Introduction to Classes

COS110: Chapter 13
Dr. Mardé Helbig
Procedural Programming

• Programming languages like C, Pascal are procedural programming languages
• Data is stored in a collection of variables and/or structures
• Data is coupled with functions that perform operations on the data
• Data and functions are separate entities
Imagine you have to write code to determine the profit or loss of a department.

You declare variables for income and expenses. You define functions to calculate the profit.

How will you do this?
Procedural Programming

No imagine you have to write code to determine the profit or loss of a department, but suddenly it is calculated in a different way. Management decides they want a break-down of expenses into running costs, training and salaries.

You have declare variables for income and expenses. You have define functions to calculate the profit using these two variables.

So how will you change this?
Procedural Programming

This shows the major problem with procedural programming:

If the program’s specifications change, data structures must be re-designed. But when the structure of data changes, we must also change the code that operates on that data.

This lead to a shift towards object-oriented programming
Object-Oriented Programming

- Procedural programming is centered on creating functions
- Object-oriented programming is centered on creating objects
- But what is an object?
- An object:
  - is an entity that contains both data and functions
  - Its data is called the object’s attributes
  - Its functions is called the object’s member functions
Object-Oriented Programming

Object

Att1
Att2
Att3
Att4

Func1
Func2

Operate on attributes
Operate on attributes
Object-Oriented Programming

- Object is self-contained unit consisting of attributes and functions
- Problems that can occur due to separation of code and data are addressed through **encapsulation** and **data hiding**
  - Encapsulation refers to combining data and code into a single object, i.e. an object with attributes and functions
  - Data hiding refers to the ability of an object to hide its attributes from code that accesses it from outside the object
- Only the object’s member functions can access the attributes
- Code from outside object must call its member functions to get access to the attributes
Object-Oriented Programming

• Data hiding protects the data (attributes) from accidental corruption
• Outside code that uses the object’s member functions, do not need to know anything about the format or internal structure of the data => it only needs to interact with the object’s member functions
• If a programmer then later changes the data structure of the object’s attributes and modifies the member functions, it will not matter to the outside code
• Outside code only interacts with the member functions, and this interaction did not change

•
Object-Oriented Programming

- **Object reusability**: An object is not a stand-alone program, but is used by other programs that needs its service.

**Example:**
- Sharon is a programmer that developed an object or render animate a character. She knows a lot about computer graphics and her object contains all the mathematical operations to perform the animation and to deal with the computer’s hardware.
- Tom is writing an educational program and wants to add an animated character, but he knows nothing about computer graphics.
- Tom can use Sharon’s object to add an animated character to his program.
Classes and Objects

• Before an object can be created, it must be designed
• A programmer creates a **class** that specifies the attributes and member functions that a specific type of object may have
• A class is a description of an object
• If we want to use an object of the class in our code, we create an object of the class which is then called an **instance** of the class
• We can then use the object to call member functions of the object to get access to the attributes of the object
Classes and Objects

• We can use a class to create many objects or instances of the class in memory
• In OO, the variables and functions are all members of the specific class
• Each object or instance of the class will have all the variables and functions that were specified in the class, i.e. its attributes and member functions
Classes and Objects

Example:

• We specify a class called **Student**
• For class Student, we specify the following variables:
  o name, surname, studentNumber
• For class Student, we specify the following functions:
  o getName, setName, getSurname, setSurname,
    getStudentNumber, setStudentNumber
• We now create an instance of class Student, called newStudent
Classes and Objects

Example (cont.):
• Now we want to use the surname of this instance of Student
• We cannot access the attribute of the object directly
• We must use its member functions
• So we can now call its member function getSurname
• We will see later what the code looks like
Introduction to Classes

• A class is similar to a structure (struct), consisting of variables and functions
• General form:

```cpp
class classNamex
{
    declaration;
    // ... more declarations
    // may follow ...
};
```

```cpp
class Rectangle
{
    double width;
    double length;
}
```

What is wrong here?
Introduction to Classes

• **Important**: Members of the class are *private* by default – they cannot be accessed from outside the class. No statements outside the *Rectangle* class can access the *width* and *length* members.
• In C++ a class’s private members are hidden and can be accessed only by functions that are members of the class.
• A class’s *public* members can be accessed from outside the class.
• *private* and *public* are reserved keywords in C++ and are called **access specifiers** – they specify how class members may be accessed.
Public Member Functions

• To enable code outside class to access the class’s private members, public member functions should be defined

• Example:

```cpp
class Rectangle {
    private:
        double width;
        double length;
    public:
        voidsetWidth(double);
        voidsetLength(double);
        double getWidth();
        double getLength();
};
```

Even if private by default, add it here so that the various types of access specifiers are clear.

