nazywane są referencyjnymi, a wartości tych typów referencjami (nazwa "referencja" pochodzi z języka SIMULA-67, w terminologii PASCALa jest to wskaźnik). Wartości typów referencyjnych są jedynie wskaźnikami do obiektów generowanych podczas wykonania programu, samo zadeklarowanie zmiennej referencyjnej nie oznacza jeszcze wygenerowania obiektu.

W LOGLAN-ie typ STRING obejmuje tylko napisy występujące w tekście programu: nie jest możliwe tworzenie nowych wartości tego typu.

Typ SEMAPHORE ma zastosowanie do synchronizacji procesów współbieżnych.

Typ FILE oznacza plik zawierający wartości dowolnego typu.

Typ tablicowy postaci: ARRAYOF <typ elementu> oznacza jednowymiarową tablicę dynamiczną elementów wskazanego typu, który również może być typem tablicowym. Tablice wielowymiarowe uzyskuje się używając tablicy referencji do tablic, które znowu mogą zawierać wskaźniki do tablic.

Wyrażenia arytmetyczne i logiczne mają konwencjonalną składnię i są zbudowane ze stałych, zmiennych, elementów tablic i wywołań funkcji (może także wystąpić zdalny dostęp) za pomocą operatorów arytmetycznych, relacyjnych i logicznych oraz nawiasów.

5. Przerywanie sekwencyjnego wykonywania instrukcji

Składnia instrukcji w LOGLAN-ie zgodna jest z kanonami strukturalnego programowania. W języku nie istnieje pojęcie etykiety ani skoku; wprowadzono natomiast zostały dodatkowe instrukcje "wymuszające" programowanie strukturalne. Należą do nich EXIT, REPEAT, RETURN i RAISE.

Instrukcja EXIT umożliwia wyjście z pętli programowej; można ją traktować jak skok poza OD zamykające pętlę. W przypadku zagnieżdżonych pętli - możliwe jest opuszczenie nie tylko wewnętrznej pętli, podwójne EXIT (tzn. EXIT EXIT) powoduje opuszczenie dwóch otaczających pętli, potrójne EXIT trzech, itd. Jeżeli liczba EXIT przekracza głębokość zagnieżdżenia pętli, to wykonanie takiej instrukcji powoduje przejście do END modułu.

Użycie instrukcji REPEAT powoduje przerwanie bieżącej iteracji i rozpoczęcie następnej; można więc traktować REPEAT jak skok przed OD zamyka-

jącej pętlę. Instrukcja REPEAT może także wystąpić w połączeniu z EXIT. Przykładowo: EXIT EXIT REPEAT powoduje opuszczenie dwóch otaczających pętli i rozpoczęcie kolejnej iteracji w trzeciej pętli.

Instrukcja RETURN w procedurze lub funkcji powoduje zakończenie wykonywania modułu i powrót do miejsca jego wywołania. (Jeżeli RETURN nie występuje explicite, to powrót wykonywany jest po natrafieniu na końcowy symbol END danego modułu).

Instrukcja RAISE służy do wysłania sygnału.

6. Komunikacja z otoczeniem

Do komunikacji z otoczeniem służą: podstawowe operacje wejścia-wyjścia, operacje na plikach oraz instrukcje grafiki ekranu. (W punkcie tym omówiono komunikacje z otoczeniem dla wersji LOGLAN-u zaimplementowanej dla mikrokomputera IBM PC).

Do grupy podstawowych operacji wejścia-wyjścia należą konwencjonalne: READ(..), READLN(..), EOLN - funkcja badająca, czy wystąpił koniec linii, WRITE(..), WRITELN(..), INKEY - funkcja zwracająca kod dziesiętny ostatnio naciśniętego znaku klawiatury.

Do komunikacji z plikiem (obiektami typu FILE) służą dodatkowo: OPEN (f,T) dla pliku typu INTERNAL oraz OPEN(f,T,<nazwa>) dla pliku typu EXTERNAL, gdzie: f - oznacza zmienną plikową, a T określa rodzaj pliku (T = TEXT dla plików tekstowych, T = (CHAR, INTERGER, REAL) dla plików binarnych zawierających znaki ASCII lub liczby całkowite czy liczby rzeczywiste; nazwa jest stałą łańcuchową określającą nazwę pliku. Procedury RESET, REWRITE oraz funkcja EOF działają na ogólnie znanych zasadach. Do komunikacji z plikami binarnymi służą instrukcje PUT(..) i GET(..).

