ТЕМА: Типи даних програміста: структури, перелічення, оператор typedef.

[Крім названих, є ще бітові поля (bit-field) і об'єднання (union) – ми не розглядаємо]

Структури: оголошення типу

Структура – це сукупність змінних, об'єднаних спільним іменем. Такі змінні, як правило, логічно зв'язані між собою. Наприклад, структура визначення адреси особи:

```
struct addr  // оголошення структури - шаблон
{
  char name[30];  // прізвище
  char city[25];  // місто
  char street[50];  // вулиця, будинок, квартира
  unsigned long int zip;  // поштовий індекс
  unsigned int family;  // кількість членів сім'ї
};  // крапка з комою в кінці
```

Приклад показує простішу форму оголошення структури. Оголошення структури визначають ключовим словом `struct`. Ціла структура називається `addr`. `addr` – це ім'я типу, але ще не сама величина. Оголошення визначає шаблон, який можна буде пізніше використати для створення конкретних величин. Тобто оголошення структури – це оголошення власного типу даних за іменем `addr`, але не самих даних.

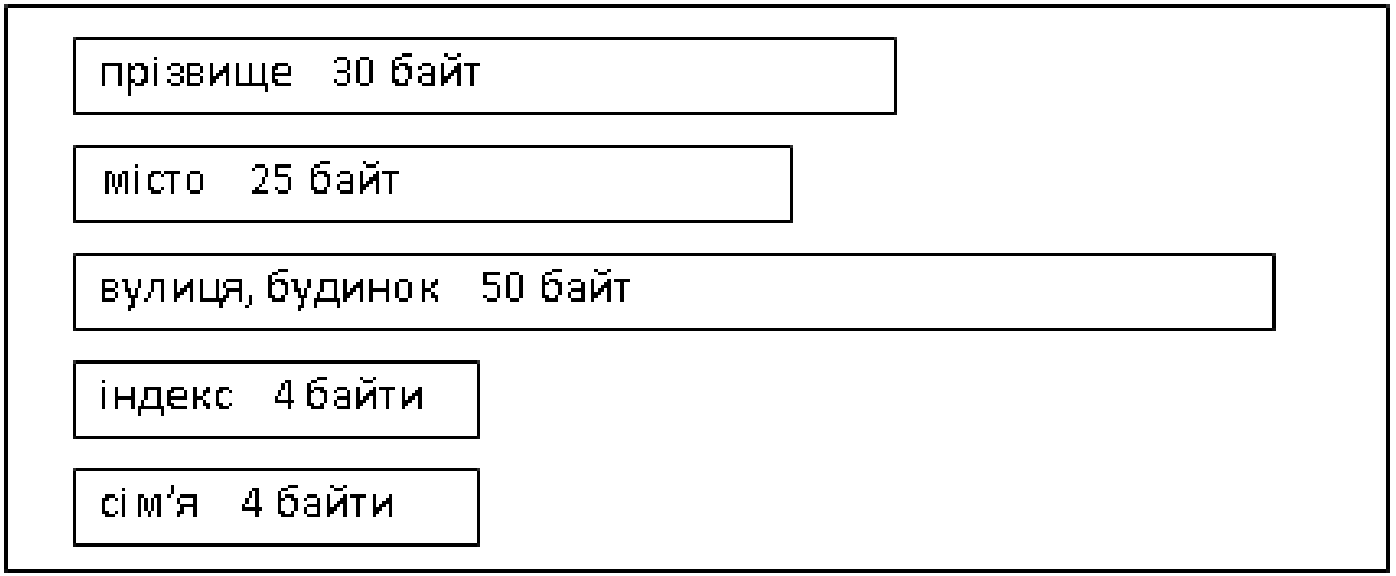
Величини, які входять в структуру, називають термінами: поля; елементи; члени. Кожне поле оголошують так само, як окремі величини за правилами С++.

З точки зору С++ оголошення структури є оператором, тому в кінці є крапка з комою.

Щоб створити величини типу `addr`, потрібно записати оператор, наприклад:

```
addr Taras, Olha;  //  створити дві величини типу addr
```

Після оголошення величин, які є структурами, компілятор автоматично надає пам’ять для полів структури:



Зауважимо, що таку пам’ять отримає кожна величини типу `addr` окремо і незалежно.

Ініціалізація структурних величин

Подібно до інших типів даних, структурні величини можна ініціалізувати в момент оголошення, записуючи перелік значень кожного поля:

```
addr Taras = { "Taras M.I.", "Stryj", "Franka, 26/9",29062,4 } ;
// перевірити отримані значення
cout << Taras.name << '\t' << Taras.city << endl;
cout << Taras.street << '\t' << Taras.zip << '\t'
     << Taras.family << endl;
```

Тип кожного значення повинний відповідати типу поля в порядку визначення. Якщо відповідності немає, наприклад:

```
addr Taras = { "Taras M.I.", 1234, "Franka,26/9", 29062, "++++" } ;
```

компілятор пробує автоматично перетворити записані значення до потрібного типу, при цьому, як правило, отримаємо неправильні результати.