Public functions to access variables.
Public Member Functions

• To enable code outside class to access the class’s private members, public member functions should be defined

• Example:

```cpp
class Rectangle
{
    private:
        double width;
        double length;
    public:
        void setWidth(double);
        void setLength(double);
        double getWidth() const;
        double getLength() const;
};
```

Set functions set the value of the variable – change value. Called **mutators**.

Get functions return the value of the variable – do not change value. Called **accessors**.

*const* indicates that function will not change any variables of class
Public Member Functions

- To enable code outside a class to access the class’s private members, public member functions should be defined.
- Example:

```cpp
class Rectangle {
private:
    double width;
public:
    void setWidth(double);
    void setLength(double);
    double getWidth() const;
    double getLength() const;
private:
    double length;
};
```

Placement of access specifiers:
- Order does not matter.
- All members of the same access specifier do not have to be in the same place.

But you should adopt a consistent standard.
**Defining Member Functions**

*Scope resolution* operator. Function `setWidth` is a member of class `Rectangle`. Appears *after* return type.

```cpp
void Rectangle::setWidth(double w) {
    width = w;
}

void Rectangle::setLength(double l) {
    length = l;
}

double Rectangle::getWidth() {
    return width;
}

double Rectangle::getLength() {
    return length;
}
```
Defining Instance of Class

Class objects are not created in memory until they are defined.

Defining a class object is called the *instantiation* of a class.

Example:

```java
Rectangle box;
```

*box* is an instance of *Rectangle*.

This statement instantiates class *Rectangle*.

After instantiation, we can call the member functions of the class on the object of the class.

```java
Rectangle box;
box.setLength(10);
Box.setWidth(5);
```
Separating Class Specifications from Implementation

We separate class specification from separation:
• Class declaration is stored in its own header file (.h)
• Function definitions are stored in a .cpp file

```cpp
class Rectangle
{
    private:
        double width;
    public:
        void setWidth(double);
        void setLength(double);
        double getWidth() const;
        double getLength() const;
    private:
        double length;
};
```

This class declaration will be store in a .h file
We separate class specification from separation:
• Class declaration is stored in its own header file (.h)
• Function definitions are stored in a .cpp file

```cpp
void Rectangle::setWidth(double w) {
    width = w;
}

void Rectangle::setLength(double l) {
    length = l;
}

double Rectangle::getWidth() {
    return width;
}

double Rectangle::getLength() {
    return length;
}
```

These function definitions are stored in the .cpp file
Inline Functions

class Rectangle
{
    private:
        double width;
    public:
        void setWidth(double);
        void setLength(double);
        double getWidth() const
        {
            return width;
        }
        double getLength() const
        {
            return length;
        }
    private:
        double length;
};

Inline function – body of member function is written inside a class declaration

Compiled differently than other functions. A process called inline expansion the compiler replaces the call to an inline function with the code of the function itself

Improved performance, but program size can increase. Why?
Constructor

• Constructor is member function that is automatically called when a class object is created

```cpp
class Rectangle
{
    private:
        double width;
    public:
        Rectangle();
        void setWidth(double);
        void setLength(double);
        double getWidth() const;
        double getLength() const;
    private:
        double length;
};
```

Constructor – same name as class and no return value

In function definition class:
```cpp
Rectangle::Rectangle()
{
    width = 1;
    length = 2;
}
```

General form:
```
ClassName::ClassName(ParameterList)
```
Default Constructor

• Constructor is member function that is automatically called when a class object is created.

```cpp
class Rectangle
{
    private:
        double width;
    public:
        Rectangle();
        void setWidth(double);
        void setLength(double);
        double getWidth() const;
        double getLength() const;
    private:
        double length;
};
```

Default constructor – it takes no arguments.

If you write a class with no default constructor, C++ will automatically create a default constructor that does nothing.
Default Constructor

- Can use it with dynamically allocated objects

```cpp
Rectangle *rectPtr = nullptr;
rectPtr = new Rectangle;
```

Object is not created yet. Defines a pointer that can hold the address of any Rectangle object.

new creates an object and automatically calls the default constructor when the object is created.
Passing Arguments to Constructors

- Constructor can have parameters and accept arguments when an object is created.

```cpp
Rectangle::Rectangle(double w, double l) {
    width = w;
    length = l;
}
```
Default Arguments with Constructors

A default parameter value is listed in list of parameters. If no value is passed for that parameter, the default value is used.

```cpp
Rectangle::Rectangle(double w, double l=1)
{
    width = w;
    length = l;
}
```

Default value
Default Arguments with Constructors

- If all parameters of a constructor have default values, the constructor can be called without any parameters => it becomes the default constructor.

```cpp
Rectangle::Rectangle(double w=1, double l=2)
{
    width = w;
    length = l;
}
```

**Default values**
Default Arguments with Constructors

- If all constructor require parameters and not all parameters have default values, no default constructor exists.
- If you create an object, then you must pass required argument values. Otherwise compiler error.

```cpp
Rectangle::Rectangle(double w, double l=2) {
    width = w;
    length = l;
}
```