Do grafiki ekranowej (dla karty Hercules) służą następujące procedury:

GRON(0) - załącza tryb graficzny (czyszcząc jednocześnie ekran),
HPAGE(..) - załącza pierwszą lub drugą stronę karty Hercules,
MOVE(X,Y) - ustawia bieżącą pozycję punktu (pixel) na ekranie,
DRAW(M,N) - rysuje linię od pozycji bieżącej do pozycji M,N,
HASCII(kod_znaku) - umożliwia pisanie tekstów na ekranie graficznym: kod znaku określa wartość dziesiętną kodu znaku ASCII,
GROFF - wyłącza tryb graficzny.

Funkcje całkowite INXPOS(0) i INYPOS(0) zwracają odpowiednio współrzędne X i Y bieżącej pozycji pixla na ekranie.

7. Typy formalne

Parametry TYPE (typy formalne) pozwalają zdefiniować algorytm lub strukturę danych działające na wartościach pewnego abstrakcyjnego, bliżej nie znanego typu.

Nazwa typu formalnego może być użyta w specyfikacjach parametrów i w lokalnych deklaracjach. Specyfikacja parametru TYPE musi poprzedzać specyfikacje innych parametrów, używające tego typu. Parametrami aktualnymi dla parametrów TYPE mogą być tylko typy referencyjne, a więc typy klas, współprogramów, procesów, tablic lub inne typy formalne.

Bogate możliwości stwarza równoczesne sparametryzowanie modułu typem (lub typami) formalnym i procedurami lub funkcjami działającymi na argumentach tego typu. Pozwala to na definiowanie abstrakcyjnych struktur danych i wyrażanie algorytmu w terminach pewnych abstrakcyjnych pojęć i operacji na nich.

8. Obiekty

1) Pojęcie obiektu

LOGLAN jest językiem obiektowo-zorientowanym, oferującym środki tworzenia i usuwania obiektów, przy jednoczesnym zapewnieniu bezpieczeństwa ich użycia.

Pojęcie obiekt można interpretować jako pewien fragment (pole) pamięci - służący do przechowywania wartości atrybutów, a także instrukcji obiektu. Program wykonywany w LOGLAN-ie jest zbiorem współdziałających obiektów. Dominującą rolę odgrywają obiekty klas, tworzone przez operator NEW i usuwane instrukcją dealokacji KILL. Każdy obiekt istnieje od chwili generacji aż do chwili usunięcia; próba odwołania do atrybutów usuniętego obiektu powoduje błąd.

Obiekty procedur, funkcji i bloków są nietrwałe i po powrocie z obiektu (tzn. po wykonaniu instrukcji RETURN lub dojściu do END i odczytaniu następnie wartości parametrów OUTPUT i INOUT) są automatycznie usuwane.

Obiekty klas, tablice dynamiczne i pliki są obiektami trwałymi i pozostają dostępne od chwili utworzenia aż do ich usunięcia przez użycie explicite instrukcji KILL. Obiekt staje się niedostępny również wtedy, gdy nie jest wskazywany przez żadną zmienną referencyjną. Bezpieczeństwo użycia polega na tym, że usunięty obiekt staje się naprawdę niedostępny i wszelkie próby dostępu do niego zostają wykryte i sygnalizowane (standardowy sygnał). Jest to istotna różnica w porównaniu z językiem ADA lub PASCAL, w których nic nie chroni programisty przed dostępem do miejsca pamięci zajmowanego przez usunięty obiekt, gdy po jego usunięciu istnieją jeszcze zmienne wskazujące na ten obiekt.

Przykład: po wykonaniu instrukcji X:=NEW POJAZD; Y:=X; X:=NONE; nadal istnieje obiekt (POJAZD) wygenerowany przez NEW i jest on wskazywany przez Y, zmieniła się jedynie wartość zmiennej referencyjnej X; natomiast po wykonaniu ciągu instrukcji: X:= NEW POJAZD; Y:= X; Z:= X; KILL(Y); utworzony obiekt klasy POJAZD został usunięty i wszystkie trzy zmienne referencyjne X, Y, Z mają wartość NONE. Instrukcja przypisana dla zmiennych referencyjnych (np. powyżej Y:=X;) oznacza, że zmienna Y wskazuje to samo pole pamięci, co zmienna X (referencyjna do tego samego obiektu). Jeżeli Y miałoby wskazywać na identyczny ale inny "egzemplarz" obiektu klasy POJAZD, należałoby użyć instrukcji COPY: np. X:= NEW POJAZD; ...; Y:= COPY(X); wówczas X i Y wskazują na dwa różne obiekty o identycznych wartościach atrybutów.