Ініціалізація може бути неповною:

```
addr Taras = { "Taras M.I.", "Stryj" } ;
             // неповна, решта - нульові значення
```

В цьому разі невизначені поля отримують нульові значення чи порожні рядки, і ініціалізація в цілому є коректною.

Доступ до полів структури

Доступ до окремих полів структури забезпечує оператор «.» (крапка). За іменем структурної величини записують знак крапки та ім’я поля:

```
strcpy(Olha.name, "Olha N.A.");
strcpy(Olha.city, "Drohobych");
Olha.family = 4;
```

Кожне поле має свій тип і з кожним полем окремо можна виконувати всі операції, допустимі для відповідного типу. Ніяких операцій над цілою структурою, як єдиним цілим, проводити не можна. Можна лише присвоїти одну структурну величину іншій:

```
addr student;  student = Taras;  // присвоєння структурних величин
```

В цьому разі відбувається копіювання значень всіх полів зразу (присвоєння).

```
// після цього поміняти значення окремого поля
strcat(student.city, " - Lviv");
```

Масиви структур

Часто структури використовують як елементи масивів. Щоб оголосити масив структур, спочатку треба визначити саму структуру. Після цього оголосити масив величин цього типу:

```
addr groupP11[25];  // масив структур з 25 елементів
```

Доступ до конкретного елемента, тобто, до конкретної структури, виконують через індекс, як в будь-якому масиві:

```
groupP11[0] = Taras;  groupP11[1] = Olha;
strcpy(groupP11[2].name, "Stefa");
groupP11[2].zip = 29016;
```

Вказівники на структури

Будувати вказівники на структури можна за тими самими загальними правилами, що й вказівники на інший тип даних. Це можуть бути посилання на існуючу структуру і динамічне створення структури.

Посилання на існуючу структуру. Використовують операцію обчислення адреси & і операцію звертання за адресою. Проте у випадку структур операція звертання за адресою може мати два способи: «операція стрілка» або «операція розіменування *»:

```
addr *formcaptain = &Olha;  // староста групи
strcat(formcaptain->name, " starosta");
formcaptain->zip = 10021;  // "стрілка"
// або операція розіменування (звернути увагу на дужки)
// strcat( (*formcaptain).name, " starosta" );
// (*formcaptain).zip = 10021;
cout << formcaptain->name << '\t' << formcaptain->zip << endl;
```

На практиці звертання за адресою частіше позначають стрілкою.

Динамічне створення структури. Початково оголошують лише вказівник, структури немає. Застосовують оператори динамічного розподілу пам’яті `new` і `delete`, які, відповідно, надають і звільняють пам’ять в часі виконання програми. Операції з полями динамічної структури виконують через стрілку або розіменування:

```
addr *restore = new addr;
        // динам.створення - наприклад, поновлення навчання
strcpy(restore->name, "Nazar");  (*restore).zip = 90021;
        // способи доступу
cout << (*restore).name << '\t' << restore->zip << endl<< endl;
// інші операції з динамічною структурою . . . . .
delete restore;  // в кінці треба звільнити пам'ять
```

Вкладені структури

Поле деякої структури може бути іншою структурою. Внутрішня структура називається вкладеною. До внутрішнього структурного поля можна застосувати своєю чергою нове вкладення. Глубина вкладень не обмежується. На практиці вкладення застосовують для будови потрібної системи даних. Наприклад, будова реєстру документів може бути такою:

```
// звернути увагу на порядок оголошень структур
//   - від внутрішніх до зовнішніх

struct ttime  // час в годинах і хвилинах
{
  int hour;  // година
  int minute;  // хвилини
};

struct date  // дата
{
  int year;  // рік NNNN
  int month;  // місяць 1-12
  int day;  // день 1-31
  ttime tt;  // час - оголошено раніше
};
```

```
struct telephone  // телефон
{
  unsigned long int mobile;  // мобільний
  unsigned long int stationary;  // стаціонарний
};
struct person  // персональні дані
{
  char name[65];  // прізвище, ім'я
  char addres[100];  // адреса проживання
  telephone pho;  // телефони
};
struct document  // структура документу
{
  char content[200];  // зміст документу
  person who;  // від кого
  date dt;  // час поступлення
};

document DeansOffice[125]; // реєстр документів на 125 екземплярів

strcpy(DeansOffice[0].content, "Прошу збільшити стипендію");
strcpy(DeansOffice[0].who.name, "Степаненко Р.");
DeansOffice[0].who.pho.mobile = 22033044;
DeansOffice[0].dt.month = 4;  DeansOffice[0].dt.day = 23;
DeansOffice[0].dt.tt.hour = 15;
```

Перелічення

Перелічення – це набір іменованих цілочисельних констант. Тобто, деяку порівняно невелику групу цілих чисел, кожне з яких відображає певний зміст, можна позначити змістовими ідентифікаторами.

