Kiểu dữ liệu tự định nghĩa trong C++

Nguyễn Đức Thắng

Tại sao cần sử dụng kiểu dữ liệu tự định nghĩa?

- Trong quá trình lập trình, những kiểu dữ liệu định nghĩa có sẵn có thể không mang lại ý nghĩa phù hợp.

```
const int SUNDAY = 1;
const int MONDAY = 2;
const int TUESDAY = 3;
const int WEDNESDAY = 4;
const int THURSDAY = 5;
const int FRIDAY = 6;
const int SATURDAY = 7;
```

```
int DAYS_OF_WEEK[7] = { 1, 2, 3, 4, 5, 6, 7 };
```

```
const int WEDNESDAY = 4;
const int SUNDAY = 1;
const int TUESDAY = 3;
const int FRIDAY = 6;
const int MONDAY = 2;
const int SATURDAY = 7;
const int THURSDAY = 5;
```

```
const float PI = 3.14;
const float ACCELERATION_OF_GRAVITY = 9.8;
const int MAX_SIZE_OF_ARRAY = 255;
//...............
```

Kiểu liệt kê enum

- Kiểu liệt kê giúp thay thế các con số bằng những cái tên có ý nghĩa, giúp chúng ta tập hợp các giá trị có ý nghĩa liên quan với nhau thành từng nhóm.

```
enum <name_of_enumeration>
{
        //list all of values inside this block
        //each enumerator is separated by a comma, not a semicolon
};
```

Kiểu liệt kê enum

- Việc khai báo kiểu dữ liệu mới (như kiểu enum) không yêu cầu chương trình cấp phát bộ nhớ, lúc nào chúng ta sử dụng thì chương trình mới cấp phát bộ nhớ.

```
enum DaysOfWeek
{
        SUNDAY,
        MONDAY,
        TUESDAY,
        WEDNESDAY,
        THURSDAY,
        FRIDAY,
        SATURDAY
};
```

```
enum Color
{
        RED,
        GREEN,
        BLUE,
        WHITE
};
```

```
enum Animal
{
        CAT,
        DOG,
        HORSE,
        MONKEY,
        CHICKEN
};
```

- Giá trị các biến enum là gì???

Kiểu liệt kê enum

- Bên cạnh giá trị tự động, chúng ta cũng có thể chủ động thay đổi giá trị cho kiểu enum (nhưng chỉ thay đổi trong phần khai báo), khi đã định nghĩa xong thì không thể thay đổi nữa.

```
enum Direction
{
        UP = 1,    //assigned 1 by programmer
        DOWN = 3, //assigned 3 by programmer
        LEFT,      //assigned 4 by compiler
        RIGHT      //assigned 5 by compiler
};

cout << UP << " " << DOWN << " " << LEFT << " " << RIGHT << endl;
```

- Những hằng số trong cùng 1 enum có thể có giá trị giống nhau. Không nên tự gán giá trị cho enum.

Sử dụng kiểu liệt kê enum

```cpp
#include <iostream>
using namespace std;
enum Test
{
    TEST1,
    TEST2,
    TEST3
};

enum Color
{
    COLOR_BLACK,
    COLOR_RED,
    COLOR_BLUE,
    COLOR_GREEN,
    COLOR_WHITE,
    COLOR_CYAN,
    COLOR_YELLOW
};
```

```cpp
int main()
{
    Color backgroundColor;
    backgroundColor = COLOR_GREEN; // True
    backgroundColor = 4; // error
    Color backgroundColor2 = TEST1; // error
}
```

Sử dụng kiểu liệt kê enum

```
enum BossState
{
        IDLING,
        RUNNING,
        JUMPING,
        DYING
};

BossState state;

void initBoss()
{
        //init something
        state = IDLING;
}

void attack()
{
        //............
}

void activated()
{
        //............
}
```

```
void updateAnimation(BossState state)
{
        switch(state)
        {
        case IDLING:
                standStill();
                break;

        case RUNNING:
                setRunningAnimation();
                break;

        case JUMPING:
                setJumpingAnimation();
                break;

        case DYING:
                setDyingAnimation();
                break;

        default;
                break;
        }
}
```

Phạm vi sử dụng của kiểu liệt kê enum

- Tương tự như khai báo biến

```
void foo()
{
        enum ItemTypes
        {
                LAPTOP,
                DESKTOP,
                MOBILE,
                NETWORK
        };

        cout << MOBILE << endl;
}

int main()      {

        cout << DESKTOP << endl; //error

        return 0;
}
```

