CS2310 Computer Programming

LT06: Class and Object

Computer Science, City University of Hong Kong (Dongguan)

Semester A 2025-26

Outline

- Introduction to class

- Class definition
- Public and private members
- Constructor and destructor

- String class

Class and Object : Example (circle)

Without class/object

```
int radius;
double getCircleArea()
{
 return 3.14*radius*radius;
}
double getCirclePerimeter()
{
  return  2*3.14*radius;
}
```

With class/object

```
class Circle {
   public:
   int radius;
   double getCircleArea()
   {
        return
          3.14*radius*radius;
   }
   double getCirclePerimeter()
   {
        return 2*3.14*radius;
   }
};
```

Class and Object

```
void main(){
  cout << "Please enter the radius of circle";
  cin >> radius;
  cout << getCircleArea();
}
```

Without class/object

```
void main(){
  Circle c;//Circle is a class, c is an object of Circle
  cout << "Please enter the radius of circle";
  cin >> c.radius;
  cout << c.getCircleArea();
}
```

With class/object

Class and Object : Example (Rect)

Without class/object

```
int width, height;
double getRectangleArea()
{
   return  width*height;
}
double getRectanglePerimeter()
{
   return  2*(width+height);
}
```

With class/object

```
class Rect{
   public:
   int width, height;
   double getRectangleArea()
   {
         return
         width*height;
   }
   double
   getRectanglePerimeter()
   {
         return
         2*(width+height);
   }
};
```

Class and Object

```
void main()
{
   cout << "Please enter the width and height of a
   rectangle";
   cin >> width >> height;
   cout << getRectangleArea();
}
```

Without class/object

```
void main(){
   Rect r; //Rect is a class, r is an object of Rect
   cout << "Please enter the width and height of a
   rectangle";
   cin >> r.width >> r.height;
   cout << r.getRectangleArea();
}
```

Thinking: Size?

Code: lec06-01-classSize.cpp

With class/object

Class and Object

- **Class** and **object** are important features of **Object-oriented** Programming Language (like C++, Java, C#)
- With **class**, variables and their directly related functions can be grouped together to form a new data type
- It promotes reusability and object-oriented design

- **Object** is an instance of class, i.e. *class* is a blue-print and its product is its *object*

Class in Computer Programming

- An abstract view of real-world objects, e.g. car, horse
- Computer program is a model of real-world problem
- **Simple** problem: program with **variables** and **functions**
- **Large** scale program: **class** and **object**
- Class:
  - Definition of program **component**
  - Consists of member variables and member functions
    - Member variable : **variable** belong to class
    - Member function: **function** primary designed to **access/manipulate** the **member variable** of the class
- Object:
  - An instance of class / runtime representation of a class

