Лекция 12.

Структурни абстракции на данните: Декартово произведение (структура)

Как в програмите могат да се групират свързани данни?

АТД структура: АМ

Структурата (запис, декартово произведение, *structure*, *record*, *cartesian product*) е съставен тип, представляващ ***подредена крайна*** съвкупност от елементи, които могат да се ***различават по своя тип***. До компонентите е осигурен ***директен достъп***

- **Основни понятия и характеристики:**
  - Компонентите се наричат **полета** или **членове** (*fields*, *members*) на структурата
  - Записът е един от малкото **комбинирани АТД** (т.е. чиито компоненти от своя страна могат да са от различни АТД)
  - В различните ЕП структурата се реализира или като **статичен** тип, или като **динамичен**, но в определени граници

АТД структура: АМ – приложение

- За моделирането на обекти, които имат много на брой характеристики (и за обръщение към този обект само с едно име) – служи за групиране на смислово свързани данни, които в общия случай представляват характеристики на едно понятие

  Пример: Понятието правилен многоъгълник се описва чрез две характеристики – брой на страните и дължина на страната

- За дефиниране на други структури от данни

  Пример: В банките клиентите се описват чрез няколко признака (например име, адрес, номер на сметка, ЕГН), някои от които на свой ред могат съдържат също няколко признака (например адресът се състои от улица, номер, град, окръг и пощенски код). Такива множества от данни могат да бъдат моделирани чрез записите.

Защо структурата се нарича декартово произведение

- В математиката **декартово произведение** на n множества $A_1, A_2, \ldots, A_n$ **е множеството, чиито елементи са подредени n- торки** $(a_1, a_2, \ldots, a_n)$**, където** a_k **принадлежи на** A_k

  Декартовото произведение се означава с $A_1 \times A_2 \times \ldots \times A_n$

Пр. Нека A={1,2}; B={a,b,c}. Декартовото произведение AxB на тези множества е множеството

A x B ={(1,a), (1,b), (1,c), (2,a), (2,b), (2,c)}

- АТД запис в ЕП предоставя възможност за дефиниране на декартови произведения за множества от данни

  Пример: Правилният многоъгълник от горния пример ще бъде елемент от декартовото произведение `integer × real` (брой на страните и дължина на страната)

- **Примери** за представяне на декартови произведения в езиците за програмиране са **структурите (записите)** в C, C++, Algol 68 и PL/I и **записите** в COBOL, Pascal и Ada.

АТД структура: Достъп и конструиране

- **Достъп**:

  Директният достъп до компонентите се осъществява чрез съпоставените им уникални **имена,** а не чрез позицията им (както е при масивите)

- **Конструиране** – стойността на структурата може да се конструира по два начина**:**
  - **чрез последователно изброяване** на стойностите на всички полета на записа, като се ограждат в някакъв вид *“скоби”*
  - **чрез достъп до всички полета** на записа за задаване на стойностите им

Структурите в езиците за програмиране

- **Типични характеристики** в ЕП от високо ниво:
  - записите са стандартен съставен тип за много от езиците но не всички (в Java, PHP не съществуват структури)
  - стойностите им се разглеждат като съставени от няколко именувани полета

- **Дефиниране** – декартовото произведение на `INTEGER` × `REAL` (Modula-3 или Pascal) :

```
TYPE Polygon =
    RECORD
            NoOfSides : INTEGER;
            SizeOfSides : REAL
    END;
VAR poly1, bigsquare : Polygon;
```

- **Достъп** – Достъпът до поле на запис се осъществява чрез т.нар. “точково” означаване (*dot notation*):

```
circumference := poly1.NoOfSides * poly1.SizeOfSides;
```

- **Конструиране** – едновременно (Modula-3):

```
bigsquare := Polygon{NoOfSides := 4; SizeOfSides := 195.6};
```

АТД структура в C++: ФМ и ЗМ

- **Физически модел**

  В паметта на компютъра всяка величина от тип структура (запис) заема толкова място, колкото е необходимо, за да бъдат съхранени всички нейни компоненти

- **Знаков модел**

  В C++ структурите служат не само като модел на декартово произведение на различни множества от стойности, те всъщност са специален вид класове. Все пак, за да се прави разлика тук ще бъде разгледано само първото им предназначение

- **Синтаксис** за деклариране на тип структура и променливи от този тип:

  `struct` *<име>*$_{незад}$ **{** *<декл. на полета>*$_{незад}$**}** *<декл. на променливи>*$_{незад}$**;**

  , където:

  – *<име>* е името на типа-структура

  – *<декл. на полета>* е списък от типове и имена на променливи, разделени с “;”, които представляват **полетата** на структурата (**членове-данни**). Полетата могат да бъдат от всеки тип освен `void`, непълен тип (затова не могат да бъдат от типа на структурата, която се декларира в момента, но могат да бъдат указател към нея) или функция

  – *<декл. на променливи>* е списък от имена на променливи от този тип структура. Препоръчва се променливите от даден тип структура да се декларират отделно, като се спазва следния синтаксис:

  `struct` $_{незад}$ *<име на тип структура> <декл. на променливи>***;**

АТД структура в C++: ЗМ – примери за деклариране

(1)

```
struct Point {
  int x;
  int y;
};
Point p1,p2;
```

(2)

```
struct Child {
  double height;
  double weight;
  int years;
  int months;
  char gender;
} Ivan, Maria;
```