Poniżej podano przykład programu ilustrującego użycie instrukcji COPY:

```
PROGRAM TESTKOB;
(* przykład kopiowania wartości atrybutów obiektu *)
   UNIT POJAZD: CLASS(NUMER_POJAZDU:     INTEGER,  (* hipotetyczny pojazd *)
                      PALIWO_STARTOWE:   REAL,
                      CZAS_JAZDY:        REAL     );

   CONST REZERWA=10.5,
         PLANOWE_POSTOJE=3.3,
         WSP1=0.6,
         WSP2=10;

   VAR   STAN_ZBIORNIKA:  REAL,
         EMISJA_SPALIN:   REAL,
         SUMA_CZASU:      REAL;

   BEGIN
         SUMA_CZASU:=CZAS_JAZDY+PLANOWE_POSTOJE;
         STAN_ZBIORNIKA:=PALIWO_STARTOWE+REZERWA-CZAS_JAZDY*WSP1;
         EMISJA_SPALIN:=WSP2*CZAS_JAZDY+WSP1*PLANOWE_POSTOJE;

   END; (* POJAZD *)

UNIT INFORMACJA:  PROCEDURE(P: POJAZD);

   BEGIN
      WRITELN;
      WRITELN("          Pojazd numer     ",   P.NUMER_POJAZDU);
      WRITELN("          Suma czasu:      ",   P.SUMA_CZASU);
      WRITELN("          Stan zbiornika:  ",   P.STAN_ZBIORNIKA);
      WRITELN("          Emisja spalin:   ",   P.EMISJA_SPALIN);

   END  ( * INFORMACJA *)

   VAR P1, P2:  POJAZD;
```

```
BEGIN (* program glowny *)
    P1=NEW POJAZD(1,50,10);          (* wygenerowanie pojazdu nr 1 *)
    CALL INFORMACJA(P1);
    P1.STAN ZBIORNIKA:=P1.STAN ZBIORNIKA+15;  (* uzupełnienie paliwa *)
    CALL INFORMACJA(P1);
    P2:=COPY(P1);                    (* utworzenie "kopii" pojazdu P1*)
    CALL INFORMACJA (P2);
END (* TESTKOB *)
```

Wykonanie programu TESTKOB jest następujące:

```
Pojazd numer      1
Suma czasu      13.3000
Stan zbiornika: 54.5000
Emisja spalin: 101.9800

Pojazd numer      1
Suma czasu:     13.3000
Stan zbiornika: 69.5000
Emisja spalin: 101.9800

Pojazd numer      1
Suma czasu:     13.3000
Stan zbiornika: 69.5000
Emisja spalin: 101.9800
```

Instrukcję COPY w symulacji systemów transportowych można wykorzystać do odnotowywania wartości atrybutów wybranych obiektów w określonych stanach tych obiektów lub w wybranych sytuacjach ruchowych. Instrukcja ta ułatwia również operowanie na danych zawartych w obiektach tablicowych.

2) Tablice dynamiczne

Obiekt tablicy dynamicznej o specyfikacji ARRAYOF T składa się z pewnej liczby jednakowych, indeksowanych liczbami całkowitymi, zmiennych typu T. Zakres indeksu, a tym samym rozmiar tablicy (liczba elementów), jest ustalany dynamicznie w chwili generacji obiektu.

Do utworzenia obiektu tablicowego służy instrukcja ARRAY Z DIM (a : b), gdzie a i b są wyrażeniami typu INTEGER oraz zachodzi a<=b, a Z jest zmienną typu ARRAYOF T. Instrukcja ta generuje tablicę o zakresie indeksu od a do b i wstawia na Z referencję do tej tablicy.

Operatory LOWER(Z) i UPPER(Z) zwracają (odpowiednio) dolny i górny zakres indeksu tablicy wskazanej przez Z (jeśli Z=NONE, to próba ich użycia spowoduje błąd).

Obiekt tablicowy jest tablicą jednowymiarową. Tablice wielowymiarowe uzyskuje się - używając tablic referencji do tablic, przy czym rozmiary (graniczne wartości indeksów) tych tablic ustalane są niezależnie.