Class in programming

Class:Robot

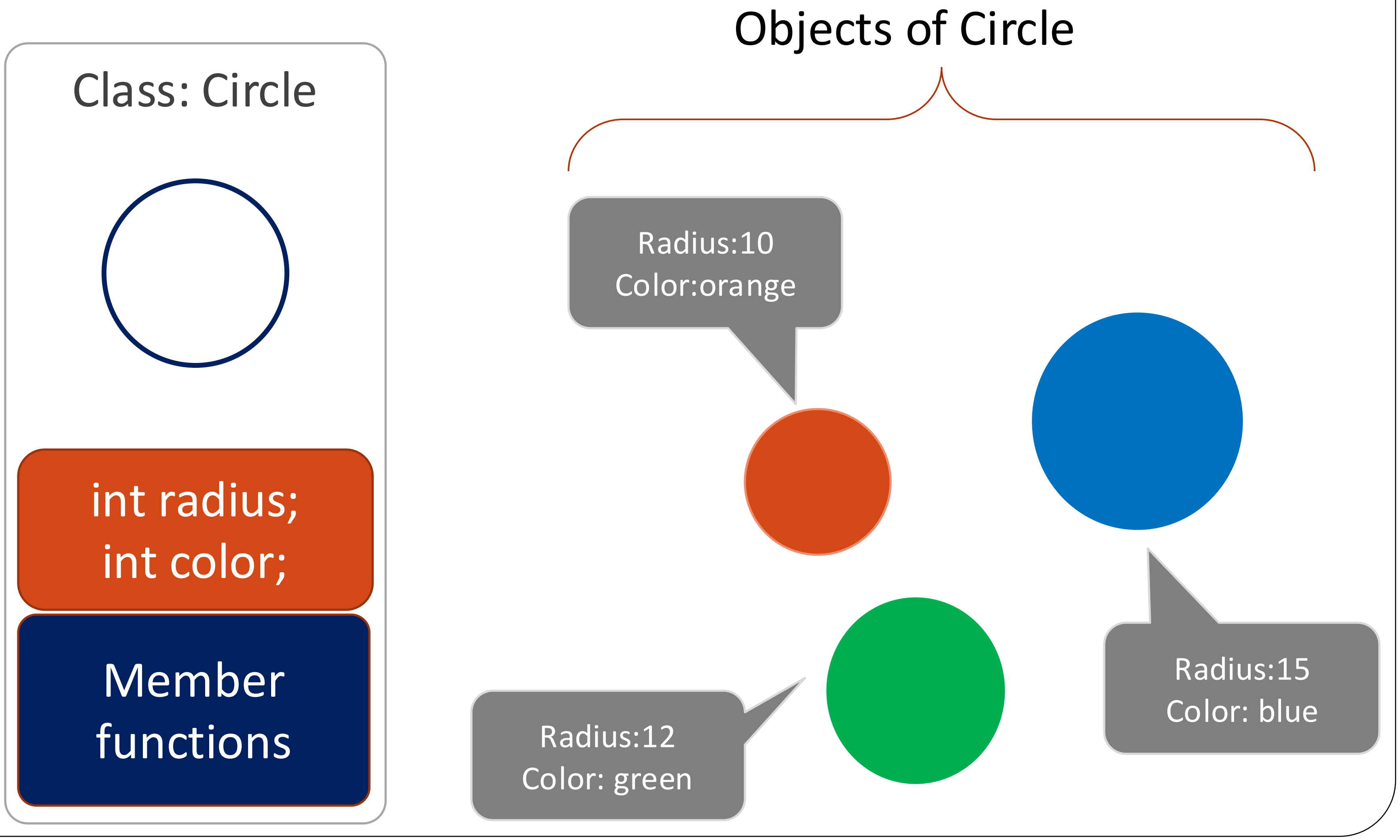
```
void start();
void shutdown();
void moveForward(int step);
void turnLeft(int degree);
void turnRight(int degree);
void takePhoto();
……………………………………
```

```
int modelNum;
int width;
int height;
int powerLevel;
……………………
```

Member variables

Member functions

What is an object?



Classes and Objects in C++

- A **class** is a **data type**, **objects** are **variables of this type** (e.g., Circle c; => int x;)
- An object is a **variable** with member functions and data values
- `cin`, `cout` are objects defined in header `<iostream> (cin.get(), cin.getline())`
- C++ has great facilities for you to define your own class and objects

Outline

- Introduction to class

- Class definition
- Public and private members
- Constructor and destructor

- String class

Defining classes

```
class class_name
{
        public / protected / private:
        variable declaration; //member variable
         return_type function ()  //member function
        {
                function body statement;
        }
};
```

Defining classes

```
class class_name
{
        public / protected / private:
        variable declaration; //member variable
        function prototype; //member function
};
return_type class_name::function() {
        method body statement;
}
```

Code: lec06-02-definition.cpp

Member function – Declare and Define

- In C++, a class definition commonly contains only the prototypes of its member functions (except for *inline functions*)
- Use classname::**functionName** to define the member function (method) of particular class

```
class Circle
{
   ……
   int radius;
   ……
   double getArea();
};
double Circle::getArea(){
   return 3.1415*radius*radius;
}
```

Defining classes (example I)

```
#include <iostream>
using namespace std;
class DayOfYear
{
public:
   int month;
   int day;
   void output(){
        cout << "month =" << month;
        cout << ", day =" << day << endl;
   }
};
```

Member variables

Member function

Code: lec06-03-dayofYear.cpp

Defining classes (example II)

```
#include <iostream>
using namespace std;
class DayOfYear
{
public:
  void output(); //member func. prototype
  int month;
  int day;
};
void DayOfYear::output()
{
  cout << "month =" << month
       << ", day =" << day << endl;
}
```

Define the method elsewhere

Main function

```
void main()
{
   DayofYear today, birthday;
   cin >> today.month >> today.day;
   cin >> birthday.month >> birthday.day;
   cout << "Today's date is: ";
   today.output();
   cout << "Your birthday is: ";
   birthday.output();
   if (today.month == birthday.month
       && today.day == birthday.day)
       cout << "Happy Birthday!\n";
}
```

