

Лекція 9.

Створення нових типів. Struct. Enum.



План на сьогодні

1

Створення нових типів

2

Оголошення структур (struct)

3

Використання структур

4

Перелічувальні типи (enum)

5

Використання enum



Нові типи даних



Нові типи даних

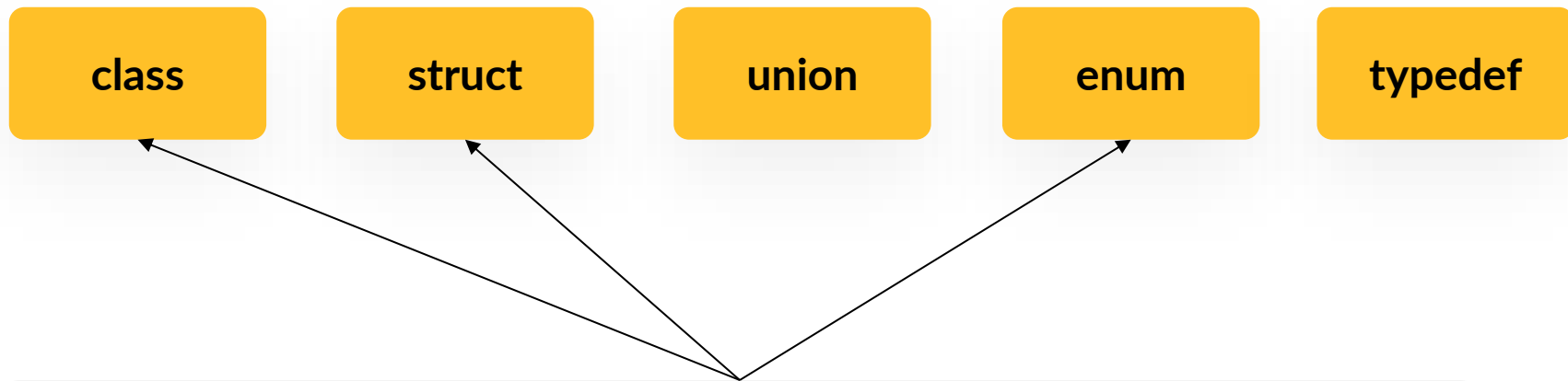
C++ дозволяє програмістам створювати свої власні типи даних.

Це дозволяє об'єднувати різні типи змінних в один логічний об'єкт.

Сприяє модульності та збільшує перевикористання коду.

Дозволяє писати “безпечний” код (де менше багів).

Користувацькі типи в C++



Ці типи поширені також і в інших мовах програмування (C#, Java, etc.)

struct



FACULTY OF APPLIED
MATHEMATICS AND
INFORMATICS
LVIV UNIVERSITY

Що таке struct?

Struct (структура) дозволяє згрупувати змінні різних типів даних разом в єдине ціле для представлення однієї сутності.

На відміну від масиву, структура може містити багато різних типів даних (int, string, bool тощо).

Структура оголошується за допомогою ключового слова «**struct**».

Оголошення структури

Ключове слово `struct`

Назва структури

```
struct Employee  
{  
    short id;  
    int age;  
    double wage;  
};
```

Поля структури

Поля - є звичайними змінними C++. Ми можемо створювати структури зі змінними різних типів даних у C++.

Методи структури

```
struct Student
{
    // Data Members
    int roll;
    int age;
    int marks;

    // Member Functions
    void printDetails()
    {
        cout<<"Roll = "<<roll<<"\n";
        cout<<"Age = "<<age<<"\n";
        cout<<"Marks = "<<marks;
    }
}
```

Метод структури

Методи - є звичайними функціями C++. Окрім змінних, ми також можемо включати функції всередину оголошення структури.

Розмір структури

```
- struct Test1
{
    int a;        // size 4, padding: 4
    char* ptr;    // size 8, padding: 0
    bool b;       // size 1, padding: 7
};               // overall padding: 8 -> size: 24

- struct Test2
{
    char* ptr;    // size 8, padding: 0
    int a;        // size 4, padding: 0
    bool b;       // size 1, padding: 3
};               // overall padding: 3 -> size: 16
```

Використання об'єкту

joe - об'єкт типу Employee

```
struct Employee  
{  
    short id;  
    int age;  
    double wage;  
};
```

Доступ до полів об'єкту joe

```
Employee joe;  
joe.id = 14;  
joe.age = 32;  
joe.wage = 24.15;
```