Vấn đề với kiểu dữ liệu enum

- Khi chương trình có nhiều enum tự định nghĩa, sẽ có nhiều giá trị trùng nhau giữa các enum khác nhau.

```
enum Color
{
        RED,
        GREEN,
        BLUE
};

enum Fruit
{
        APPLE,
        BANANA
};

int main() {

        Color color = GREEN;
        Fruit fruit = BANANA;

        if (color == fruit)
                cout << "It's the same" << endl;
        else
                cout << "It's not the same" << endl;

        return 0;
}
```

Enum class

```cpp
enum class Fruit
{
    APPLE,
    BANANA
};

enum class Color
{
    CRED,
    GREEN,
    BLUE
};

int main()
{
    Color color = Color::GREEN;
    Fruit fruit = Fruit::BANANA;

    if (color == fruit)
        cout << "It's the same" << endl;
    else
        cout << "It's not the same" << endl;

    return 0;
}
```

```cpp
if (color == 2) // error
{

}
```

```cpp
if(color == Color::GREEN)
{
        //OK
}
```

struct

- Giả sử chúng ta cần lưu trữ thông tin cá nhân của nhiều người:

```
std::string name[10];
std::string currentJob[10];
std::string homeAddress[10];
int birthYear[10];
int birthMonth[10];
int birthDay[10];
float height[10];
float weight[10];
//...............
```

-> Nhiều biến, khó quản lý chương trình

- **Struct** cho phép nhóm nhiều biến có thể sử dụng khác nhau để lưu trữ một tập các dữ liệu cần thiết mô tả một đơn vị nào đó.

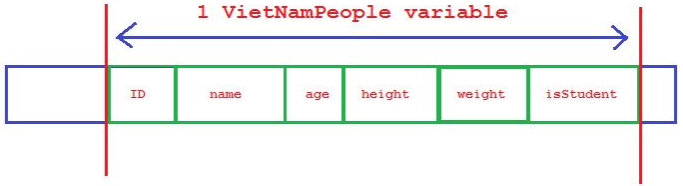
Cú pháp khai báo struct

- Tổng quát:

```
struct <name_of_new_type>
{
        <variables>;
};
```

- Ví dụ:

```
struct VietNamPeople
{
        __int32 ID;
        std::string name;
        __int16 age;
        float height;
        float weight;
        bool isStudent;
};
```



Khởi tạo cho biến struct

- Ví dụ:

```
struct Employee
{
        __int32 ID;
        std::string name;
        __int32 age;
        __int32 year_of_exp;
};

//.................

Employee nguyenDucThang = { 1, "Nguyen Duc Thang", 28, 5 };
```

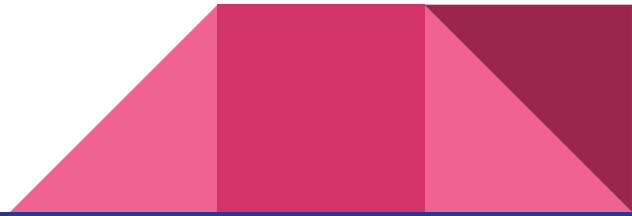
- Giá trị mặc định sẽ được khởi tạo nếu không được chỉ định cụ thể.

Truy cập dữ liệu biến struct

- Cú pháp: <tên biến>.<thuộc tính>

- Thực hiện phép gán và xuất như biến bình thường.

- Truyền vào hàm không cần truyền tất cả các thuộc tính. Có thể truyền tham trị hay tham chiếu như 1 biến bình thường, cũng có thể sử dụng làm kiểu trả về của hàm.



Sử dụng kiểu dữ liệu struct

- Chúng ta có thể định nghĩa hàm bên trong struct

```
struct Vector2D
{
        float x;
        float y;

        void normalize()
        {
                float length = sqrt(x * x + y * y);
                x = x / length;
                y = y / length;
        }
};
```

```
int main()
{
        Vector2D vec = { 1, 4 };
        vec.normalize();

        printVector2D(vec);

        return 0;
}
```

Cấu trúc lồng nhau của struct

- Struct là một tập hợp kiểu dữ liệu dùng để tạo nên dữ liệu mới, nên có thể sử dụng 1 kiểu struct khác để làm 1 trường cho dữ liệu struct được tạo ra.

```
struct Birthday
{
        __int32 day;
        __int32 month;
        __int32 year;
};

struct Employee
{
        __int32 ID;
        std::string name;
        Birthday birthday;
        __int32 year_of_exp;
};
```

