

Лабораторна робота № 1

Робота з базовими типами даних

Мета роботи: отримати практичні навички по роботі з базовими типами даних (простими і складними типами даних).

Методичні вказівки

Тип даних визначає множину значень, набір операцій, які можна застосовувати до таких значень, а також спосіб реалізації зберігання значень і виконання операцій.

Розрізняють прості, складові та інші типи даних.

Прості типи даних можна розділити на:

- цілочисельні;
- дійсні;
- символьні;
- перерахування;
- логічні.

Складові (складні) типи даних можна розділити на:

- масиви;
- строкові;
- структури,
- об'єднання,
- класи.

Також існують інші типи даних:

- покажчики;
- посилання.

Коротко згадаємо основні типи даних.

Цілочисельні дані можуть бути представлені в знаковій та беззнаковій формі.

Беззнакові цілі числа можуть бути представлені у вигляді послідовності бітів в діапазоні від 0 до $2^n - 1$, де n - кількість розрядів.

Знакові цілі числа можуть бути представлені в діапазоні $-2^{n-1} \dots + 2^{n-1} - 1$. При цьому старший біт даних відводиться під знак числа («0» відповідає позитивному числу, «1» - негативному).

Дійсний тип призначений для представлення дробових чисел. Дійсні числа представляються в розрядній сітці обчислювальної машини у нормованій формі. *Нормована форма* числа передбачає наявність однієї значущої цифри до поділу цілої і дробової частини та множиться на основу системи числення у відповідній степені числа. Наприклад, число 12345,678 у нормованій формі для десяткової системи числення можна представити як:

$$12345,678 = 1,2345678 \cdot 10^4.$$

Або число 0,009876 у нормованій формі для десяткової системи числення можна представити як:

$$0,009876 = 9,876 \cdot 10^{-3}.$$

У двійковій системі числення значущий розряд, який розташовується перед роздільником цілої і дробової частини, може бути лише 1. У разі коли число не можна представити у нормованій формі (наприклад, число 0), тоді значущий розряд перед

роздільником цілої і дробової частини дорівнює 0. Значущі розряди числа в нормованій формі, які стоять після роздільника цілої та дробової частини, називають мантиєю числа.

Приклад: представимо дійсне число -178,125 у 32-розрядній сітці (тип float). Для цього виконаємо перетворення окремо цілої та дробової частин у двійкову систему числення:

$$178_{10} = 10110010_2.$$

$$0,125_{10} = 0,001_2.$$

Тоді

$$178,125_{10} = 10110010,001_2.$$

Для перетворення числа у нормовану форму здійснюється зсув коми на 7 розрядів вліво, щоб залишився один розряд.

$$178,125_{10} = 10110010,001_2 = 1,0110010001 \cdot 2^{11}$$

Для визначення ступеня числа виконаємо збільшення числа на 127, щоб уникнути від'ємних чисел:

$$0111111 + 00000111 = 10000110.$$

Таким чином, число -178,125 можна представити у 32-розрядній сітці як:

Зн.	Степень								Мантисса																								
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	1	1	0	0	0	0	1	1	0	0	1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Перерахування — тип даних у якого множина значень являє собою обмежений список ідентифікаторів.

Загальний синтаксис визначення шаблону перераховується типу має вигляд:

```
enum ім'я_шаблону {
    ідентифікатор1,
    ідентифікатор2,
    // інші ідентифікатори
};
```

Приклад визначення типу перерахування для днів тижня:

```
enum days_of_week {
    Mon,
    Tue,
    Wed,
    Thu,
    Fri,
    Sat,
    Sun
};
```

Замість типу перерахування можна було б використовувати оголошення констант наступним чином:

```
const char Mon = 0;
const char Tue = 1;
const char Wed = 2;
тощо.
```

Подібне рішення допустимо, але не цілком прийнятно, оскільки ми не можемо, наприклад, гарантувати, що аргумент, який передається в функцію, буде дорівнювати лише 0, 1 або 6. Використання типу перерахування вирішує дану проблему, тобто тепер змінні можуть приймати лише значення, які вказані у перерахуванні.

Якщо необхідно, щоб перерахування відповідали певним значенням, наприклад, починалися з одиниці, константи перерахування необхідно ініціалізувати. У цьому випадку попередній приклад можна записати в наступному вигляді:

```
enum days_of_week {  
    Mon=1,  
    Tue,  
    Wed,  
    Thu,  
    Fri,  
    Sat,  
    Sun  
};
```

Структури — набір різних типів даних, що зберігаються як єдине ціле, але передбачають доступ до окремих полів структури. Для доступу до полів структури використовується точкова нотація.

Структури:

- полегшують написання і розуміння програм;
- допомагають згрупувати дані, що об'єднуються будь-яким загальним поняттям;
- дозволяють групу пов'язаних між собою змінних використовувати як множина окремих елементів, а також як єдине ціле.

Структури доцільно використовувати там, де необхідно об'єднати дані, що відносяться до одного об'єкту.

Визначення структури складається з двох кроків:

- визначення шаблону структури (задання нового типу даних, що визначається користувачем);
- оголошення змінних визначеного шаблону.

Загальний синтаксис визначення шаблону структури має вигляд:

```
struct ім'я_шаблону {  
    тип1 назва_змінної1;  
    тип1 назва_змінної2;  
    // інші ідентифікатори даних;  
};
```

Назва шаблонів повинна бути унікальною в межах області визначення для того, щоб компілятор міг розрізняти різні типи шаблонів. Задання шаблону здійснюється за допомогою ключового слова *struct*, за яким слідує ім'я шаблону структури і список елементів, що розміщений у фігурних дужках.