Ініціалізація структури

```
#include <iostream>
using namespace std;

struct Point {
    int x = 0; // It is Considered as Default Arguments and no Error is Raised
    int y = 1;
};

int main()
{
    Point p1;

    // Accessing members of point p1
    // No value is Initialized then the default value is considered. ie x=0 and y=1;
    cout << "x = " << p1.x << ", y = " << p1.y<<endl;

    // Initializing the value of y = 20;
    p1.y = 20;
    cout << "x = " << p1.x << ", y = " << p1.y;
    return 0;
}
```

x=0, y=1

x=0, y=20

Ініціалізація структури

```
struct Point {  
    int x, y;  
};  
  
int main()  
{  
    // A valid initialization. member x gets value 0 and y  
    // gets value 1. The order of declaration is followed.  
    Point p1 = { 0, 1 }; // x = 0; y = 1;  
  
    // C++11  
    Point p2 { 0, 1 }; // x = 0; y = 1;  
  
}
```

Приклад

```
// Declare a structure named "car"
struct Car {
    string brand;
    string model;
    int year;
};

int main() {
    // Create a car structure and store it in myCar1;
    Car myCar1;
    myCar1.brand = "BMW";
    myCar1.model = "X5";
    myCar1.year = 1999;

    // Create another car structure and store it in myCar2;
    Car myCar2;
    myCar2.brand = "Ford";
    myCar2.model = "Mustang";
    myCar2.year = 1969;

    // Print the structure members
    cout << myCar1.brand << " " << myCar1.model << " " << myCar1.year << "\n";
    cout << myCar2.brand << " " << myCar2.model << " " << myCar2.year << "\n";

    return 0;
}
```

BMW X5 1999

Ford Mustang 1969

Передавання struct в функцію

```
void printInformation(const Employee& employee)
{
    std::cout << "ID: " << employee.id << "\n";
    std::cout << "Age: " << employee.age << "\n";
    std::cout << "Wage: " << employee.wage << "\n";
}

void main()
{
    Employee joe = { 14, 32, 24.15 };
    Employee frank = { 15, 28, 18.27 };

    printInformation(joe);
    printInformation(frank);
}
```

```
struct Employee
{
    short id;
    int age;
    double wage;
};
```

```
ID: 14
Age: 32
Wage: 24.15
```

```
ID: 15
Age: 28
Wage: 18.27
```

Можна передати всю-структуру в функцію, яка повинна працювати з її членами

Вкладені структури

```
struct Employee
{
    short id;
    int age;
    float wage;
};
```

```
struct Company
{
    Employee CEO;
    int numberOfEmployees;
};
```

Тип Company використовує тип Employee, як одну зі змінних.

```
Company myCompany = {{ 1, 42, 60000.0f }, 5 };
```


Вказівник на структуру

```
#include <iostream>
using namespace std;

struct Point {
    int x;
    int y;
};

int main()
{
    struct Point p1 = { 1, 2 };

    // p2 is a pointer to structure p1
    struct Point* p2 = &p1;

    // Accessing structure members using
    // structure pointer
    cout << p2->x << " " << p2->y;
    return 0;
}
```



1 2

Як з примітивними типами, ми можемо мати вказівник на структуру. Якщо ми маємо вказівник на структуру, доступ до її членів здійснюється за допомогою оператора стрілки (->) замість оператора крапки (.).

Массив структур

```
struct Point {  
    int x, y;  
};  
  
int main(){  
    const size_t n = 3;  
  
    struct Point arr[n];  
  
    cout << "Enter 3 points:\n";  
  
    for (size_t i = 0; i < n; i++){  
        cout << "arr[" << i << "].x=";  
        cin >> arr[i].x;  
        cout << "arr[" << i << "].y=";  
        cin >> arr[i].y;  
    }  
  
    cout << "You Entered:\n";  
    for (size_t i = 0; i < n; i++){  
        cout << "Point " << i << " (" << arr[i].x << ", " << arr[i].y << ")" << endl;  
    }  
  
    return 0;  
}
```

enum

Що таке enum?

Перелічувальні типи (Enums) — це користувацькі типи, які складаються з іменованих цілих констант.

Вони допомагають присвоїти константам набір імен, що робить програму легшою для читання, підтримки та розуміння.

Перерахування оголошується за допомогою ключового слова «**enum**».

Оголошення enum

Оголошення enum

```
// Визначення нового перелічення Color
enum Color
{
    COLOR_BLACK,
    COLOR_RED,
    COLOR_BLUE,
    COLOR_GREEN,
    COLOR_WHITE,
    COLOR_CYAN,
    COLOR_YELLOW,
    COLOR_MAGENTA
};
```

```
// Визначення змінних типу Color

Color paint = COLOR_WHITE;
Color house(COLOR_BLUE);
```

Оголошення enum не виділяє пам'ять. Коли змінна типу enum визначається (наприклад paint), тоді виділяється пам'ять для цієї змінної.

Перелічування - enum

Кожен перелічувач отримує ціле значення, залежно від його позиції в списку enum.