Truy xuất struct lồng nhau

```
Employee emp = { 1, "Nguyen Duc Thang", {1, 2, 2000}, 5 };

std::cout << emp.ID << std::endl;
std::cout << emp.name << std::endl;
std::cout << emp.birthday.day << "/" << emp.birthday.month << "/" <<emp.birthday.year
std::cout << emp.year_of_exp << std::endl;
```

Bài tập

1. Định nghĩa kiểu dữ liệu PhanSo đại diện cho kiểu phân số. Qua đó, viết chương trình cho phép người dùng thực hiện các phép cộng, trừ, nhân, chia trên 2 phân số.

2. Viết chương trình thực hiện phân tích thống kê 1 lớp khoảng 20 sinh viên. Thông tin của mỗi sinh viên gồm ID, tên, tuổi, điểm tổng kết học kỳ 1, điểm tổng kết học kỳ 2. Những thông tin cần thống kê bao gồm:

- Điểm trung bình cuối năm của cả lớp
- Điểm tổng kết cuối năm của sinh viên nào là cao nhất
- Liệt kê danh sách những sinh viên có tiến bộ trong học tập

Con trỏ và struct

- Giả sử chúng ta cần định nghĩa kiểu dữ liệu letter.

```
struct Letter
{
        char from[50];
        char to[50];
};
```

- Vấn đề:

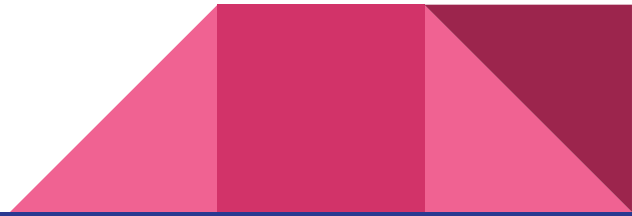
```
int main()
{
        Letter myLetter;
        std::cout << "Address of myLetter: " << &myLetter << std::endl;
        std::cout << "Address of from field: " << &myLetter.from << std::endl;

        return 0;
}
```

Con trỏ và struct

- Địa chỉ biến đầu tiên trong struct cũng chính là địa chỉ biến struct đó.

- Có thể trỏ đến kiểu struct bằng con trỏ cùng kiểu với struct.

```
Letter myLetter;
Letter *pLetter = &myLetter;
```



Truy xuất phần tử của con trỏ struct

```
struct BankAccount
{
        __int64 accountNumber;
        __int64 balance;
};

int main()
{
        BankAccount myAccount = { 123456789, 50 }; // $50
        BankAccount *pAccount = &myAccount;

        std::cout << "My bank account number: " << myAccount.accountNumber << std::endl;
        std::cout << "My bank account number: " << pAccount->accountNumber << std::endl;

        std::cout << "My balance: " << myAccount.balance << std::endl;
        std::cout << "My balance: " << pAccount->balance << std::endl;

        return 0;
}
```

Một số nhầm lẫn khi sử dụng con trỏ struct

- Nhầm lẫn khởi tạo biến struct thường và biến con trỏ struct.

```
struct BankAccount
{
        __int64 accountNumber;
        __int64 balance;
};

int main()
{
        BankAccount myAccount = { 12345, 50 };

        BankAccount *pAccount = { 12345, 50 }; //error

        return 0;
}
```

Một số nhầm lẫn khi sử dụng con trỏ struct

- Khắc phục lỗi: Trỏ đến con trỏ struct khác, hoặc cấp phát bộ nhớ.

```
struct BankAccount
{
        __int64 accountNumber;
        __int64 balance;
};

int main()
{
        BankAccount myAccount = { 0, 0 };

        BankAccount *pAccount = &myAccount;

        *pAccount = { 12345, 50 };

        std::cout << pAccount->accountNumber << " " << pAccount->balance << std::endl;

        return 0;
}
```

```
BankAccount *pAccount = new BankAccount;
*pAccount = { 12345, 50 };
```

Chú ý

- Phân biệt rõ cách truy xuất của struct thường với con trỏ struct.

```
struct BankAccount
{
        Date registrationDate;
        __int64 accountNumber;
        __int64 balance;
};

int main()
{
        BankAccount *pAccount = new BankAccount;
        *pAccount = { {2, 5, 2016}, 12345, 50 };

        std::cout << pAccount->registrationDate.year << std::endl;

        return 0;
}
```