Перелічення в основній формі визначають так:

```
enum тип { список_констант } ;
```

Наприклад, програма опрацьовує карту маршрутів руху, і потрібно визначати географічний напрям руху:

```
enum direction { north, east, south, west } ;
```

Або для опрацювання музикальних нотних записів треба мати позначення нот:

```
enum note { doo, re, mi, fa, sol, la, si } ;
```

Отже, кожну константу позначають ідентифікатором, який має бути унікальним і не може бути використаний в іншому контексті. Кожен ідентифікатор списку означає ціле число. Найперша константа списку дорівнює 0. Значення кожної константи списку на 1 більше від попередньої. Наприклад, для першого прикладу це означає, що `north=0`, `east=1`, `south=2`, `west=3`.

Подібні позначення можна записувати в тексті програми замість чисел, що значно покращує наглядність програми.

Зробимо зразу порівняння. Якщо в програмі записати, наприклад

```
const int st = 2;  // st – це комірка пам’яті
```

то це визначає змінну-константу, яка має в пам’яті свою комірку, хоча й визначає таке саме число 2. А перелічення – це лише символьне позначення константи, наприклад:

```
int summer = south;  // south – це позначення, не комірка
```

Значення констант в переліченні можна явно задавати з допомогою ініціалізатора. Для цього після константи записують знак «дорівнює» і ціле число, причому як додатнє, так і від’ємне. Наприклад:

```
enum  bits { one=1, two=2, four=4, eight=8 } ;  // 1, 2, 4, 8
enum decimalsteps { ds1=10,ds2=100,ds3=1000,ds4=10000,ds5=100000 } ;
enum  gtone { ax, bx=0, cx, dx=1 } ;  // 0, 0, 1, 1
```

Не забувати, що явно не ініціалізовані константи на 1 більші від попередньої.

Іменовані константи, визначені в переліченнях, можна використати у всіх операціях, де допускають цілі операнди.

Оператор typedef

З допомогою ключового слова `typedef` можна визначити нове ім'я типу даних, - але не сам тип. Новий тип при цьому не створюється, лише вже існуючий тип отримує нове ім'я (синонім). Загальний вигляд оператора `typedef`:

```
typedef тип нове_ім'я
```

Такий оператор використовують в основному для двох цілей:

1) надати типу інше ім'я для покращення наглядності програми:

```
typedef  float  temperature;
temperature  d_12_04, d_13_04;
typedef  unsigned char  byte;
byte  letter, cipher;
```

2) позначити тип коротким іменем:

```
typedef  double **  dynmatrix;  // динамічна матриця
typedef  double *  row;  // рядок динамічної матриці
typedef  double  element;  // тип елемента матриці
dynmatrix  m = new row[10];
for (int i=0; i<10; i++)  m[i] = new element[15];  // і т.д.
```

Приклад задачі

[Звернути увагу на особливості програмування структур і типів]

На площині в певні моменти часу з'являються кольорові точки. Кожна має свої координати, колір і час появи. Параметри точок записані в масиві. Надрукувати:

1) перелік точок, які мають жовтий колір;

2) точку, найвіддаленішу від початку координат.

Алгоритм. Простий обхід всіх точок в масиві параметрів.

```
struct coordinates { float x,y; } ;  // координати точки

enum  spectrum { red, orange, yellow, green, blue, violet,
                 indigo, ultraviolet } ; // кольори

char colorname[][15] = {   // текстові назви кольорів
   "red", "orange", "yellow", "green", "blue", "violet",
   "indigo", "ultraviolet" } ;
```

```
struct ttime  // час в годинах і хвилинах
{
  int hour;  // година
  int minute;  // хвилини
};

struct  point  // параметри точки
{
  coordinates  xy;
  spectrum  color;
  ttime  tm;
} ;

point  wall[5] =  // масив 5 точок - визначаємо "вручну",
                  //  на практиці – читати з файла
{
  { { 16, 2 }, green, { 11, 50 } },
  { { -1, 14 }, orange, { 13, 30 } },
  { { 8, -2 }, yellow, { 8, 30 } },
  { { 0, 10 }, red, { 10, 10 } },
  { { 12, -9 }, yellow, { 15, 5 } }
} ;
```

```
// пошук жовтих точок
for (int k=0; k<5; k++)
  if (wall[k].color == yellow)
    cout << colorname[wall[k].color] << '\t' << "x="
         << wall[k].xy.x << '\t' << "y=" << wall[k].xy.y
         << '\t' << "time=" << wall[k].tm.hour << "."
         <<  wall[k].tm.minute << endl;
// найвіддаленіша від початку координат
float distance = -1;   int nm = -1;  // відстань і номер точки
float d;
for (int k=0; k<5; k++)
{
  d = sqrt( pow(wall[k].xy.x, 2) + pow(wall[k].xy.y, 2) );
  if ( d > distance ) { distance=d; nm=k; } // більше - запам'ятати
}
cout << "Maximum distance:  ";
cout << colorname[wall[nm].color] << '\t'
     << "x=" << wall[nm].xy.x << '\t'
     << "y=" << wall[nm].xy.y << '\t' << "time="
     << wall[nm].tm.hour << "." << wall[nm].tm.minute << endl;
```

Результати:

```
yellow   x=8      y=-2     time=8.30
yellow   x=12     y=-9     time=15.5
Maximum distance:  green   x=16   y=2  time=11.50
```