Introduction to Object-Oriented Programming

Programs

What is a program?

– A list of instructions for a computer to execute.

Program Hierarchy:

Machine – A program in a language the hardware can understand (1's and 0's). CPU's generally support a certain machine language know as the architecture (eg. X86, PowerPC)

Assembly – A program in a language that mirrors machine language which is human readable but still cumbersome to program in.

High-level Procedural (C, Fortran, Pascal) – High level languages that provide an abstraction of the hardware.

Object-oriented (C++, Java, Python) – Another even higher-level abstraction (often built on top of or in conjunction with procedural languages).

Goals of Object-oriented Languages

What do object-oriented languages (attempt) to give us?

- A way of writing code that is more structured.
- The ability to write code that is more reusable.
- An abstraction of the computer hardware that is more in line with how humans think about solving problems.
- Less worry about handling some programming intricacies
- A way to protect ourselves from doing things we shouldn't be defining the way our data can be accessed.

Object-Oriented basics

- A fundamental concept in an object-oriented language is the encapsulation of data and procedures (functions) together into units called **objects**.
- An object consists of:

  Name – a way of referring to an object inside a program (eg. A Fraction object might be called F1).

  Member Data – data contained within an object (eg. Fraction has an integer numerator and denominator).

  Member Functions – routines that act upon the data within an object (eg. The fraction object might have a function that returns the decimal representation of the fraction stored).

  Interface – defines the ways a programmer may directly access the member data/functions of an object (more on this next lecture).

Classes

- A **class** is another fundamental concept in an object-oriented language that provides a blueprint for a new type ('classification') of object.
- A class outlines the data, functions and the interface objects of that class will receive.
- A class also defines how objects of that class behave by providing code that implements the functions associated with the class.
- A programmer can create one or more objects from a class
  - Similar to building multiple houses from one set of blueprints.

What is so special about objects/classes?

- The C programming language provides a similar concept called **structs**:

```
struct point {    /* define a new type called point */
                  int x;
                  int y;
};
point p1; /* create an instance of point called p1 */
p1.x = 1; /* use p1 */
p1.y = 2;
```

- C structs only hold data.
- Objects hold data and functions that act on that data.
  - Much more powerful.

How to use objects

- DDU – Declare, Define, Use
  - Declare a class

    Choose what objects of this class will store (member variables), and how objects will behave (member functions).
  - Define member functions

    Provide an implementation for the member functions in the class.
  - Use class to create objects

    You can declare an new object instance of your class just like declaring any other variable (eg. int x).

Example Class Declaration

```
class Circle
{
public: /* interface, we will cover later */

   void SetRadius(double r);  /* sets member variable radius to r */
   double AreaOf();           /* returns area of circle as a double */

   double radius;             /* radius of circle stored as double */

}; /* don't forget ';' */
```

Defining Member Functions

There are two ways to provide the member function definitions for a class:

- Inside the class declaration using {} (we will not use)
- After the class declaration (this is the method we choose)

How to refer to a member function:

className::memberFuntionName

- this identifier refers to the member function memberFunctionName of class className (e.g. Circle::SetRadius)
- The double colon :: is called the scope resolution operator

After the class declaration, member functions are defined just like any other function

Example Member Function Definition

```
//Declaration:
class Circle
{
public:
   void SetRadius(double r); /*sets member variable radius to r */
   double AreaOf();               /* returns area of circle as a double */
private:
   double radius;                    /* radius of circle */
};

/* Definition (Implementation) */
void Circle::SetRadius(double r)
{
   radius = r;  /* radius refers to this object’s member variable */
'radius'
}

double Circle::AreaOf()
{
  return (3.14*radius*radius);
}
```

Object Use

After a class has been declared and defined, an object of that class can be be declared (also known as creation or instantiation) and used.

A programmer can declare an object with the following format:

```
ClassName ObjectName;
```

This statement creates an object based on the blueprint of class ‘ClassName’ and the object can be referred to by the identifier (variable name) ‘ObjectName’

The ‘ . ’ (dot) operator can be used to access an object’s public members

The format for referring to an object’s member is:

```
ObjectName.MemberFunction()      OR
ObjectName.MemberVariable
```

Putting it All Together

- See sample1.cpp
- To recap, this program:
    - declares the class Circle and outlines its members and interface
    - defines the implementation for the member functions of the Circle class
    - declares two objects of the class Circle, referred to as C1 and C2
    - uses the interfaces of C1 and C2 to store the radius of two circles and later to calculate the area of those circles

Review

What is a class?

What is an object?

What is the difference between a declaration and a definition?