Przykład: Konwencjonalna tablica prostokątna o wymiarach M na N uzyskiwana jest w LOGLAN-ie w następujący sposób:

```
VAR A: ARRAYOF ARRAYOF INTEGER;  (* deklaracja tablicy A zawierającej
                                    liczby całkowite *)
..... (* fragment programu *) .....
ARRAY A DIM (1 : N);     (* generacja wektora referencji do N wierszy *)
FOR  I:=1  TO  M
   DO  (* generacja kolejnych wierszy o rozmiarze  1 : N *)
      ARRAY A(I) DIM (1 : N);
   OD;
```

W przypadku generowania macierzy trójkątnej dolnej postać instrukcji FOR jest następująca:

```
FOR  I:= 1  TO  N
   DO  (* generacja kolejnych wierszy o rozmiarze  1 : I *)
      ARRAY  A(I)  DIM (1 : I);
   OD;
```

W każdej chwili może zostać zmieniony kształt powyższej struktury na nieregularny (np. po wykonaniu instrukcji: A(1):=COPY A(M); A(2):=COPY A(M-1); A(5):=COPY A(3); ... itd.).

W języku LOGLAN można w łatwy sposób generować tablice o różnych kształtach: trójkątne, pasmowe, przestrzenne schodkowe itp. Podstawowym zyskiem (oprócz oszczędności pamięci) jest pełna kontrola zakresów indeksów.

W przypadku tablic wielowymiarowych można składowymi tablicami operować niezależnie (używając mniej indeksów), np. tak:

```
VAR  TABLICA_JEDNOWYMIAROWA : ARRAYOF                    REAL,
     TABLICA_DWUWYMIAROWA   : ARRAYOF ARRAYOF            REAL,
     TABLICA_TRÓJWYMIAROWA  : ARRAYOF ARRAYOF ARRAYOF    REAL;
......
     TABLICA_JEDNOWYMIAROWA(M), TABLICA_DWUWYMIAROWA(M,N) :=1;
     TABLICA_TRÓJWYMIAROWA(M,N,Q) :=2.73;
     TABLICA_DWUWYMIAROWA(1)  := TABLICA_JEDNOWYMIAROWA;
     TABLICA_TRÓJWYMIAROWA(2) := TABLICA_DWUWYMIAROWA;
     TABLICA_TRÓJWYMIAROWA(1,1):= TABLICA_JEDNOWYMIAROWA;
......
```

3) Obiekty procedur, funkcji i bloków

Obiekty procedur, funkcji i bloków są tworzone w chwili wywołania procedury lub funkcji albo wykonania instrukcji bloku i następnie, po powrocie z procedury, funkcji lub bloku są usuwane.

Procedury i funkcje mogą być wywoływane rekurencyjnie.

4) Obiekty klas

Obiekty klas są tworzone przez użycie generatora postaci: NEW nazwa klasy (lista parametrów aktualnych).

Wartością generatora jest referencja do nowo utworzonego obiektu. Generator może występować w wyrażeniach lub samodzielnie, np. X := NEW A; NEW C1(7);...

W odróżnieniu od pozostałych modułów, atrybuty klasy mogą być również używane na zewnątrz tej klasy poprzez zdalny dostęp, np. X.ART(tzw. dostęp kropkowy), gdzie X jest wyrażeniem referencyjnym wskazującym obiekt, z którego ma pochodzić atrybut ATR. Wartość X musi być różna od NONE, ATR musi być nazwą zmiennej, procedury, funkcji lub klasy.

W języku LOGLAN, podobnie jak w SIMULI-67, występuje operator THIS. Wartością wyrażenia THIS A (gdzie A jest nazwą klasy) jest referencja do najbliższego otoczającego obiektu klasy A.

W LOGLAN-ie, w odróżnieniu od SIMULI-67, zastosowano mechanizmy ochrony, których przeznaczeniem jest zabezpieczenie atrybutów klasy przed niewłaściwym użyciem (wybrane atrybuty klasy mogą zostać zastrzeżone jako "prywatne" - tzn., że zabronione jest używanie ich na zewnątrz klasy). Możliwe jest zatem tworzenie klas jako pewnych zamkniętych struktur (czarnych skrzynek) jako produktów użytkowych - bez możliwości ingerowania w ich wnętrze. Zastrzeżenie ma postać: CLOSE lista atrybutów i może wystąpić wśród deklaracji (zabrania ono zdalnego dostępu do wymienionych atrybutów).

