Структури (struct)

У попередніх розділах для зберігання даних розглянуто прості типи даних: числа, символи, рядки тощо. Але часто на практиці перед програмістом постає завдання запрограмувати той чи інший об`єкт, який характеризується водночас кількома параметрами, приміром, точка на площині задається парою дійсних чисел (*x*, *y*), а дані про людину можна задавати кількома параметрами: прізвище, ім`я, по-батькові – рядки, рік народження – число тощо. Для поєднання кількох параметрів в одному об`єкті в C++ існує спеціальний тип даних, який має назву структура. На відміну від масивів, які також є структурованим типом даних і містять елементи одного типу, структури дозволяють поєднувати дані різних типів.

Структура – це складений тип даних, змінні (поля) якого можуть містити різноманітну й різнотипну інформацію.

Опис структури має синтаксис

```
struct <ім`я_типу_структури>
      {
            <тип1> <поле1>;
            <тип2> <поле2>;
            . . .
                  <типN> <полеN>;
      } <список_змінних>;
```

Ім`я типу структури задається програмістом виходячи зі змісту даних, які зберігатимуться у структурі. Список змінних наприкінці оголошення структури може бути відсутнім, у цьому разі після фігурної дужки має бути крапка з комою.

Наведемо приклад оголошення структури з ім`ям sportsman, яка поєднає в собі інформацію про змагання спортсменів: прізвище і вік спортсмена, країна, кількість його очок, утворюючи відповідні поля:

```
struct sportsman
{
      char prizv[15]; char kraina[10]; // Прізвище та країна,
      int vik; float ochki; // вік та кількість очок.
      };
```

Як наслідок цього оголошення створено новий тип sportsman. Тепер у програмі цей тип може використовуватись нарівні зі стандартними типами для оголошення змінних. Оголошені змінні типу sportsman матимуть чотири поля: два рядки (prizv і kraina), ціле (vik) і дійсне (ochki) числа.

```
      sportsman Sp; /* Змінна Sp типу sportsman може зберігати інформацію про
одного спортсмена. */
      sportsman Mas[15]; /* Масив Mas типу sportsman може містити інформацію
про 15 спортсменів. */
```

При оголошуванні змінної типу структури пам`ять під усі поля структури виділяється послідовно для кожного поля. У наведеному прикладі структури sportsman

під змінну Sp послідовно буде виділено 15, 10, 4, 4 байти – усього 33 байти. Поля змінної Sp у пам`яті буде розміщено у такому порядку:

Слід зазначити, що сумарну пам`ять, яка виділяється під структуру, може бути збільшено компілятором на вимогу процесора для так званого вирівнювання адрес. Реальний обсяг пам`яті, яку займає структура, можна визначити за допомогою sizeof(). При оголошуванні структур їхні поля можна ініціалізовувати початковими значеннями. Наприклад, оголошення змінної Sp типу sporsman й ініціалізація її даними про 20-річного спортсмена з України за прізвищем Бойко, який набрав 75.3 очок, матиме вигляд:

```
sporsman Sp = { "Бойко", "Україна", 20, 75.3 };
```

Оголошення типу “точка на площині” POINT з полями-координатами – x та y і змінної spot цього типу – точки з координатами (20, 40) може бути таким:

```
struct POINT
      {
            int x, y;
      } spot = { 20, 40 }; // Точка має координати x = 20, y = 40
```

При ініціалізації масивів структури кожен елемент масиву слід записувати у фігурних дужках, наприклад при створенні двовимірного масиву compl розміром 2*x*3 типу структури complex з полями real та im ініціалізація початкових значень елементів може мати вигляд:

```
struct complex
      {
            float real, im;
      } compl[2][3] = { { { 1.4 },{ 5 },{ -3.1 } },{ { 2 },{ -5.2 },{ 0 } } };
```

Доступ до полів структури здійснюється за допомогою операції “крапка”:
<структурна змінна>.<ім'я поля>

Наприклад, для розглянутої змінної Sp типу sporsman надання значень її полям може мати вигляд:

```
      strcpy(Sp.prizv, "Бойко");
      strcpy(Sp.kraina, "Україна");
      Sp.vik = 20;
      Sp.ochki = 75.3;
```

Можна створювати тип структури за допомогою typedef, наприклад:

```
typedef struct
      {
            AnsiString priz, gruppa;
            int Bal_math, Bal_inform;
      } student; // Оголошення типу “студент”
```

```
student W; // і змінної W цього типу.
```

Тут означено тип структури за ім`ям student для зберігання даних про успішність студентів з полями: прізвище, група, екзаменаційні оцінки з математики та інформатики й оголошено змінну W цього типу.

