717310: Game Programming

Steffan Hooper

Wednesday, 5 August 2015

Overview

- More C++:
  - Classes
  - Constructors, Destructors, Methods
  - Encapsulation
  - Inheritance, Overloading
  - Virtual Methods, Pure Virtual
  - Object Copying
- The C++ Standard Template Library
  - Containers, Algorithms

Console Game Programming

- C++: Classes
  - Keyword: **`class`**
  - Split Architecture: Two files…
    - Generally *declaration* in the **`.h`** file:
      - Member data.
      - Member function signatures/prototypes.
    - The **`.cpp`** file contains the *implementation*:
      - Method definitions.
  - Access Modifier Keywords:
    - **`public`**, **`protected`**, **`private`**.

Console Game Programming

- C++: Declaring Classes
  - Example **`enemy.h`** file:

```
class Enemy
{
    public:
        Enemy();
        ~Enemy();
        void setHealth(int h);
        int getHealth() const;

    private:
        int m_iHealth;
};
```

Public Interface

Constructors

Methods

Encapsulated member data

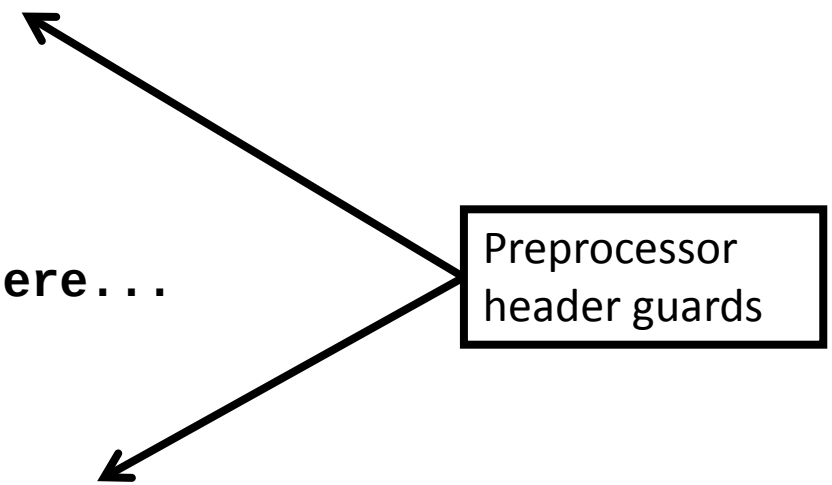
Console Game Programming

- C++: Declaring Classes continued…
  - Header Guards (Macro Guard/Include Guard)
    - Use the preprocessor to avoid double inclusion…
  - Example **`enemy.h`** file, with header guards:

```
#ifndef __ENEMY_H__
#define __ENEMY_H__

class Enemy
{
    // Declaration here...
};

#endif // __ENEMY_H__
```



Console Game Programming

- C++: Defining Classes
  - Example **`Enemy.cpp`** file:

```
#include "enemy.h"

Enemy::Enemy()
: m_iHealth(0)
{
    // Constructor Body...
}

Enemy::~Enemy()
{
    // Destructor Body...
}
```

Get access to the class's declaration.

Member initialisation list

Destructor!

Console Game Programming

- C++: Defining Classes continued…
  - Example **`Enemy.cpp`** file continued…

```
void
Enemy::setHealth(int h)
{
    m_iHealth = h;
}

int
Enemy::getHealth() const
{
    return (m_iHealth);
}
```

Accessor methods

This method can be **`const`** as it does not mutate the state of the object!

Console Game Programming

- C++: Defining Classes continued…

Declaration

```
#ifndef __ENEMY_H__
#define __ENEMY_H__

class Enemy
{
public:
    Enemy();
    ~Enemy();

    void SetHealth(int health);
    int GetHealth() const;

private:
    int m_iHealth;
};

#endif // __ENEMY_H__
```

enemy.h

```
#include "enemy.h"

#include <iostream>

Enemy::Enemy()
: m_iHealth(0)
{
    std::cout << "Enemy created!" << std::endl;
}

Enemy::~Enemy()
{
    std::cout << "Enemy destroyed!" << std::endl;
}

void
Enemy::SetHealth(int health)
{
    m_iHealth = health;
}

int
Enemy::GetHealth() const
{
    return (m_iHealth);
}
```

enemy.cpp

Definition

Console Game Programming

- C++: Using Classes (Objects!)
  - Declaring and using an object, allocated in the stack frame…
  - Example **`main`** stack frame **`Enemy`** object:

```
// The class header included above here...
int main()
{
    Enemy badGuy;
    badGuy.SetHealth(50);

    std::cout << badGuy.GetHealth() << std::endl;
}
```

Console Game Programming

- C++: The Freestore:
  - The Heap…
    - Dynamic memory allocation…
  - Keywords: **`new`**, **`delete`**
  - Example:

```
int* pFreestore = new int;
*pFreestore = 47;
std::cout << *pFreeStore << std::endl;
delete pFreestore;
pFreestore = 0;
```

Console Game Programming

- C++: The Freestore continued…
  - Dynamic memory allocation:
  - Keywords: **`new []`**, **`delete[]`**
  - Example:

```
int* pArray = new int[10];
pArray[0] = 110;
// ...
pArray[9] = 990;
delete[] pArray;
pArray = 0;
```

Console Game Programming

- C++: The Freestore continued…
  - Allocating and using objects on the Freestore.
  - Example:

```
int main()
{
    Enemy* pEnemyObject = new Enemy();

    pEnemyObject->SetHealth(87);
    int x = pEnemyObject->GetHealth();

    return (0); // This program will leak!
}
```