(3)

```
typedef  struct Rectangle {
  int left, top;
  int width, height;
} r1, r2, r3;
```

(4)

```
struct Date {      //дата
     int Day;
     int Mon;
     int Year;
};
enum  Merka{Kg, M, L, Br} ;
struct Stoka {     //стока
     char Ime[30];
     Merka Mer;
     double EdCena;
     Date GodnaDo; //вложен
                   //запис
} S;
```

(5)

```
struct movies {
  string title;
  int year;
};

movies amovie;
movies * pmovie; // указател
```

АТД структура в C++: ЗМ – множество от стойности

Множеството от стойности на структурата е**:**

Съвкупността от всички възможни комбинации от стойности на неговите полета. Стойността на всяка променлива от даден тип структура съдържа всички декларирани полета

АТД структура в C++: Операция селектор

Изразът за достъп до членове (*member-selection expression*) на структура има две форми:

- Универсална – чрез “точково” означаване:
  - за променливи-структури или псевдоними на структури

```
p1.x = 150; p1.y = 50;
Point po1, &pps=po1;
pps.x = 150;
```

```
Stoka S;
cout << S.GodnaDo.Day;
```

  - за достъп до структури, към които сочи указател

```
Point *pp;
//…
(*pp).x = 150;
(*pp).y = 50;
```

```
Stoka *pS=new Stoka;
//…
cout << (*pS).GodnaDo.Day;
```

- Само за достъп до структури, към които сочи указател – чрез “->”:

```
int a = pp->x; //еквивалентно на a = (*pp).x;
cout << pS->GodnaDo.Day;
```

АТД структура в C++: ЗМ – конструктор и инициализиране

1. **Поелементно** (чрез обхождане)

```
Date toD; Stoka S;

//конструира toD
toD.Day=30; toD.Mon=6; toD.Year=2002;

//конструира S
strcpy(S.Ime,"Ivan");
S.Mer=Kg; S.EdCena=1.99; S.GodnaDo=toD;
```

2. При **инициализиране** :

```
Date Den={1,1,2003};
```

АТД структура в C++: Операции и релации

- **Въвеждане и извеждане**: структурите могат да се въвеждат и извеждат само поелементно (ако >>/<< не са предефинирани за съответния тип)
- **Присвояване** – на променлива от тип структура може да бъде присвоена променлива от същия тип структура

```
Point p1, p2;
p1.x=3;
p1.y=2;
p2=p1; /*копира полетата на p1 в p2*/
```

, което е еквивалентно на:

```
p2.x=p1.x;
p2.y=p1.y;
```

- **Всички останали операции** със структури се извършват поелементно (могат да се предефинират от програмиста)
- **Релации** със структури не са дефинирани стандартно (програмистът може сам да предефинира избрана релация за някой тип структура)
- Структурите могат да се подават като **параметри на функции** и да бъдат техен **резултат**

АТД структура в C++: Особености

- Когато не е необходимо името на типа структура може да се пропусне (анонимна структура, *anonymous*) ! Зависи от компилатора и се позволява за някои версии на езика:

(1) когато всички необходими променливи се декларират на едно място

```
struct {float x, y;} complex;
```

(2) при вложени структури

```
struct somestruct{
  struct // Анонимна
    { int x, y; } point;
  int type;
} w;
```

```
struct phone{
        int areacode;
        long number;
};
struct person{
   char name[30];
   struct phone; // Анонимна
} Jim;
Jim.number = 1234567;
```

- MS C++ позволява последното поле на структура да бъде масив с недефиниран размер, което дава възможност към структурата да се добавят низове с различна дължина или друг масив

```
struct PERSON {
   unsigned number;
   char     name[];     //масив с недефиниран размер};
```

Примерна програма за работа с структури

Условие: Да се напише програма, която позволява да се въвеждат от клавиатурата данни за две книги и да се извеждат въведените данни

Проектиране:

(1) Представяне на данните:

Входни данни: Данните за всяка книга удобно се представят чрез комбинирания тип данни структура

(2) Алгортъм:

Извеждането на данните за една книга може да се обособи в отделна функция

Програмна реализация на примера

Програмата използва шаблонния клас `string` от заглавния файл `<cstring>` от STL:

```
#include <iostream>
#include <cstring>
using namespace std;
struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};
int main( ) {
   struct Books Book1;        // деклариране на Book1 от тип Book
   struct Books Book2;        // деклариране на Book2 от тип Book

   // въвеждане на данни за Book 1
   strcpy( Book1.title, "Fundamentals of Programming C++");
   strcpy( Book1.author, „Richard Halterman");
   strcpy( Book1.subject, "C++ Programming");
   Book1.book_id = 345407;
```

```
   // въвеждане на данни за Book 2
   strcpy( Book2.title, "Data Structures and Algorithm in C++");
   strcpy( Book2.author, "Clifford Shaffer");
   strcpy( Book2.subject, " C++ Programming");
   Book2.book_id = 6495700;
    // извеждане на данните за  Book1
   printBook( Book1 );
   // извеждане на данните за Book2
   printBook( Book2 );
   return 0;
}
void printBook( struct Books book ) {
   cout << "Book title : " << book.title <<endl;
   cout << "Book author : " << book.author <<endl;
   cout << "Book subject : " << book.subject <<endl;
   cout << "Book id : " << book.book_id <<endl;
}
```