Структури можуть розміщуватись у динамічній пам`яті за допомогою оператора new і оголошуватися як покажчики. У такому разі доступ до полів структури здійснюється за допомогою операції вибору – “стрілка” (->, на клавіатурі послідовно натискаються клавіші “мінус” і “більше”):

```
sportsman *p = new sportsman; // Виділення пам“яті,
p->ochki = 95.3; // присвоєння очок полю ochki
(*p).vik = 23; // і віку полю vik.
```

Зверніть увагу на те, що якщо p – покажчик на структуру, то (*p) – сама структура і доступ до її полів здійснюється за допомогою крапки.

Якщо змінні типу “структура” оголошуються у програмі лише один раз, то, зазвичай, їх записують наприкінці оголошення структури і при цьому ім`я структури можна явно не зазначати, наприклад:

```
struct
{
      char prizv[15];
      char kraina[10];
      int vik; float ochki;
}
Sp;
```

Тут оголошено змінну Sp як структуру з чотирма полями без створення типу “спортсмен”.

Поля структури можуть бути якого завгодно типу, у тому числі й масивом, покажчиком чи структурою, окрім типу тієї ж самої структури (але можуть бути покажчиком на неї). За приклад створимо дві структури для опрацювання інформації про співробітників у відділі кадрів: прізвище, ім`я, по-батькові співробітника, домашня адреса, домашній телефон, дата народження, дата вступу на роботу, зарплатня.

```
struct date
{
      int day, month, year; // День, місяць та рік
};
struct person
{
      char prizv[20], adresa[150]; // Прізвище, адреса,
      char phone[10]; date birth_date; // телефон, дата народження,
      date work_date; // дата вступу на роботу
      float salary; // та зарплатня.
};
```

У цьому прикладі оголошується новий тип – структура date для зберігання дати (день, місяць, рік). Окрім того, оголошується тип – структура person, яка використовує тип date для полів birth_date і work_date. Посилан-ня на поля вкладеної структури формується з імені структурної змінної, імені структурного поля й імені поля вкладеної структури. Те, що поля структур самі можуть бути структурами, надає можливість цілісно описувати доволі складні об`єкти реального світу. Оскільки структури, на відміну від масивів, зберігають дані різних типів, їх відносять до неоднорідних типів даних.

Оголосимо змінну створеного типу person:

```
person P;
```

Присвоювання прізвища у змінну P:

```
strcpy(P.prizv, "Шкуропат");
```

Присвоювання дати народження 5 березня 1987 року у змінну P:

```
P.birth_date.day=5;
P.birth_date.month=3;
P.birth_date.year=1987;
```

Розмістимо змінну man типу person у динамічній пам‟яті:

```
person *man = new person;
```

І присвоїмо ті ж самі значення:

```
strcpy(man->prizv, "Шкуропат");
(man->birth_date).day=5;
(man->birth_date).month=3;
(man->birth_date).year=1987;
```

Оголосимо масив з даними про 100 осіб:

```
person P[100];
```

Попередні присвоювання для четвертої людини:

```
strcpy(P[3].prizv, "Шкуропат");
P[3].birth_date.day=5;
P[3].birth_date.month=3;
P[3].birth_date.year=1987;
```

Уведемо значення всіх полів для всіх співробітників до масиву:

```
for (i = 0; i<100; i++)
     {
          cout << "Введіть призвіще: "
               gets(P[i].prizv);
          cout << "Введіть адресу: "
               gets(P[i].adresa);
          cout << "Введіть телефон: "
               gets(P[i].phone);
          cout << "Введіть дату народження: " << endl;
          cout << "День: "; cin >> P[i].birth_date.day;
          cout << "Місяць: "; cin >> P[i].birth_date.month;
          cout << "Рік: " cin >> P[i].birth_date.year;
          cout << "Введіть дату вступу до роботи: "endl;
          cout << "День: "; cin >> P[i].work_date.day;
          cout << "Місяць: "; cin >> P[i].work_date.month;
          cout << "Рік: " cin >> P[i].work_date.year;
     }
```

