

COMP 322 Lecture 6 - Classes in C++

Junji Duan

2024/2/9

C structs: data and code are separated

Today's Outline

- Review of OO concept
- C structs
- C++ Classes
- Constructors
- Destructors

```
15 struct person
16 {
17     int age;
18     char sex;
19 };
20
21 bool canVote(int age)
22 {
23     if (age >= 18)
24         return true;
25     return false;
26 }
27
28 // main function
29 int main()
30 {
31     person Mike;
32     Mike.age = 24;
33     Mike.sex = 'M';
34
35     if (canVote(Mike.age))
36         cout << "Mike is eligible to vote" << endl;
37     else
38         cout << "too bad for Mike" << endl;
39     return 0;
40 }
```

Structure
* data
* algorithm
object

should be reused in same structure

OO programming: Quick review

- Approach to design modular and reusable systems
- Programming is about manipulating data through code (methods or algorithms)
 - Object: is a coupling of both data and methods
- Extension to the concept of structures
 - Not only we group multiple data elements but we also attach the intelligence needed to manipulate the data (also called encapsulation)

C++ Classes: introduction

```
17 class Person
18 {
19 public:
20     bool canVote()
21     {
22         if (age >= 18)
23             return true;
24         return false;
25     }
26     int age;
27     char sex;
28 };
29
30 // main function
31 int main()
32 {
33     Person Mike;
34     Mike.age = 24;
35     Mike.sex = 'M';
36
37     if (Mike.canVote())
38         cout << "Mike is eligible to vote" << endl;
39     else
40         cout << "too bad for Mike" << endl;
41     return 0;
42 }
```

- Class is a user defined type
- It has data and methods
 - Referred to as members of the class
 - Methods are functions which are part of the class
- Members may have different access levels
 - public, private or protected
 - private by default if not specified
 - Public members are called interface of the class
- Instances of a class are called objects
 - Mike is an object of class Person

C++ Classes: access levels (public)

| | |
|---|---|
| <pre>17=class Person 18 { 19 public: 20 bool canVote() 21 { 22 if (age >= 18) 23 return true; 24 return false; 25 } 26 27 int age; 28 char sex; 29 }; 30 31 // main function 32 int main() 33 { 34 Person Mike; 35 Mike.age = 24; 36 Mike.sex = 'M'; 37 38 if (Mike.canVote()) 39 cout << "Mike is eligible to vote" << endl; 40 else 41 cout << "Too bad for Mike" << endl; 42 return 0; 43 }</pre> | <ul style="list-style-type: none">Access levels are also called access specifiers or access modifiersPublic<ul style="list-style-type: none">Accessible from anywhere (inside and outside the class)Public is not a default mode so you need to explicitly specify itPublic methods are the interface of the class<ul style="list-style-type: none">This is what the clients can use to manipulate the inner state of the object |
|---|---|

C++ Classes: access levels (private)

| | |
|---|--|
| <pre>17=class Person 18 { 19 public: 20 bool canVote() 21 { 22 if (age >= 18) 23 return true; 24 return false; 25 } 26 27 // private 28 int age; 29 char sex; 30 }; 31 32 // main function 33 int main() 34 { 35 Person Mike; 36 Mike.age = 24; // ERROR 37 Mike.sex = 'M'; // ERROR 38 39 if (Mike.canVote()) 40 cout << "Mike is eligible to vote" << endl; 41 else 42 cout << "Too bad for Mike" << endl; 43 return 0; 44 }</pre> | <ul style="list-style-type: none">Private<ul style="list-style-type: none">Accessible only from within the classPrivate is a default mode so you don't necessarily need to explicitly specify itFriend functions are allowed to access private members (we will discuss this later)Best practice is to define data as private and provide a public interface to access the dataUsed to achieve data hiding |
|---|--|