Create object and access its member function

- To declare an object of a class

  Class_name *object_name;*

Examples:

```
Circle c1,c2;
DayofYear today;
```

- A member function of an object is called using the dot operator:
  - `today.output();`
  - `c1.getArea();`

Create pointer and access its function

- We can also declare the pointer of a class

  Class_name pointer_name;*

Examples:

```
Circle *cPtr1, *cPtr2;
DayofYear *today;
```

- A member function of the pointer is called using the arrow operator (->):
  - `today->output();`
  - `cPtr1->getArea();`

Outline

- Introduction to class
- Class definition
- Public and private members
- Constructor and destructor
- String class

Public and private members

- By default, all members of a class are private
  - By default, all members of a structure are **public**

- You can declare public members using the keyword `public`

- Private members can be accessed only by **member functions** (and *friend* functions) of that class, i.e. only from within the class, not from **outside** (e.g., main function).

Code: lec06-04-dayofYearPrivate.cpp

Private Variable and Access functions

- Member functions that give you access to the values of the private member variables are called ***access functions***, e.g., `get_month()`, `set(…)`

- Useful for controlling access to **private members**:
  - E.g. Provide **data validation** to ensure data integrity

- Needed when testing equality of 2 objects. (The predefined **equality operator = =** does **not work** for objects and variables of structure type.), e.g. **obj1==obj2** (**not work!**)

Code: lec06-05-equal.cpp

Why private variable?

- Prevent others from accessing the variables directly, i.e. variables can be **only** accessed by **access functions**.

```
class DayOfYear
{
  .........
  private:
       int month;
       int day;
  .........
};
```

```
void DayOfYear::set(int new_m, int
   new_d)
{
   ......
     month = new_m;
     day   = mew_d;
   ......
}
int DayOfYear::get_month()
{
   return month;
}

int DayOfYear::get_day()
{
   return day;
}
```

Why private variable?

- Change of the internal presentation, e.g. **variable name**, **type**, will not affect the how the others access the object. **Caller still calling the same function with same parameters**

```
class DayOfYear
{
  ………
  private:
        int month;
        int d;//day
  ………
};
```

```
void DayOfYear::set(int new_m, int new_d)
{
   ……
     month = new_m;
     d = new_d;
   ……
}
int DayOfYear::get_month()
{
   return month;
}
int DayOfYear::get_day()
{
   return d;
}
```

A new main program

```
void main()
{ DayOfYear today, birthday;

  today.input();
  birthday.input();
  cout << "Today's date is:\n";
  today.output();
  cout << "Your birthday is:\n";
  birthday.output();

  if (today.get_month()==birthday.get_month()
                                      &&
      today.get_day() == birthday.get_day())
    cout << "Happy Birthday!\n";
}
```

Private or public members?

- The common style of class definitions
  - To have all member **variables** **private**
  - Provide enough **access functions** to get and set the member variables
  - **Supporting functions** used by the **member functions** should also be made **private** (e.g., isValid(…))
  - Only functions that need to **interact** with the **outside** can be made **public**

Friend function

- In some cases, we want to access the private members of a class through functions outside the class directly

- Instead of changing the private members into public, we can declare functions with keyword `friend`

- A friend function is a normal function defined outside the class, while we declare it inside the class with keyword `friend`

Code: lec06-06-friend.cpp

Friend function

```
class DayOfYear
{

friend void printDate(DayOfYear *d);

public:
  set(int m, int d){
    month = m;
    day = d
  }

private:
  int month;
  int day;
};
```

Friend function

- We define and use the friend function like normal function outside the class

```
int main(){
  DayOfYear day;
  day.set(1, 1);
  printDate(&day);
  return 0;
}
void printDate(DayOfYear *d){
  cout<<"The month is "<< d->month <<endl;
  cout<<"The day is "<< d->day <<endl;
}
```

Assignment operator for objects

- It is **legal** to use assignment operator = with objects

- E.g. `DayOfYear due_date, tomorrow;`

```
    tomorrow.input();
    due_date = tomorrow;
```

Outline

- Introduction to class

- Class definition
- Public and private members
- Constructor and destructor

- String class

Constructors for initialization

- Class contains variables and functions

- Variables must be initialized before use

- In C++, a **constructor** is designed to initialize variables

- A **constructor** is a member **function** that is automatically called when an object of that class is declared

```
DayOfYear due_date;
```