Структурні змінні одного типу можна переприсвоювати одну одній, при цьому виконується присвоювання усіх полів, наприклад:

```
person man=P[0];
```

Як вже було зазначено вище, поля структури можуть бути якого завгодно типу, у тому числі масивами. Наприклад, оголошення структури “погода за місяць” може мати вигляд

```
struct pogoda
	{
		int month; // Номер місяця,
		int temp[31];// масив температур за 31 день.
	};
	pogoda M; // Оголошення змінної M типу pogoda (погода за один місяць).
	pogoda G[12]; // і масиву G типу pogoda (погода за рік).
```

Визначимо значення температури повітря за 14 липня:

```
M.month = 7;
M.temp[13] = 27;
```

Присвоїмо випадкові значення температури за весь лютий:

```
M.month = 2;
for(int i=0; i<28; i++) M.temp[i] = random(5)-10;
```

Виведення температури повітря восени (за 3 місяці):

```
for (int i = 9; i <= 11; i++)
	{
		G[i - 1].month = i;
		for (int j = 0; j<31; j++)
			if (j == 30 && (i == 9 || i == 11))
				continue; // Пропустити 31.09 та 31.11
			else
				cout << G[i - 1].temp[j] << " ";
		cout << endl;
	}
```

У C++ структура при описуванні, окрім звичних полів, може містити елементи-функції. Наприклад, можна дописати до структури pogoda функцію month_name, яка визначатиме рядок – назву місяця за номером.

```
struct pogoda
	{
		int month, temp[31]; // Номер місяця і масив температур за 31
день.
		char *month_name()
		{
			switch (month)
			{
			case 1: return "Січень";
			case 2: return "Лютий";
			case 3: return "Березень";
			case 4: return "Квітень";
			case 5: return "Травень";
			case 6: return "Червень";
```

```
                case 7: return "Липень";
                case 8: return "Серпень";
                case 9: return "Вересень";
                case 10: return "Жовтень";
                case 11: return "Листопад";
                case 12: return "Грудень";
                default: return "Помилковий номер";
                }
        }
};
```

У програмі цю функцію можна викликати в такий спосіб:

```
pogoda M;
        . . .
                cin >> M.month;
        cout << M.month_name();
```

Після виконання цих операторів на екран буде виведено рядок, утворений за допомогою функції month_name().

Наведемо ще один приклад елемента-функції. Зорганізуємо структуру student з функцією Show(), яка будуватиме рядок типу AnsiString, поєдную-чи усі поля цієї структури. Надалі цей рядок можна буде використовувати для виведення записів на екран:

```
struct student
        {
                String prizv, grupa;
                int Bal_math, Bal_inform;
                String Show()
                {
                        return prizv + " " + grupa + " " + Bal_math+ " " +
                                Bal_inform;
                }
};
```

До функцій структури передаються як звичайні змінні.

Об’єднання (union)

*Об`єднання* – формат даних, який може містити різні типи даних, але лише один тип водночас. Можна сказати, що об`єднання є окремим випадком структури, всі поля якої розташовуються за однією й тією самою адресою. Формат опису об`єднання є такий самий як і у структури, лише замість ключового слова struct використовується слово union. Але, тоді як структура може містити, скажімо, елементи типу і int, і short, і double, об`єднання може містити чи то int, чи short, чи double:

```
union prim
{
        int x;
        short y;
        double z;
```

```
};
```

Обсяг пам`яті, яку займає об`єднання дорівнює найбільшому з розмірів його полів. Об`єднання застосовують для економії пам`яті у тих випадках, коли відомо, що понад одного поля водночас не потрібно. Часто об`єднання використовують як поле структури. При цьому до структури зручно включити додаткове поле, яке визначає, який саме елемент об`єднання використовується даного моменту:

```
struct
{
      int type;
      union id
      {
            long id_num; char id_ch[20];
      };
} price;
. . .
      if (price.type == 1) cin >> price.id.id_num;
      else cin >> price.id.id_ch;
```

Тут структура price містить ціле поле type та поле id, яке може набувати лише одне зі значень чи то цілого числа id_num, чи то рядка id_ch залежно від значення поля type. У наведеному прикладі ім`я об`єднання можна не зазначати, а звертатися безпосередньо до його полів. У цьому разі об`єднання є анонімним, його елементи стають змінними, розташованими за однією адресою.

```
struct
      {
            int type;
            union
            {
                  long id_num;
                  char id_ch[20];
            };
      } price;
      . . .
            if (price.type == 1) cin >> price.id_num;
            else cin >> price.id_ch;
```

Перерахування (enum)

При написанні програм часто виникає потреба у визначенні наперед відомої кількості іменованих констант, які мають мати різні значення (при цьому конкретні значення можуть бути неважливими). Для цього зручно користуватися типом перерахування enum, всі можливі значення якого задаються списком цілочисельних констант:

enum [<*ім`я_типу*>] {<*список_констант*>};

Ім`я типу задається в тому разі, якщо у програмі потрібно визначати змінні цього типу. Змінним перелічувального типу можна присвоювати кожне значення зі

списку констант, зазначених при оголошуванні типу. Імена перелічуваних констант мають бути унікальними. Перераховні константи подаються й опрацьовуються як цілі числа і можуть ініціалізуватися у звичайний спосіб. Окрім того вони можуть мати однакові значення. За відсутності ініціалізації перша константа обнулюється, а кожній наступній присвоюється значення на одиницю більше, аніж попередній.

```
enum week { sat = 0, sun = 0, mon, tue, wed, thu, fri } rob_den;
```

Тут описано перерахування week з відповідною множиною значень і оголошено змінну rob_den типу week. Перераховні константи sat та sun мають значення 0, mon – значення 1, tue – 2, wed – 3, thu – 4, fri – 5.

До перераховних змінних можна застосовувати арифметичні операції й операції відношення, наприклад:

```
enum days { sun, mon, tue, wed, thu, fri, sat };
      days day1 = mon, day2 = thu;
      int diff = day2 - day1;
      cout << "Різниця у днях: " << diff << endl;
      if (day1 < day2)
            cout << "day1 настане раніш, аніж day2 \n";
```

Арифметичні операції над змінними перелічуваного типів зводяться до операцій над цілими числами. Проте, не зважаючи на те, що компіляторові відомо про цілочисельну форму подання перераховних значень, варто використовувати цей факт надто обережно. Якщо спробувати виконати присвоювання

```
day1 = 5;
```

компілятор видасть повідомлення (хоча компіляція відбудеться без помилок). Рекомендовано без нагальної потреби не використовувати цілочисельну інтерпретацію перераховних значень.

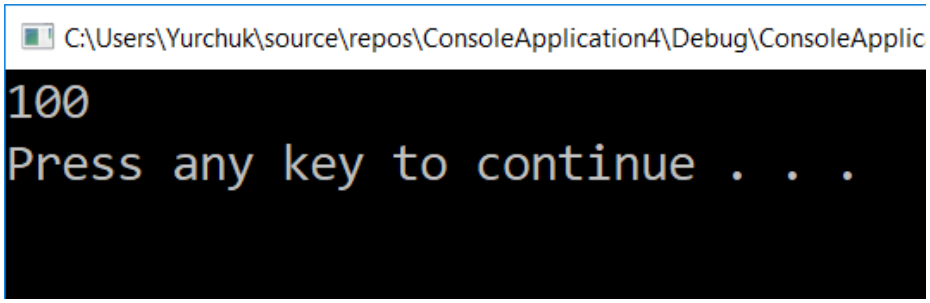
Значення перелічуваним константам можна встановлювати явно, ініціалізуючи їх при оголошенні, причому значення мають бути лише цілими числами, наприклад:

```
enum { first, second = 100, third };
```

У цьому разі first за замовчуванням дорівнює 0, а наступні неініціалізовані константи перерахування перевищують своїх попередників на 1. Приміром, third матиме значення 101.

Істотним недоліком типів перерахування є те, що вони не розпізнаються засобами введення-виведення C++. При виведенні такої змінної виводиться не її формальне значення, а її внутрішнє подання, тобто ціле число.

```
      enum xx { first, second = 100, third };
      xx a;
      a = second;
      cout << a;
```


C:\Users\Yurchuk\source\repos\ConsoleApplication4\Debug\ConsoleApplic
100
Press any key to continue . . .