C++ Classes: access levels (private)

| | |
|--|---|
| <pre>17=class Person 18 { 19 public: 20 bool canVote() 21 { 22 if (age >= 18) 23 return true; 24 return false; 25 } 26 27 void setAge(int age) 28 { 29 this->age = age; 30 } 31 32 void setSex(char sex) 33 { 34 this->sex = sex; 35 } 36 37 private: 38 int age; 39 char sex; 40 }; 41 42 // main function 43 int main() 44 { 45 Person Mike; 46 Mike.setAge(24); 47 Mike.setSex('M'); 48 49 if (Mike.canVote()) 50 cout << "Mike is eligible to vote" << endl; 51 else 52 cout << "Too bad for Mike" << endl; 53 return 0; 54 }</pre> | <p>在C++中，this是一个关键字，表示当前对象的指针。它是一个隐式参数。对于类的非静态成员函数来说，this指向调用该成员函数的对象。使用this指针，你可以访问类的成员变量和成员函数。</p> <p>这里有几个使用this指针的常见场景：</p> <ol style="list-style-type: none">在成员函数中访问当前对象成员变量：当局部变量名与成员变量名冲突时，可以使用this来区分它们。 <pre>class Example { int value; public: void setValue(int value) { this->value = value; // 明确指定成员变量 } };</pre> |
|--|---|

C++ Classes: access levels (protected)

| | |
|---|--|
| <pre>17=class Person 18 { 19 public: 20 bool canVote() 21 { 22 if (age >= 18) 23 return true; 24 return false; 25 } 26 27 void setAge(int age) 28 { 29 this->age = age; 30 } 31 32 void setSex(char sex) 33 { 34 this->sex = sex; 35 } 36 37 protected: 38 int age; 39 char sex; 40 }; 41 42 // main function 43 int main() 44 { 45 Person Mike(24, 'M'); 46 cout << Mike.getAge() << endl; 47 }</pre> | <ul style="list-style-type: none">Protected<ul style="list-style-type: none">Accessible from within the classAccessible also from derived classes (we will discuss this later)Protected is not a default mode so you need to explicitly specify itFriend functions are allowed to access protected members (we will discuss this later)Used to achieve data hiding |
|---|--|

C++ Classes: method definition

| | |
|---|---|
| <pre>17=class Person 18 { 19 public: 20 bool canVote() 21 { 22 if (age >= 18) 23 return true; 24 return false; 25 } 26 27 void setAge(int age) 28 { 29 this->age = age; 30 } 31 32 void setSex(char sex) 33 { 34 this->sex = sex; 35 } 36 37 protected: 38 int age; 39 char sex; 40 }; 41 42 // main function 43 int main() 44 { 45 Person Mike(24, 'M'); 46 cout << Mike.getAge() << endl; 47 }</pre> | <ul style="list-style-type: none">Member methods can be defined within the class definition or outside of the classMethods defined inside the class declaration are considered "inline" methods even without the use of the "inline" keyword |
|---|---|

| | |
|---|--|
| <pre>17=class Person 18 { 19 public: 20 bool canVote(); 21 void setAge(int age); 22 void setSex(char sex); 23 protected: 24 int age; 25 char sex; 26 }; 27 28 bool Person::canVote() 29 { 30 if (age >= 18) 31 return true; 32 return false; 33 } 34 35 void Person::setAge(int age) 36 { 37 this->age = age; 38 } 39 40 void Person::setSex(char sex) 41 { 42 this->sex = sex; 43 }</pre> | <ul style="list-style-type: none">To define a method outside of the class, we need to declare it within the class, then we provide the implementation outside of the class using the scope operator ::Methods defined outside of the class declaration can still be declared "inline" but with the explicit use of the "inline" keyword |
|---|--|

C++ classes: constructors

- When instantiating a class, a special method is implicitly called first
 - This method is the constructor
- Every class has a constructor (at least one)
- If a constructor is not provided by the programmer, the compiler will provide a default implicit constructor (that does basically nothing)
 - This is how the construction of the Person class from the previous example was possible
- The Constructor method:
 - is used to initialize the data members of a class
 - must have the same name as the class
 - must be declared public in general
 - There are some exceptions when implementing advanced design patterns
 - does not have a return type
 - Constructors don't return values

| | |
|--|---|
| <pre>17=class Person 18 { 19 public: 20 Person(); 21 Person(int age, char sex); 22 int getAge(); 23 protected: 24 int age; 25 char sex; 26 }; 27 28 Person::Person() 29 { 30 this->age = 0; 31 this->sex = 'U'; 32 } 33 34 Person::Person(int age, char sex) 35 { 36 this->age = age; 37 this->sex = sex; 38 }</pre> | <ul style="list-style-type: none">Constructor can be personalized using parametersUser can define as many different constructors as needed |
|--|---|

Constructor: initialization list

```
class Person {
public:
    Person();
    Person(int age, char sex);
    int getAge() {return age;};
protected:
    int age;
    char sex;
};

Person::Person():age(0), sex('U')
{
}

Person::Person(int age, char sex):age(age), sex(sex)
{
}
```

- Initialization is listed outside of the body of a constructor
- Initialization list is preferred to regular initialization because it yields better performance

Public, Private and Protected

In C++, public, private, and protected are three types of access specifiers used to set the access level for class members. Below is an explanation of each access specifier and its usage through an example.

- public
 - Public members can be accessed anywhere, including outside the class. This means that if a class member is declared as public, any external code can access it directly.
- private
 - Private members can only be accessed by member functions of the same class. This means that if you attempt to access a private member from outside the class, the compiler will throw an error.
- protected
 - Protected members are similar to private members in that they restrict external access. However, unlike private, protected members are accessible in derived classes (subclasses).

Suppose we have a class named Person with three members: name, age, and height. We want name to be accessible by anyone, height to be accessible only within the class or its subclasses, and age to be accessible only within the class.

```
class Person {
public:
    string name; // 公有成员，任何地方都能访问

protected:
    int height; // 保护成员，只能在类内部及其子类中访问

private:
    int age; // 私有成员，只能在类内部访问

public:
    Person(string n, int h, int a) : name(n), height(h), age(a) {}

    void printInfo() {
        cout << "姓名: " << name << ", 身高: " << height << ", 年龄: " << age << endl;
    }
};

class Student : public Person {
public:
    Student(string n, int h, int a) : Person(n, h, a) {}

    void printHeight() {
        cout << "学生身高: " << height << endl; // 可以访问protected成员
    }
};
```

```
int main() {
    Person p("张三", 170, 30);
    p.printInfo();
    // cout << p.age; // 错误：age是私有成员，不能在类外访问
    cout << "姓名: " << p.name << endl; // 正确：name是公有成员

    Student s("李四", 160, 20);
    s.printHeight(); // 正确：派生类可以访问protected成员
    // cout << s.height; // 错误：height在派生类外部不可访问
}
```

In this example, name is a public member, so it can be accessed directly in the main function. height is a protected member, so it can be accessed in the member function printHeight of the Student class (derived from Person), but not directly in the main function. Finally, age is a private member, so it cannot be accessed directly in the main function or in the Student class.

Struct, Class

In C++, struct and class are both composite data types used for data encapsulation, with the main difference lying in their default access level and default inheritance mode. In C++, types defined using struct or class can contain data members (member variables) and function members (member functions or methods), thereby encapsulating data and behavior.

struct

- Default access level: The members of a struct are public by default, meaning that by default, members of a struct can be accessed from outside the struct.
- Usage: Typically used for smaller data structures, more focused on storing data rather than emphasizing behavior.

```

struct Person {
    std::string name;
    int age;

    void printInfo() const {
        std::cout << "Name: " << name << ", Age: " << age << std::endl;
    }
};

int main() {
    Person p1;
    p1.name = "张三";
    p1.age = 30;
    p1.printInfo(); // 直接访问并使用Person的数据成员和成员函数

    return 0;
}

```

class

- Default access level: The members of a class are private by default, meaning that by default, members of a class cannot be accessed from outside the class, ensuring encapsulation and data safety.
- Usage: Usually used to define complex data types containing both data and function members, more focused on the behavior and interface of objects.

```

class Animal {
private:
    std::string name;
    int age;

public:
    Animal(const std::string& n, int a) : name(n), age(a) {}

    void printInfo() const {
        std::cout << "Name: " << name << ", Age: " << age << std::endl;
    }
};

int main() {
    Animal a1("小白", 5);
    a1.printInfo(); // 使用Animal的公有成员函数

    return 0;
}

```

Summary

- The essential difference between struct and class lies in their default access level (struct is public, class is private).
- In C++, both struct and class can be used to define types containing data members and member functions.
- The choice between struct and class often depends on your requirements for the default access level of the type and how you intend to use it (more emphasis on data or behavior).

P.S. Constructor and Destructor notes are in Lecture 7.