Classes

Object Oriented Programming in C++

Contents

- Explanation of classes
- Example of a class
- OO design
- Encapsulation
- Data Abstraction
- Data protection
- Access specifiers
- .h and .cpp files
- The *this* pointer

Classes

- Classes are very similar to structures in the way they are created and in the way they are accessed.
  - You are creating a new data type which is reusable in, hopefully many projects
- The overall design process is similar as well, but
  - With procedural we're identifying data and operations
  - With OO we're identifying objects, their relationships and their interfaces.

Classes Cont.

- A class is both data (variables), representing an object's internal state, and the operations (functions) that can be performed on that data.

Example of a simple class

```
class Sprite
{
public:

    //Data
    float x, y;
    int textureID;

    //Functions
    int GetTextureID()
    {
        return textureID;
    }

    void SetTextureID(int id)
    {
        textureID = id;
    }
};
```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Object Oriented Design

- 4 Basic Concepts
  - Encapsulation and data abstraction
  - Data Protection / Interface
  - Inheritance
  - Polymorphism
- We will focus on the top two in this lecture

Encapsulation

- Classes allow us to encapsulate data and functions
  - That is, it allows us to group all the data and functions belonging to one object into one place
  - Instead of having enemyX, enemyY, we can have an Enemy object, with an x and y variable.
  - We can then create as many Enemy objects as we like, and each will have their own x and y variable.
  - You have done this already with structures.

Encapsulation

- It is not just variables that can be stored in classes, we can also store functions as well:

```
class Enemy
{
public:
	float m_fX, m_fY;
	float m_fMovementX, m_fMovementY;

	void Move();
};

//Further on...
Enemy oEnemy1;
oEnemy1.Move();
```

Data Abstraction

- Abstraction means we should implement “generic” classes.
- Generic classes can be used across multiple projects and aren’t tied down to any low level details.

Data Protection

- With classes, we can restrict who is able to access the variables and functions owned by this class.
- One use for this is to validate data passed to the object before it is assigned to a variable.
- This is done using the public, protected and private keywords. These are known as access specifiers.
  - We will cover public and private in this lecture.
  - We will cover the protected keyword when we discuss inheritance and the friend keyword
- In order for data protection to be done correctly, the class needs to be well planned out.

Access Specifiers

- Access specifiers allow explicit enforcement of encapsulation
- There are three access specifiers
  - **public**
  - **protected**
  - **private**
- Generally good practice for data and non-interface functions to be declared private

Access Specifiers (cont.)

- **public**
  - any function or class can access this member
- **protected**
  - only functions from this class or *derived classes* or *friends* can access this member
  - (derived classes and friends to be covered later)
- **private**
  - only functions from this class and friends can access this member

Public

- Publicly accessible data and functions can be accessed and called outside of that class.
- Other objects and functions can use these to modify and interact with your class.

Private

- Private members can only be accessed by objects of that class
- They are used to hide data and functions which should not be accessed by other objects
- Often, public accessor functions are used to access private variables to allow us to validate the data stored in them

Private

```
class Enemy
{
public:
	void SetXPosition(float a_fX)
	{
		if (a_fX < 0.0f)
			m_fX = 0;
		else
		 m_fX = a_fX;
	}

private:
	float m_fX;
};
```

```
void main()
{
	Enemy oEnemy;
	//Error!  This won't compile
	oEnemy.m_fX = 5.0f;

	/*
	Instead, we use the SetXPosition()
function,	allowing us to validate the
data first.
	*/
	oEnemy.SetXPosition(5.0f);
}
```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Simplifying .h files

- Usually we place class definitions inside a header file.
- In here we only list the function prototypes, not the definitions.
- Notice the preprocessor directives – these are placed here to make sure the Sprite class is not declared more than once, even if the header file is included in multiple locations.

```
#ifndef _SPRITE_H_
#define _SPRITE_H_

class Sprite
{
public:

     // get and set texture ID
     int GetTextureID();
     void SetTextureID(int a_id);

     float m_x, m_y; // position

private:
     int m_textureID; // texture
};

#endif
```

Creating .cpp files

- The function definitions are placed inside a corresponding .cpp file.
- The .cpp file is beyond class scope, so the class name and the :: (scope resolution) operator must be used before the name of each function (but *after* the return type!)
- Don’t forget to include the corresponding header file

```
#include “Sprite.h"

// Texture Access
int Sprite::GetTextureID()
{
     return m_textureID;
}

void Sprite::SetTextureID(int a_id)
{
     m_textureID = a_id;
}
```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Member Functions

- Member functions have slightly different scoping rules than other functions
- They follow all the normal scoping rules but ...
- They can access other member variables and functions of the class

Miscellaneous

- Within a class, *this* refers to the class itself; e.g.:

```
class Test
{
public:
     int m_x;
     int getX()
     {
          return m_x;
     }
     void setX(int a_x)
     {
          this->m_x = a_x;
     }
};
```

- *this* is a pointer to the current object
  - therefore we use -> to access the elements of the object being pointed to.

Summary

- Classes and objects are quite a big topic, and can be a difficult concept to grasp. It can take some time before OOP will be a natural thought process. As always, practice is the best way to solidify the concept.
- There are many more concepts involved with OOP that will make the use of classes seem more obvious.

References

- Gaddis, T, 2012, *Starting Out with C++: From Control Structures Through Objects*, 7th edition, Pearson Education

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX