



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

CS205

Prof. Shiqi Yu (于仕琪)

yusq@sustech.edu.cn

<http://faculty.sustech.edu.cn/yusq/>



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Classes and Objects



Structures

- A `struct` in **C** is a type consisting of a sequence of data members.
- Some functions/statements are needed to operate the data members of an object of a `struct` type.

```
struct Student
{
    char name[4];
    int born;
    bool male;
};

struct Student stu;
strcpy(stu.name, "Yu");
stu.born = 2000;
stu.male = true;
```

		13
		12
		11
		10
		9
male	1	8
		7
born	2000	6
		5
		4
	0	3
	0	2
name	'u'	1
	'Y'	0
		-1
		-2



Classes

- You should be very careful to manipulated the data members in a `struct` object.
- Can we improve `struct` to a better one?
- Yes, it is `class`! We can put some member functions in it.

Safer solution!

firstclass.cpp

```
class Student
{
public:
    char name[4];
    int born;
    bool male;
    void setName(const char * s)
    {
        if (s == NULL){...}
        size_t len = sizeof(name) - 1;
        strncpy(name, s, sizeof(name));
        name[len] = '\0';
    }
    void setBorn(int b) { ... }
```

```
Student yu;
yu.setName("Yu");
```



Access Specifiers

- You can protect data members by access specifier **private**.
- Then data member can only be accessed by well designed member functions.

access-attribute.cpp

```
class Student
{
    private:
        char name[4];
        int born;
        bool male;
    public:
        void setName(const char * s)
        {
            ...
        }
        void setBorn(int b)
        {
            ...
        }
}
```

Student yu;
yu.born = 2001;





Member Functions

- A member function can be defined inside or outside class.

inline function



```
class Student
{
private:
    char name[4];
    int born;
    bool male;
public:
    void setName(const ch
    {
        strncpy(name, s, sizeof(name));
    }
    void setBorn(int b)
    {
        born = b;
    }
    void setGender(bool isMale);
    void printInfo();
};
```

```
inline void Student::setGender(bool isMale)
{
    male = isMale;
}
void Student::printInfo()
{
    cout << "Name: " << name << endl;
    cout << "Born in " << born << endl;
    cout << "Gender: " << (male ? "Male" : "Female") << endl;
}
```



File Structures

- The source code can be placed into multiple files

student.hpp

```
#pragma once
class Student
{
    private:
        char name[4];
        int born;
        bool male;
    public:
        void setName(const ch
        {
            // ...
        }
        void setBorn(int b)
        {
            // ...
        }
        void setGender(bool isMale);
        void printInfo();
};
```

student.cpp

```
#include "student.hpp"
void Student::setGender(bool isMale)
{
    male = isMale;
}
void Student::printInfo()
{
    cout << "Name: " << name << endl;
    cout << "Born in " << born << endl;
    cout << "Gender: " << (male ? "Male" : "Female") << endl;
}
```

main.cpp

```
#include "student.hpp"
int main()
{
    Student yu;
    yu.setName("Yu");
    yu.setBorn(2000);
}
```



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Constructors and Destructors



Constructors

- Different from `struct` in C, a constructor will be invoked when creating an object of a `class`.
 - `struct` in C: allocate memory
 - `class` in C++: allocate memory & invoke a constructor
- But ... No constructor is defined explicitly in previous examples.
 - The compiler will generate one with empty body



Constructors

- The same name with the class.
- Have no return value

```
class Student
{
    private:
        // ...
    public:
        Student()
        {
            name[0] = 0;
            born = 0;
            male = false;
        }
        Student(const char * initName, int initBorn, bool isMale)
        {
            setName(initName);
            born = initBorn;
            male = isMale;
        }
};
```



Constructors

- The members can also be initialized as follows

```
Student(const char * initName): born(0), male(false)
{
    setName(initName);
}
```



Destructors

- The destructor will be invoked when the object is destroyed.
- Be formed from the class name preceded by a tilde (~)
- Have no return value, no parameters

```
class Student
{
    // ...
public:
    Student()
    {
        name = new char[1024]{0};
        born = 0;
        male = false;
        cout << "Constructor: Person()" << endl;
    }
    ~Student()
    {
        delete [] name;
    }
};
```

destructor.cpp



Destructors

```
Student * class1 = new Student[3]{  
    {"Tom", 2000, true},  
    {"Bob", 2001, true},  
    {"Amy", 2002, false},  
};
```

- What is the difference between the following two lines?

```
delete class1;
```

```
delete []class1;
```



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

this Pointer



Why is `this` needed?

- How does a member function know which `name`?

```
Student yu = Student{"Yu", 2000, true};  
Student amy = Student{"Amy", 2000, true};  
yu.setName("yu");  
amy.setName("Amy");
```

```
name: "Yu"  
born: 2000  
male: true
```

```
name: "Amy"  
born: 2001  
male: false
```

```
void setName(const char * s)  
{  
    strncpy(name, s, 1024);  
}
```



this Pointer

- All methods in a function have a `this` pointer.
- It is set to the address of the object that invokes the method.

```
void setBorn(int b)
{
    born = b;
}
```

```
void setBorn(int b)
{
    this->born = b;
}
```

```
void setBorn(int born)
{
    this->born = born;
}
```

`this.cpp`



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

const and static Members



const Variables

- Statements for constants

```
#define VALUE 100
```

```
const int value = 100;
```

```
const int * p_int;
```

```
int const * p_int;
```

```
int * const p_int;
```

```
void func(const int *);
```

```
void func(const int &);
```



const Members

- `const` member variables behavior similar with normal `const` variables
- `const` member functions promise not to modify member variables.

```
class Student
{
private:
    const int BMI = 24;
    // ...
public:
    Student()
    {
        BMI = 25; //can it be modified?
        // ...
    }
    int getBorn() const
    {
        born++; //Can it be modified?
        return born;
    }
};
```

`const.cpp`



static members

- static members are not bound to class instances.

```
class Student
{
    private:
        static size_t student_total; // declaration only
    public:
        Student()
        {
            student_total++;
        }
        ~Student()
        {
            student_total--;
        }
        static size_t getTotal() {return student_total;}
};
// definition it here
size_t Student::student_total = 0;
```

```
name: "Tom"
born: 2000
male: true
```

```
name: "Bob"
born: 2001
male: true
```

```
name: "Amy"
born: 2002
male: false
```

```
student_total: 3
```