ZSI.Techniki Programowania

Wykład 8

Janusz Jabłonowski

Moduły w Pascalu

Często zdarza się, że tworzymy zestaw procedur (funkcji, stałych, typów i zmiennych) realizujących powiązane ze sobą operacje (zwykle dotyczące jednego typu danych, np. operacje na wektorach). Ponieważ taki zestaw można potem wykorzystać w wielu programach, byłoby wygodnie nie musieć kopiować go tekstowo za każdym razem, gdy chcemy go użyć, lecz móc tyko wskazać, że dany zestaw ma być teraz użyty. (Podobnie jak z procedurą, raz ją definiujemy, a potem wielokrotnie używamy podając tylko jej nazwę). Taki zestaw nazywamy modułem. Zwykle moduł zawiera deklaracje potrzebne lokalnie (na potrzeby modułu) oraz deklaracje eksportowane, czyli takie, które będzie widać poza modułem.

Moduły można bardzo łatwo tworzyć w Pascalu (precyzyjniej: w różnych implementacjach Pascala, jak np. Free Pascal, Delphi, Turbo Pascal). Moduł zapisujemy w pliku z rozszerzeniem .pas. W samym module trzeba dopisać kilka specjalnych klauzul i informację o eksportowanych deklaracjach (opisane poniżej). Po skompilowaniu otrzymujemy plik o takiej nazwie jak plik .pas ale o innym rozszerzeniu (.ppw we Free Pascalu, .dcu w Delphi, .tpu w Turbo Pascalu). Ten plik zawiera skompilowane treści procedur i funkcji modułu (oraz informacje o innych deklaracjach z modułu). Jeśli ktoś chce skorzystać z tego modułu, to potrzebuje TYLKO pliku skompilowanego, plik źródłowy (.pas) nie jest już potrzebny!

- Struktura modułu

Moduł ma następującą strukturę:

unit <nazwa>;

interface

<lista eksportowanych deklaracji>

implementation

<lista deklaracji lokalnych i definicje eksportowanych procedur>

begin
<inicjalizacja>
end.

Uwagi:
1) <nazwa> musi być taka jak nazwa pliku.
2) W <liście eksportowanych deklaracji> podajemy same nagłówki eksportowanych procedur, ich treści (wraz z powtórzonymi nagłówkami) muszą się znajdować w części implementation.

3) <inicjalizacja> wykonuje się tylko raz, przed rozpoczęciem wykonywania głównego programu.
4) Jeśli inicjalizacja jest zbędna, to można ją pominąć wraz ze słowem **begin**.
5) Zamiast słowa kluczowego **begin** można użyć słowa **initialization**. W takim przypadku można też umieścić w module część rozpoczynającą się słowem **finalization**, opisującą operacje, które mają się wykonać, podczas kończenia pracy programu.

- Korzystanie z modułów

Korzystanie z modułów jest bardzo proste: na początku korzystającego z nich programu (lub modułu) umieszczamy klauzulę
uses <nazwa>;
i od tej pory wszystkie deklaracje z importowanego modułu są widoczne w programie tak, jakby były w nim zadeklarowane.

- Struktura programu korzystającego z modułów

(rysunek)

- Zalety stosowania modułów
  - lepsza strukturalizacja programu,
  - ukrywanie implementacji,
  - szybsza kompilacja.

Kolejki i stosy

W bardzo wielu algorytmach pojawia się potrzeba wykorzystania struktury danych, umożliwiającej wstawianie i pobieranie danych. Dwiema najprostszymi taki strukturami są stos i kolejka.

- Stos

Umożliwia wstawianie elementów i ich pobieranie z początku (wierzchołka) stosu. Czasami mówi się, że stos realizuje strategię wstawiania/pobierania LIFO (Last In, First Out; ostatni wszedł, pierwszy wyjdzie). Stos odwraca kolejność wstawianych elementów.

Operacje:

```
procedure wstaw(var s: TStosu; elt: TDanych);
procedure pobierz(var s: TStosu; var elt: TDanych);
function pusty(s: TStosu): boolean;
-----------------------------------------------
procedure pierwszy(s: TStosu; var elt: TDanych);
procedure ini(var s: TStosu);
procedure usuń_wszystko(var s: TStosu);
```

Uwagi implementacyjne:

- co to jest TDanych,
- jak reprezentować stos, czyli co to jest TStosu (tablica, lista),
- czemu pierwszy i pobierz nie są funkcjami,
- czy parametr funkcji pusty powinien być przekazywany przez zmienną, czy przez wartość,

- czasami definiuje się trochę inne operacje (pobierz tylko usuwa element ze struktury danych),
- często spotykane nazwy angielskie (wstaw - push; pobierz - pop; pusta - empty; pierwszy - top),
- czym się różni ini od usuń_wszystko,
- czy warto wprowadzać usuń_wszystko.

- Kolejka

Umożliwia wstawianie nowych elementów na koniec i pobieranie elementów z początku. Czasami mówi się, że kolejka realizuje strategię wstawiania/pobierania FIFO (Firsty In, First Out; pierwszy wszedł, pierwszy wyjdzie). Kolejka zachowuje kolejność wstawianych elementów.

Operacje:

```
procedure wstaw(var k: TKolejki; elt: TDanych);

procedure pobierz(var k: TKolejki; var elt: TDanych);

function pusta(k: TKolejki): boolean;

-----------------------------------------------

procedure pierwszy(k: TKolejki; var elt: TDanych);

procedure ini(var k: TKolejki);

procedure usuń_wszystko(var k: TKolejki);
```

Uwagi implementacyjne:

- co to jest TDanych,
- jak reprezentować kolejkę, czyli co to jest TKolejki (tablica, lista),
- czemu pierwszy i pobierz nie są funkcjami,
- czy parametr funkcji pusta powinien być przekazywany przez zmienną, czy przez wartość,
- czasami definiuje się trochę inne operacje (pobierz tylko usuwa element ze struktury danych),
- często spotykane nazwy angielskie (wstaw - enqueue, put; pobierz - dequeue, get; pusta - empty; pierwszy - first),
- czym się różni ini od usuń_wszystko,
- czy warto wprowadzać usuń_wszystko.

- Przykłady zastosowań:
  Algorytmy grafowe (stos, kolejka), realizacja rekursji (stos), ONP i wyliczanie wartości wyrażeń (stos).

- Realizacja stosu i kolejki za pomocą modułów
  - podział na część publiczną i prywatną
  - decyzja czy operacje mają mieć strukturę danych (stos, kolejkę) jako pierwszy parametr

Struktury listowe

Typy wskaźnikowe są niezwykle elastyczne i pozwalają budować praktycznie dowolnie wymyślne struktury danych. Oprócz pojedynczych list można definiować tablice list, listy tablic, listy list itp.

Przykłady zastosowań bardziej rozbudowanych struktur listowych:
- rzadkie macierze (skrypt)
- sortowanie topologiczne (skrypt)

Dużo informacji o strukturach listowych można znaleźć w skrypcie (dostępnym w naszej bibliotece) "Pracownia Programowania I" pod redakcją M. Cichego i S. Szpakowicza.