Overview

- Classes
- File Organization
- Class Members
    - Non-static Member Functions
    - Static Class Members
- Constructors
    - Default Constructor
    - Copy Constructor
- Copy Assignment Operator
- Destructor

Classes

What are classes?

- group data and functionality
- formal description of objects
- corresponds to the real world

Why do we need classes?

- hide complexity
- makes reusable code
- helps to understand code better
- imitate the real world

Classes in C++

Classes are basis of object oriented programming

```
class Car{
private:
 int horsepower;
 int weight;
 string brand;
public:
 void setHorsepower(int horsepower);
 void setWeight(int weight);
 void setBrand(string brand);
 static int numberOfCars; // static
};
```

Objects are instances of classes

```
Car bmw;
bmw.horsepower = 100; // Error: is private
bmw.setHorsepower(100);
string description = bmw.description();
```

Classes in C++

Definition and implementation of classes is often split into separate files.

- Definition of class `Car` in header file `Car.h`
- Implementation in file `Car.cpp`, has to include the header file with `#include "Car.h"`
- Must be included by every other file that wants to access members `#include "Car.h"`
- If the class declaration changes, all files that include `Car.h` must be recompiled

Header

```
#ifndef _RECTANGLE_H
#define _RECTANGLE_H

class Rectangle{
private:
 int _x, _y;
protected:
 int _width, _height;
public:
 void setCoordinates(int x, int y);
 void setSize(int width, int height);
 int computeArea();
};

#endif
```

Include guards to prevent compiler errors

- Compilation fails if a class is defined multiple times (including a header file twice)
- Use `#ifndef` and `#define` to prevent this

Header

```
#ifndef _RECTANGLE_H
#define _RECTANGLE_H

class Rectangle{
private:
 int _x, _y;
protected:
 int _width, _height;
public:
 void setCoordinates(int x, int y);
 void setSize(int width, int height);
 int computeArea();
};

#endif
```

Access specifier

- private: from within the class and friends
- protected: from within the class, friend and derived class

- public: from everywhere

Implementation

- Implementation file (e.g. Rectangle.cpp) contains the corresponding member definitions
  - Must include `Rectangle.h`

- Any change to a member function that is defined in a class declaration may cause a lot of recompilations

Implementation

- Use scope operator `::` to define members of the class from outside of the class defintion
- Use default value for method arguments (called default arguments)

```
void Rectangle::setCoordinates(int x = 0, int y = 0){
 _x = x;
 _y = y;
}

void Rectangle::setSize(int width = 0, int height = 0){
 _width = width;
 _height = height;
}

int Rectangle::computeArea(){
 return _width*_height;
}
```

Objects in C++

```
#include <iostream>
#include "Rectangle.h"
using namespace std;

int main(){
 Rectangle rectangle;
 rectangle.setSize(5,10);
 cout << rectangle.computeArea();

 rectangle.setSize(); // correct, using default arguments (0,0)
 rectangle.setSize(5); // correct, using arguments (5,0)

 return 0;
}
```

Members (Variables, Functions)

```
class Rectangle{
private:
 int _x, _y; // Member variables

protected:
 int _width, _height; // Member variables

public:
 // Member functions
 void setCoordinates(int x, int y);
 void setSize(int width, int height);
 int computeArea();

 void print() const; // Member const functions
 static int defaultColor; // Static (class) variable
 static Rectangle add(Rectangle r1, Rectangle r2); // Static function
};
```

Members (Variables, Functions)

- Non-Static members
  - are bound to objects
  - exists for every object
  - member function declared `const` can't change object
- Static members (called class variables/functions)
  - exist only once per class
  - kind of global variables/functions

```
int Rectangle::defaultColor = 255;
void Rectangle::print() const{
 ...
}
```

Constructor

The constructor

- is a special method declared in a class
- has the same name as the class
- has no return type
- is called when a new object is created

There can be multiple constructor

```
class Rectangle{
private:
 ...
protected:
 ...
public:
 Rectangle();
 Rectangle(x,y);
};
```

```
Rectangle::Rectangle(){
 _x = 0;
 _y = 0;
}

Rectangle::Rectangle(x,y){
 _x = x;
 _y = y;
}
```

Constructor

```
#include "Rectangle.h"
...
Rectangle rectangle; // Calls Rectangle();
Rectangle rectangle(); // Compiler error
Rectangle rectangle(5,5); // Calls Rectangle(int x, int y);
```

Only the explicitly (user defined) defined constructors can be called.

If class has no constructor, the compiler implicitly generates a

- default constructor (constructor with no arguments)
- copy constructor
- copy assignment operator (covered later in operator overloading)

Initialization List

```
class Rectangle{
 int _x;
 int _y;
public:
 Rectangle();
 Rectangle(x,y);
};
```

```
Rectangle::Rectangle(x,y) : _x(x), _y(y){
 // executed after data members have been initialized
 ...
}
```

Base initializers are only allowed in constructors

Copy Constructor

- An object duplicate is created with the copy constructor of the class
- The copy constructor of a class can be called with a reference to an object of this class as argument
  - Is called in variable initializations (!= assignment)
  - Is called when passing objects as function arguments
  - Is called when returning objects as function results

- If a class has no copy constructor, an implicit copy constructor is automatically generated
  - Calls the copy constructors of all non-static data members