Назви елементів в одному шаблоні також повинні бути унікальними. Однак в різних шаблонах можна використовувати однакові назви елементів.

Задання тільки шаблону не потребує резервування пам'яті компілятором. Шаблон надає компілятору необхідну інформацію про елементи змінної структурного типу для

резервування місця в оперативній пам'яті, організацію доступу до змінної і використання окремих елементів структурної змінної.

Приклад оголошення шаблону структури типу *date*:

```
struct date {  
    unsigned char nWeekDay  
    unsigned char nMonthDay;  
    unsigned char nMonth;  
    unsigned char nYear; //останні 2 цифри року  
};
```

Багато задач вимагають створення мінімальних за розміром програм. Один із способів економії пам'яті у програмах, використовуючи мову програмування C/C++, полягає в використанні *бітових полів* для представлення сукупності даних цілого типу.

В цьому випадку синтаксис визначення шаблону структури матиме вигляд:

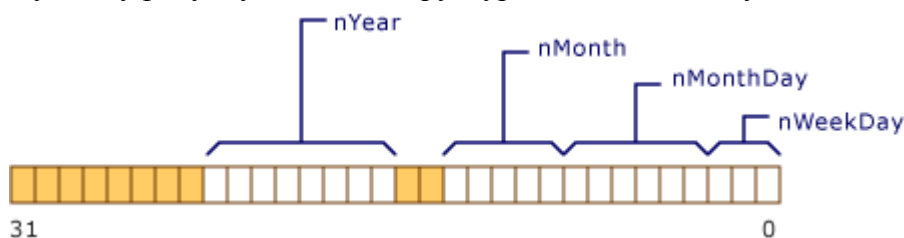
```
struct ім'я_шаблону {  
    тип1 назва_змінної1 : r1;  
    тип1 назва_змінної 1: r2;  
    // інші ідентифікатори : r3;  
};
```

де r1, r2, r3 — розрядність полів структури.

У наступному прикладі визначається структура типу *date*, яка аналогічна попередньому прикладу, але містить бітові поля:

```
struct date {  
    unsigned short nWeekDay : 3; // 0..7 (3 bits)  
    unsigned short nMonthDay : 6; // 0..31 (6 bits)  
    unsigned short nMonth : 5; // 0..12 (5 bits)  
    unsigned short nYear : 8; // 0..100 (8 bits)  
};
```

На наступному рисунку показана структура пам'яті для типу *date*.



Структура займає у пам'яті 32 біти (2 слова *unsigned short*). Перші 4 поля розміщені у першому слові *unsigned short* (сума розрядів усіх полів дорівнює 14 біт, що не перевищує 16 біт). Зверніть увагу, що поле *nYear* має розрядність 8 біт і не може бути розміщене в останніх 2 бітах *unsigned short*, тому воно перебуває на початку нового слова *unsigned short*. Зовсім не обов'язково, щоб всі бітові поля поміщалися в один об'єкт базового типу. Допускається використання анонімних бітових полів (тобто бітові поля без ідентифікатору). Неіменоване бітове поле шириною 0 забезпечує вирівнювання наступного бітового поля по кордону типу члена структури.

Об'єднання є значення або структура даних, яке може мати кілька різних представлень даних.

Об'єднання схожі на структури, з тією різницею, що всі поля розміщуються за однією адресою. Це означає, що розмір об'єднання дорівнює розміру найбільшого його поля.

Об'єднання можуть бути корисні для економії пам'яті при наявності множини об'єктів і/або обмеженій кількості пам'яті. Однак для їх правильного використання потрібна підвищена увага.

Загальний синтаксис оголошення шаблону об'єднання має вигляд:

```
union ім'я_шаблону {  
    тип1 назва_змінної1;  
    тип1 назва_змінної2;  
    // інші ідентифікатори даних;  
};
```

Розглянемо приклад. Припустимо, що необхідно до 16-розрядного значення мати можливість звертатися побайтово. В цьому випадку можна оголосити об'єднання:

```
union data16 {  
    struct {  
        unsigned char low;  
        unsigned char high;  
    } bytes;  
    unsigned short word;  
};
```

Порядок виконання роботи

1. Записати і заповнити структуру даних зберігання поточного часу (включаючи секунди) і дату в найбільш компактному вигляді. Визначити обсяг пам'яті, яку займає змінна даного типу. Порівняти зі стандартною структурою *tm* (*time.h*). Вивести вміст структури в зручному вигляді для користувача на дисплей..
2. Реалізувати введення цілочисельного значення типу *signed short*. Визначити знак і значення, використовуючи: 1) структури даних та об'єднання; 2) побітові логічні операції.
3. Виконати операції:
а) $5 + 127$; б) $2-3$; в) $-120-34$; г) (*unsigned char*) $(- 5)$; д) $56 \& 38$; е) $56 | 38$.

Всі значення (константи) повинні зберігатися в змінних типу *signed char*. Виконати перевірку результату в ручну. Пояснити результат, використовуючи двійкову систему числення.

4. Записати і заповнити структуру даних (об'єднання) для зберігання дійсного числа типу *float* в найбільш компактному вигляді. Реалізувати відображення на дисплей: 1) значення побитово; 2) значення побайтово; 3) знака, мантиси і ступінь значення. Виконати перевірку результату в ручну. Визначити обсяг пам'яті, яку займає змінна користувацького типу.