Увод в програмирането

Лекция 12:

Структури

Допълнителен материал

за **Информатика и ИС**

Проблем (1)

- Функция, която изважда два момента от време (часове, минути, секунди) и връща разликата отново в часове, минути и секунди:
- ```
  void subtract(int h1, int m1, int s1,
                int h2, int m2, int s2,
                int &h3, int &m3, int &s3) {
      int t = h2 * 3600 + m2 * 60 + s2;
      t -= h1 * 3600 + m1 * 60 + s1;
      if (t < 0) t += 24 * 60 * 60;
      h3 = t / 3600;
      m3 = t / 60 % 60;
      s3 = t % 60;
  }
  ```
- Списъкът с параметри е потресаващ
  ```

Проблем (2)

- Не можем ли да си дефинираме нов тип данни, примерно Time?
- Тогава функцията би изглеждала доста по-човешки:
- `Time subtract(Time start, Time end)`

Структура / запис (1)

- Можем:
- ```
  struct Time
  {
      int hours;
      int minutes;
      int seconds;
  };
  ```
- Така дефинирахме нов тип данни!
  ```

Структура / запис (2)

- Съставен тип данни
- Представя крайна редица от елементи
- Редицата е с фиксирана дължина
- Елементите могат да са от различни типове
- Пряк достъп до всеки елемент по име
  - (поле на записа / член-данна на записа)
- Физическо представяне: елементите се записват последователно в паметта
- Прилики и разлики с масивите

Синтаксис

- struct <име> { <поле> { <поле> } };
- <поле> ::= <тип> <идентификатор>
  { , <идентификатор> };
- В една дефиниция не може да има две полета с еднакъв идентификатор
- Може да се дефинират както извън функциите, така и в тях (в произволен блок)
  - От това зависи видимостта им

Синтаксис – още примери

- Комплексно число:

```
struct Complex {
    double re, im;
};
```

- Можем да си дефинираме и доста по-различни типове данни:

```
struct Student {
  char name[50];
  int fn;
  double grade;
};
```

Променливи от тип запис

- Променливи се дефинират по познатия ни начин - <тип> <име>
- **Time moment;**
  Complex z1, z2;
  Student george;

Инициализация

- Инициализация – синтаксис
- ```
  Complex z1 = {0.0, -1.0}, z2;
  Student s = {"Ivan Ivanov", 12345, 6.0};
  ```
- Виждали сме го при масивите
  ```

Операции

- Присвояване на нова стойност:

```
Time t1 = {1, 30, 0}; // инициализация
Time t2;
t2 = t1; // присвояване
```

- Вход и изход със cin и cout не е възможен:

~~`cin >> t1; cout << t1;`~~

- Достъп до поле – с оператор “.”

<променлива>.<поле>

```
t1.seconds = 0;
cin >> t1.hours >> t1.minutes;
cout << t1.hours << ':' << t1.minutes;
int *ptr = &t1.hours;
```

Масиви от записи

- Примерно приложение: представяне на таблици
- И тук няма нищо, което да не ни е познато:
- ```
  Student group[10] = {
      {"Ivan Ivanov", 12345, 6.0},
      {"Petar Petrov", 12399, 5.5}
  };
  cout << group[0].name << endl;
  for (int i = 0; i < 10; i++)
      cin >> group[i].grade;
  ```
  ```

- Типовете на полетата могат също да бъдат структури:
- ```
  struct TimeInterval
  {
      Time start, end;
  };
  ```
- ```
  TimeInterval exam = {{9, 15, 0}, {12, 0, 0}};
  cout << "The exam starts at " <<
      exam.start.hours << ':' <<
      exam.start.minutes << endl;
  ```

Структури и функции

- Подаване на запис като параметър на функция
  - `… f(Time time) { … }`
- Връщане на запис от функция
  - `Time f(…) { … }`
- Предават се и се връщат по стойност
  - Т.е. прави се копие на стойността, както при int
  - Ако искаме функцията да може да промени подадената ѝ стойност – указател или псевдоним:
    `… f(Time &time) { time.hours = …; … }`

Задачи

- Да се напише програма, която:
  - Въвежда масив от студенти
  - Извежда списък на скъсаните студенти
  - Намира средния успех на всички студенти
  - Подрежда студентите по факултетен номер

Указатели към записи

- ```
  Student s = {"Ivan Ivanov", 12345, 6.0};
  Student *p = &s, *q = NULL;
  *p = s;
  (*p).fn = 71717;
  ```
- При достъпване на поле чрез указател съществува и съкратен запис:
  ```
  p->fn = 71717;
  ```
- Псевдоними:
  ```
  Student &r = s;
  r.fn = 54321;
  cout << s.fn << endl; // 54321
  ```
  ```

Рекурсивни структури

- Възможно е поле да бъде от същия тип като дефинираната структура:

```
struct Employee {
   char name[60];
//Employee boss; // грешно!
//Employee &boss; // грешно!
   Employee *boss; // вярно
};
```

- Задължително трябва да използваме указател
  - В първия грешен случай – безкрайна рекурсия

```
struct Message
{
    char recipient[64];
    char text[256];
};
Message m;
strcpy(m.recipient, "всички студенти");
strcpy(m.text, "Имате ли въпроси?");
cout << "До " << m.recipient << ":\n";
cout << m.text << endl;
```