Zastosowanie klas może być szerokie i różnorodne. Najprostszym zastosowaniem klasy - jest użycie jej (obiektu) jako rekordu dynamicznego, przy czym użycie parametrów ułatwia nadawanie wartości początkowych. Klasa może definiować nie tylko strukturę danych (rekord), ale i operacje na tej strukturze. Klasy można użyć jako pakietu definiującego struktury danych i operacje na nich: za pomocą klas można również budować języki problemowo-zorientowane, systemy hierarchiczne itd.

Przykład zdalnego dostępu do atrybutów obiektu zawiera pokazany wcześniej program TESTKOB.

9. Współprogramy

Współprogramy są silnym narzędziem programistycznym ułatwiającym programowanie w języku sekwencyjnym. Komunikacja między współprogramami polega głównie na przekazywaniu sterowania. Współprogramy są mechanizmem sekwencyjnym - oznacza to, że w danym momencie wykonywania programu może być aktywny tylko jeden obiekt współprogramu.

Współprogramy mogą znaleźć zastosowanie przy modelowaniu w sekwencyjnym języku programowania zjawisk przebiegających w sposób równoległy; w symulacji, a także do eleganckiego zapisu pewnych algorytmów, w których w sposób naturalny można wyodrębnić fazy - wzajemnie przeplatające się.

Z punktu widzenia użytkownika współprogram może być uznany za rozszerzenie klasy, podlegające na wyposażeniu go w mechanizmy przekazywania sterowania.

Obiekt współprogramu może znajdować się w jednym ze stanów:

- zainicjalizowany,
- aktywny,
- zawieszony,
- zakończony.

Natychmiast po utworzeniu obiekt współprogramu znajduje się w stanie "zainicjalizowany". Pozostaje w tym stanie, dopóki sterowanie nie osiągnie instrukcji RETURN. Wykonanie instrukcji RETURN powoduje zawieszenie obiektu danego współprogramu oraz przekazanie sterowania z powrotem do instancji, która go utworzyła. Zawieszony obiekt może być wznowiony instrukcją ATTACH. Przez cały czas oczekiwania pamiętane jest miejsce, od którego wykonanie ma być wznowione. Instrukcja DETACH zwraca sterowanie do obiektu współprogramu, który ostatnio aktywował (używając instrukcji ATTACH) dany obiekt. Gdy lista instrukcji współprogramu zostanie wyczerpana (sterowanie osiągnie END kończący ciało współprogramu), obiekt współprogramu staje się "zakończony". Zakończony obiekt współprogramu jest niedostępny dla operacji przekazywania sterowania. Pozostaje on jedynie jako zbiór lokalnych danych, do których można odwołać się z zewnątrz za pomocą zdalnego dostępu. (Można wówczas utożsamiać zakończony obiekt współprogramu z utworzonym obiektem klasy. Podobnie jak obiekt klasy, zakończony obiekt współprogramu można "zabić" używając instrukcji KILL).

Współprogram może mieć budowę złożoną, tzn. zawierać w sobie inne współprogramy, klasy, funkcje lub procedury. Deklaracja współprogramu ma następującą postać:

```
UNIT nazwa_współprogramu : prefix COROUTINE (parametry formalne);
 <lokalne deklaracje>
BEGIN
 <lista instrukcji>
END;
```

Deklaracja współprogramu różni się od deklaracji klasy tylko tym, że słowo kluczowe CLASS jest zastępione słowem kluczowym COROUTINE.

Instrukcja RETURN może się znajdować w dowolnym miejscu ciągu instrukcji. Mogą ją poprzedzać np. inicjalizujące lokalne dane współprogramu; może wystąpić wewnątrz pętli, może też wystąpić wielokrotnie (istotne jest jednak tylko pierwsze napotkane w trakcie wykonania wystąpienie, wszystkie następne są równoważne instrukcji pustej).

Deklaracja współprogramu służy jako wzorzec, według którego tworzone są jego instancje. Możliwe jest utworzenie wielu instancji tego samego współprogramu. Instancję (obiekt) tworzy się podobnie jak obiekt klasy przy użyciu instrukcji <u>generatora</u>:

NEW nazwa_współprogramu (parametry aktualne).