За замовчуванням, перший - 0, і кожний наступний - має значення на одиницю більше попереднього

```
enum Color
{
    COLOR_BLACK, // = 0
    COLOR_RED, // = 1
    COLOR_BLUE, // = 2
    COLOR_GREEN, // = 3
    COLOR_WHITE, // = 4
    COLOR_CYAN, // = 5
    COLOR_YELLOW, // = 6
    COLOR_MAGENTA // = 7
};
```

Перелічування - enum

Оскільки перелічувачі розташовані в тому ж просторі імен, що і ціле перелічення, то ім'я перелічувача не може бути використано в декількох переліченнях в межах одного простору імен:

```
enum Color{  
    RED,  
    BLUE, // BLUE є в глобальному просторі імен  
    GREEN  
};  
  
enum Feeling{  
    HAPPY,  
    TIRED,  
    BLUE // error, BLUE вже визначено в enum Color  
};
```

Приклад симуляція світлофору

```
// Enum to represent the states of a traffic light
enum TrafficLight {
    RED,
    YELLOW,
    GREEN,
};
```

```
The numerical value assigned to lighth1 : 0
The numerical value assigned to lighth2 : 1
The numerical value assigned to lighth3 : 2
```

```
void printNumericValues( ) {

    TrafficLight lighth1 = RED;
    TrafficLight lighth2 = YELLOW;
    TrafficLight lighth3 = GREEN;

    cout << "The numerical value "
          << "assigned to lighth1 : " << lighth1 << endl;

    cout << "The numerical value "
          << "assigned to lighth2 : " << lighth2 << endl;

    cout << "The numerical value "
          << "assigned to lighth3 : " << lighth3 << endl;
}
```


Перетворення в int

Оскільки перелічувачі мають цілі значення, вони можуть бути присвоєні цілочисельним змінним. Також вони виводяться на консоль як цілі числа:

Компілятор не може неявно конвертувати цілі числа до типу `enum`.

```
int mypet = ANIMAL_PIG;  
cout << ANIMAL_HORSE; // 5
```

```
Animal animal = 5; // помилка компіляції
```

```
Color color = static_cast<Color>(5); // ugly
```

```
int inputColor;  
cin >> inputColor;  
Color color = static_cast<Color>(inputColor);
```

Магічні значення


Перелічування корисні для зручності читання коду, уникнення помилок, коли вам потрібно представити певний, заздалегідь визначений набір станів.

Наприклад – функції часто повертають ціле число, коли щось сталося всередині:

Магічні значення -
джерело помилок!

```
int readFileContents()
{
    if (!openFile())
        return -1;
    if (!readFile())
        return -2;
    if (!parseFile())
        return -3;

    return 0; // success
}
```



Магічні значення

Перелічування корисні для зручності читання коду, уникнення помилок, коли вам потрібно представити певний, заздалегідь визначений набір станів.

Немає магiчних значень



```
enum ParseResult
{
    SUCCESS = 0,
    ERROR_OPENING_FILE = -1,
    ERROR_READING_FILE = -2,
    ERROR_PARSING_FILE = -3
};

ParseResult readFileContents()
{
    if (!openFile())
        return ERROR_OPENING_FILE;
    if (!readFile())
        return ERROR_READING_FILE;
    if (!parsefile())
        return ERROR_PARSING_FILE;

    return SUCCESS;
}

if (readFileContents() == SUCCESS)
{
    // do something
}
else
{
    // print error message
}
```

enum class

C++ 11 визначає нову концепцію: `enum class` (також називають областю дії перелічування), що робить перелічення сильно типізованим.

Приклад з enum class

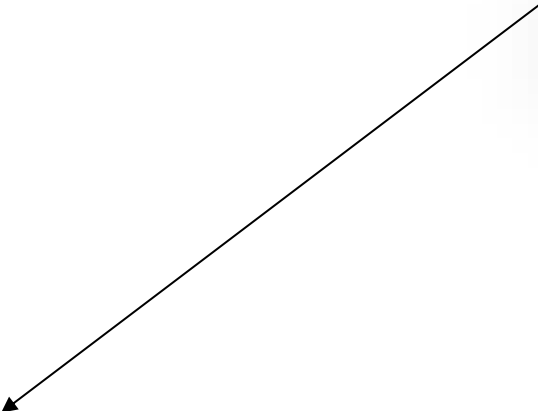
```
enum class Color
```

```
{  
    RED,  
    BLUE  
};
```

```
enum class Fruit
```

```
{  
    BANANA,  
    APPLE  
};
```

Вказуємо тип
перелічення



```
Color color = Color::RED;
```

```
Color color1 = static_cast<Color>(0) // погана практика, бажано так не робити
```

```
Fruit fruit = Fruit::BANANA;
```

```
if (color == fruit) // помилка компіляції, оскільки не знає як порівняти два різних типи Color та Fruit
```

```
    std::cout << "color and fruit are equal\n";
```

```
else
```

Дякую!



FACULTY OF APPLIED
MATHEMATICS AND
INFORMATICS
LVIV UNIVERSITY