- Special rules:
  - A constructor must have the same name as the class
  - A constructor definition cannot return a value

Default constructors

- A constructor with no parameters
- Will be called when no argument is given
- Mainly for initializing variables (to 0) and set pointers to NULL.

```
class Circle{
   int radius;
   Circle();
   double getArea();
};
Circle:: Circle(){
   radius=0;
}
double Circle::getArea(){
   return 3.1415*radius;
}
```

```
void main(){
   Circle circle;
   circle.getArea();
}
```

Code: lec06-07-constructor.cpp

Default constructors

- Default constructor will be generated by compiler automatically if NO constructor is defined so far
- However, if a non-default constructor is defined, the compiler will not generate its default constructor

```
class Circle{
  int radius;
  Circle(int r);
  double getArea();
};
Circle::Circle(int r){
  radius=r;
}
double Circle::getArea(){
  return 3.1415*radius;
}
```

```
void main(){
  Circle circle2; //illegal
  Circle circle(6); //OK
}
```

Example: Bank account

- E.g., Suppose we want to define a bank account class which has member variables `balance` and `interest_rate`. We want to have a constructor that initializes the member variables.

```
class BankAcc
{
public:
   BankAcc(int dollars, int cents, double rate);
   ...
private:
   double balance;
   double interest_rate;
};
...
BankAcc::BankAcc(int dollars, int cents, double rate)
{
    balance = dollars + 0.01*cents;
    interest_rate = rate;
}
```

Constructors

- When declaring objects of `BankAcc` class:

```
    BankAcc account1(10,50,2.0),
            account2(500,0,4.5);
```

```
 BankAcc(int dollars, int cents, double rate);
```

- Note: A constructor **cannot** be called in the same way as an ordinary **member function** is called:

```
 account1.BankAcc(10,20,1.0); // illegal
```

Constructors

- **More than one** versions of **constructors** are usually defined (overloaded) so that objects can be **initialized** in **more than one way**, e.g.

```
class BankAcc
{
public:
   BankAcc(int dollars, int cents, double rate);
   BankAcc(int dollars, double rate);
   BankAcc();
   ...
private:
   double balance;
   double interest_rate;
};
```

Constructors

```
BankAcc::BankAcc(int dollars, int cents, double rate)
{
   balance = dollars + 0.01*cents;
   interest_rate = rate;
}
BankAcc::BankAcc(int dollars, double rate)
{
   balance = dollars;
   interest_rate = rate;
}
BankAcc::BankAcc()
{
   balance = 0;
   interest_rate = 0.0;
}
```

Constructors

- When the constructor has no arguments, DO NOT include any parentheses in the object declaration.
- E.g.

```
BankAcc acc1(100, 50, 2.0), // OK
        acc2(100, 2.3),     // OK
        acc3(),          // error
        acc4;          // correct
```

- The compiler thinks that it is the prototype of a **function** called `acc3` that takes no arguments and returns a value of type `BankAcc`

Destructors

- Automatically invoked when an object is destroyed
- Has the same name as the class name preceded with a tilde (~) symbol
- **No** return value and **no** arguments
- **Cannot** have **more than one** version

```
class DayOfYear {
  public:
       DayOfYear()    {} // constructor
       ~DayOfYear()   {} // destructor

};
```

Code: lec06-08-destructor.cpp

Object as a Member Variable

- Member object

```
class AClass {};
class BClass {
   AClass a;
};
```

- Whose constructor/destructor is called first?

```
class AClass {
public:
   AClass()        { cout << "A-Construction" << endl;}
   ~AClass()       { cout << "A-Destruction" << endl;}
};

class BClass {
public:
   BClass()        { cout << "B-Construction" << endl;}
   ~BClass()       { cout << "B-Destruction" << endl;}

   AClass a;
};
```

Code: lec06-09-objectMember.cpp

Initializer List

- Initialize member variables of a class

```
class DayOfYear {
  public:
       DayOfYear(int d, int m):m_Day(d),m_Month(m) {}
       int m_Day;
       int m_Month;
};
```

- It is a preferred way to initialize member variables
  - More efficient
  - Sometime we have to use it
    - For constant member variables
    - For member objects that do not have a default constructor

Code: lec06-10-initializer.cpp

Initializer List

- Example

```
class AClass {
   int x;
   AClass(int i)  { x = i;}
};

class BClass {
   AClass a;
   const int y;

   BClass(int j): a(j), y(10) {}
};
```

Static Members

- Static member variable
  - Shared by all the objects
  - Memory is allocated during compilation
  - Declared in the class, but initialized outside the class

```
class Circle {
public:
   static int cnt_pub;
private:
   static int cnt_pri;
};

int Circle::cnt_pub = 0;
int Circle::cnt_pri = 0;
```

```
int main() {
   Circle c1, c2;
   c1.cnt_pub++;
   c2.cnt_pub++;

   cout << Circle::cnt_pub << endl;

   // not allowed:
   cout << Circle::cnt_pri << endl;
}
```

Code: lec06-11-static.cpp

Static Members

- Static member function
  - It can only access static member variables

```
class Circle {
public:
   static void print(int x) {
         cnt_pub = x;
         cout << cnt_pub << endl;
   }
   static int cnt_pub;
   int radius;
};

int Circle::pub = 0;

Circle c1;
c1.print(10);
Circle::print(100);
```

Outline

- Introduction to class

- Class definition
- Public and private members
- Constructor and destructor

- String class

C++ string class

- It is a class
  - `#include <string>`
- Many useful functions and operators
  - search, replacement, compare, substring, etc.
  - assign, concatenate, etc.

Create String Objects

- Different constructors
  - string();
  - string(a string constant);
  - string(another string object);
  - string(int n, char c);

```
#include <iostream>
#include <string>
…
string str1;
string str2(“cs2310”);
string str3(str2);
string str4(5, ‘a’); // str4 stores “aaaaa”
```

Code: lec06-12-string.cpp

Assignment

- Use assignment operator `=`
  - `string str5(“cs2310”);`
  - `string str6 = str5;`
- Use `assign()` function
  - `string str7.assign(str6); // “cs2310”`
- Use `assign(str, pos, n)` function, which gets `n` characters from `str`, starting at `pos`
  - `string str7;`
  - `str7.assign(str6, 2, 4); // str4 stores “2310”`

Concatenation

- Use operator +=
  - `string str8 = “cs”; str8 += “2310”;`
- Use `append()` function
  - `str8.append(“-computer”);`
- Use `append(str, pos, n)` function, which appends `n` characters from `str`, starting at `pos`
  - `str8.append(“-programming-language”, 0, 12);`

Access Character(s)

- Function `.at(index)`
  - returns the character at position `index`
  - can throw an exception if index is out of range
  - `string str9 = “cs2310”; cout << str9.at(0) << endl;`
- [] has no range checking
  - `cout << str9[1] << endl;`

Find and Replace

- `find/rfind` function
  - `find` function searches from **beginning to end** of the string
  - `rfind` function searches from **end to beginning** of the string
  - if found, the index is returned
  - if not found, -1 is returned
    - `string str10 = “cs2310cs”;`
    - `cout << str10.find(“cs”) << endl;     // 0`
    - `cout << str10.rfind(“cs”) << endl;    // 6`
- replace function
  - `str11.replace(pos, n, str2);`
    - `pos`: starting from position `pos` of `str1`
    - `n`: number of characters to be replaced
    - `str`: replaced by `str2`

Insert and Delete

- `str12.insert(pos, str2);`
  - insert `str2` from position `pos` of `str1`
- `str12.erase(pos);`
  - delete all characters from `pos` to end of `str1`
- `str12.erase(pos, n);`
  - delete `n` characters from `pos` of `str1`

Substring

- `str.substr(pos, n);`
  - `string str13 = "cs2310";`
  - `string str14 = str13.substr(2, 4); // "2310"`
- Example: extract the username from an email address, e.g., abc@cityu-dg.edu.cn
  - **Step 1**: find the position of '@'
  - **Step 2**: use `substr()` to get the username part

Compare

- May use this function for sorting
- `str16.compare(str17);// str17:”aab”`
- Return **three** possible values:
  - 0: when they are **equal**
  - < 0: either the value of the first character that does not match is smaller in `str16`, or all compared characters match but the `str16` string is shorter
  - > 0: either the value of the first character that does not match is greater in `str16`, or all compared characters match but the `str16` string is longer
  - E.g.,
    - `str16` is `aac`, `aaa`, `aabc`, `aa`, `aab`, etc.