Przy przekazywaniu sterowania instrukcja ATTACH musi być użyta z parametrem aktualnym, np. ATTACH(X) - gdzie X jest wyrażeniem, którego wartością jest referencja do obiektu współprogramu. Blok główny programu również jest uważany za obiekt współprogramu. Jest on dostępny pod nazwą MAIN. Zatem ATTACH(MAIN) oznacza przekazanie sterowania do bloku głównego programu.

Poniżej podano prosty przykład przekazywania sterowania między współprogramami:

```
PROGRAM PROSTY;
(* prosty przykład przekazywania sterowania między współprogramami *)
VAR P,P1: PIERWSZY, P2: DRUGI;
UNIT PIERWSZY: COROUTINE(PAR: INTEGER);
  BEGIN
     RETURN;
     ATTACH(P2);
     DETACH;
     ATTACH(MAIN);
     DETACH;
  END; (* PIERWSZY *)
UNIT DRUGI: COROUTINE;
  BEGIN
     RETURN;
     ATTACH(P);
     DETACH;
     ATTACH(MAIN);
     ATTACH(P1);
     DETACH;
  END; (* DRUGI *)
BEGIN (* ===  MAIN  === *)
  P  :=NEW  PIERWSZY(1);
  P2:=NEW   DRUGI;
  P1:=NEW   PIERWSZY(2);
  ATTACH(P);
  ATTACH(P2);
  DETACH;
  ATTACH(P1);
  DETACH;
END  (* PROSTY *).
```

Recenzent: Doc. dr hab. inż. Krzysztof Chwesiuk

Wpłynęło do Redakcji 19.11.1987 r.

ХАРАКТЕРИСТИКА ВОЗМОЖНОСТЕЙ ЯЗЫКА ЛОГЛАН
ДЛЯ ПОТРЕБНОСТЕЙ МОДЕЛИРОВАНИЯ
ТРАНСПОРТНЫХ СИСТЕМ
ЧАСТЬ I: СИНТАКСИС, СТРУКТУРА, ОБЪЕКТЫ, КОПРОГРАММЫ

Р е з ю м е

В настоящей статье представляется язык программирования ЛОГЛАН с точки зрения его использования как инструмента моделирования транспортных систем. Часть I содержит описание синтаксиса языка, характеристики модульной структуры программы и определение объекта как основной единицы цикла выполнения программы. Представлены типы данных и выражения а также принципы останова выполнения инструкций, принципы связи с внешней средой (т.е. механизмы создавания файлов и элементов компьютерной графики), возможность создавния абстракционных типов данных и.тп. ЛОГЛАН является объектно- ориентированным языком, дающим возможность создавания и удаления объектов. Понятие "объект" можно интерпритировать как некий фрагмент памяти для сохранения величины аттрибутов. Программа написанная на языке ЛОГЛАН является множеством содействующих объектов. Объектами могут быть процедуры, функции, блоки, классы, процессы, программы, динамические таблицы и файлы. Достуя к ресурсам объекта происходит через референционные переменные.

В статье обращено особое внимание на безопасность программирования а также на элементы параллельные программированию с использаванием техники копрограмм. Приведены два примера программ иллюстрирующие существенные свойства языка.

CHARACTERISTIC OF THE LOGLAN LANGUAGE RESERVES
FOR NEEDS OF MODELLING THE TRANSPORT SYSTEMS
PART I: Syntax, structure, objects, coroutines

S u m m a r y

The present article describes the LOGLAN language from the view - point of its using as an instrument for modelling the transport system. Part I comprises description of the language syntax, characteristic of program modular structure, definition of an object as a fundamental unit of program realization phase. Types of data and expressions as well as principles of interrupting the instruction's performing, principles of communication with operator, (i.e. mechanisms of forming disk files and elements of computer graphics) possibility of forming abstract types of data, etc. have been presented too.

LOGLAN in the object - oriented language offering means of the objects' creation and elimination. The notion: "object" may be interpretted as a certain fragment of memory which serves storing the attribute values. Program performed in LOGLAN is a set of cooperating objects. The objects can be: procedures, functions, blocks, classes, processes, coroutines, dynamic arrays and files. An access to the object's reserves is executed through reference variables.

A special attention has been paid in the article to the safety of programming and to the elements of parallel programming - with the use of coroutines' technique. The whole is supplemented by two examples of programs exposing essential features of the language.