Copy Constructor

- Is a special kind of constructor
- Used to make a copy of an object

It's up to the programmer how to implement it!

```
class Rectangle{
private:
 ...
virtual:
 ...
public:
 Rectangle(const Rectangle
&rect){
  _x = rect._x; // private
member
  _y = rect._y;
  ...
 }
};
```

```
// x, y, width, height
Rectangle rect1(0,0,10,40);

// Examples of copy
constructor
Rectangle rect2(rect1);
Rectangle rect3 = rect1;
foo(rect1); // Argument
return rect1; // Return
value
```

In this example the copy constructor is implemented directly in the header file. Functions need not to be implemented in the .cpp file, but may increase the readability/maintainability of your project.
Note that object assignment operation only calls the copy constructor, if the object is not created yet. Otherwise the copy assignment operator is used.

Constructor - Implicit conversion

Constructors

- may be called in unexpected situations
- are implicitly called to perform type conversions

```
class Rectangle{
public:
 // Constructor
 Rectangle(int x = 0, int y = 0, int width = 0, int height = 0);

 // Returns intersection
 Rectangle intersection(Rectangle rect);
};
```

```
Rectangle rect;          // calls Rectangle(0,0,0,0)
Rectangle rect = 1;      // calls Rectangle(1,0,0,0)
rect.intersection(5);    // calls Rectangle(5,0,0,0)
```

Constructor - Prevent implicit conversion

- Implicit constructor calls may be unwanted
- With keyword `explicit` unexpected constructor calls can be avoided

```
class Rectangle{
public:
 explicit Rectangle(int x = 0, int y = 0, int width = 0, int height =
0);
 Rectangle intersection(Rectangle rect);
};
```

```
Rectangle rect;          // calls Rectangle(0,0,0,0)
Rectangle rect = 1;      // Compiler error
Rectangle rect(1);       // calls Rectangle(1,0,0,0)
rect.intersection(5);    // Compiler error
rect.intersection(static_cast<Rectangle>5);     // calls
Rectangle(5,0,0,0)
```

Copy Assignment Operator

- Object assignment is performed with the copy assignment operator
  - Object assignment can be controlled by the programmer

- The copy assignment operator of a class can be called with a reference to an object of this class argument
  - Is called in object assigments (not in object initializations!)

- If a class has no copy assignment operator, an implicit copy assignment operator is automatically generated
  - Calls the copy assignment operators of all non-static data members

Copy Assignment Operator

```
class Rectangle{
public:
 Rectangle& operator=(const Rectangle& rect){
  _x = rect._x;
  _y = rect._y;
  return *this;
 }
};
```

```
Rectangle rect1(0,0);
Rectangle rect2(rect1); // Copy constructor
rect2 = rect1; // Copy assignment operator
```

Destructor

The destructor

- is a special method declared in a class
- has the same name as the class but preceded with a tilde sign (~)
- has no return type and no arguments
- is called when an object is destroyed

There can only be one destructor per class

```
class Rectangle{
private:
 ...
protected:
 ...
public:
 Rectangle();
 ~Rectangle();
};
```

Destructor

Object destruction can be controlled by the programmer

- Destructor of a class can be called without arguments
- Is called on local variable when declaration scope is left
- Is called on dynamically allocated objects when `delete` is called
- Is called on statically allocated objects when program is terminated
- Is called on every array element, if an array is destroyed

If a class has no destructor, an implicit destructor is generated

- Calls the destructors of all non-static data members

Summary - Classes

```
#ifndef _RECTANGLE_H
#define _RECTANGLE_H // Guard to prevent compiler error

class Rectangle{
friend class Circle; // friend

private: // private access specifier
 int _x, _y;

protected: // protected access specifier
 int _width, _height;

public: // public access specifier
 Rectangle();
 Rectangle(const Rectangle& rect);
 Rectangle& operator=(const Rectangle& rect); // copy assignment
operator
 ~Rectangle();

 void print() const; // const method

 static int defaultColor; // static/class variable
```

```
 static Rectangle add(Rectangle r1, Rectangle r2); // static/class
function
};
#endif
```

Summary - Classes

```
Rectangle rect1;
Rectangle rect2(rect1);
rect2 = rect1;
```

Summary - Non-Static Member Functions

A non-static member function is "bound to" a particular object

- Every object of a class has an instance of this function attached

The function operates "within" the object to which it is bound

- It can access all non-static data members of the object without object qualification
- A member function declared as `const` does not change the object

- Constructors/destructors/copy assignment operators are special cases of non-static member functions

Summary - Static Members

Static member variables (class variables)

- Exist only once per class (the same for all objects)
- Kind of global variables

Static member functions (class methods)

- Kind of global functions
- Can only refer to static data

Summary - Members Example

```
class Rectangle{
private:
 int _x, _y;
protected:
 int _width, _height;
public:
 void setCoordinates(int x, int y);
 void setSize(int width, int height);
 int computeArea();

 void print() const;
 static int defaultColor;
 static Rectangle add(Rectangle r1, Rectangle r2);
};
```

```
Rectangle rect1; Rectangle rect2;
rect1._x = 10; // Error: member variable is private
Rectangle::defaultColor = 255;
Rectangle unionRect = Rectangle::add(rect1, rect2);
```