Console Game Programming

- C++: The Freestore continued…
  - Allocating, and forgetting to de-allocate causes memory leaks…
  - Example:

```
int main()
{
    Enemy* pEnemyObject = new Enemy();
    // ... Object used over time ...
    delete pEnemyObject;
    pEnemyObject = 0;
    return (0);
}
```

Console Game Programming

- C++: Inheritance:
  - Declaring and Using Subclass
  - Example:

```
class MySuper
{
    //...
};
class MySub : public MySuper
{
    //...
};
```

Generally these declaration are in separate **.h** files

Console Game Programming

- C++: Inheritance continued…
  - Declaring Interfaces
  - Example:

```
class MyInterface
{
    public:
        virtual void method() = 0;
};
```

Pure Virtual Method: There will be no definition body for this method… the deriving class must provide a body.

Console Game Programming

- C++: Inheritance continued…
  - Declaring Abstract Data Types
  - Example:

```
class MyADT
{
    public:
        virtual void method();
    protected:
        int m_data;
};
```

Virtual Method: The class **MyADT** may provide a method implementation, but a subclass can also… its abstract!

Console Game Programming

- C++: Polymorphism:
  - Different runtime object types, used through an single interface…
  - Example:

```
class MyADT
{
    public:
        virtual void polymorphicFunc();
};
```

  - Subclasses implement their own behaviour…

Console Game Programming

- C++: Polymorphism continued…
  - Dynamic Method Binding:
    - Method assignment unknown at compile time.
    - Method address assigned at runtime.
  - V-Table:
    - Virtual Function Table… pointers to functions!
  - Generally:
    - The compiler creates a hidden data member in the class declaration when the **`virtual`** keyword is used with the class.

Console Game Programming

- C++: Other Keywords:
  - **`this`**: Hidden pointer inside each class method.
  - **`static`**:
    - Static local variable…
    - Static class member…
    - Static class method…
  - **`const`**: “Const correctness”
    - Constant variable…
      - Locals and Parameters!
    - Constant methods…
    - **`const`** used with pointers…

Console Game Programming

- C++: Operator Overloading:
  - Create operator behaviour for types…
  - Example:

```
class CVector
{
public:
    int x, y;
    CVector operator+(CVector);
};

CVector CVector::operator+(CVector param)
{
  CVector temp;
  temp.x = x + param.x;
  temp.y = y + param.y;
  return (temp);
}
```

Console Game Programming

- C++: Object Copying:
  - Example:

```
#include "enemy.h"

int main()
{
    Enemy a;
    Enemy b;

    b = a; // Member-wise copy a to b.
}
```

Two separate **Enemy** instances stored in the **main** stack frame, instances are named **a** and **b**.

Copy each member field's data from **b** into the **a** object. This is known as a "Shallow Copy"…

Console Game Programming

- C++: Object Copying continued…
  - Disable implicit object copying…
    - Make the assignment operator private…
    - And the copy constructor…
      - Now there will be compile errors when objects are copied!
  - Example:

```
class Example
{
    private:
        Example(const Example& obj);
        Example& operator= (const Example& obj);
    public:
        /* Other class members here */
};
```

Avoid accidental object copying!

Console Game Programming

- C++: Constructors:
  - Default Constructor: No parameters
  - Overloaded Constructor: Parameters!
  - Copy Constructor: Reference of same type param.
  - Example:

```
class Enemy
{
public:
    Enemy();
    Enemy(int startingHealth);
    Enemy(Enemy& copyMe);
```

Console Game Programming

- C++: Destructor:
  - Only one signature for the Destructor!
  - Example:

```
class Enemy
{
   public:
         ~Enemy();
};
```

  - If a class has any **`virtual`** methods…
    - Declare the destructor **`virtual`** too!
    - Allow for correct destruction of objects…

Console Game Programming

- C++: Templates and STL…
  - Generics/Templates:

```
template <class T>
class aPair
{
    T values[2];
public:
    aPair (T first, T second)
    {
        values[0] = first;
        values[1] = second;
    }
};
```

aPair of any type T can be instanced at compile time. The code is generated by the compiler for the particular type requested.

For example:

```
aPair<int> myIntPair;
```

Or:

```
aPair<Enemy> enemies;
```

  - Standard Template Library:
    - Containers, Iterators
    - Algorithms, Functors

Console Game Programming

- C++ Strings:
  - **`#include <string>`**
  - Class type: **`std::string`**
  - Useful Methods:
    - **`c_str()`** returns a **`char*`** to the text in the string object.
    - **`at()`** gets a character in the string…
      - **`[ ]`** indexing is also overloaded do the same!
    - **`length()`** returns the length of the string.
    - **`append()`** adds to the string.

Console Game Programming

- C++ Vectors:
  - **`#include <vector>`**
  - Class type: **`std::vector`**
    - It's a resizeable container… a dynamic array!
  - Useful Methods:
    - **`size()`** returns the number of items in the container.
    - **`at()`** access an element in the container…
      - **`[ ]`** indexing is also overloaded do the same!
    - **`empty()`** returns if the container is empty.
    - **`front()`** gets the first item in the container.
    - **`back()`**gets the last item in the container.

Console Game Programming

- C++ Vectors: Iteration:
  - **`#include <vector>`**
  - Class type: **`std::vector<>::iterator`**
    - This object allows you to iterate through the container…
  - **`std::vector`** Methods:
    - **`begin()`** returns an iterator to the beginning.
    - **`end()`**returns an iterator to the end.
  - Using the **`std::vector<>::iterator`**:
    - **`*`** "dereferences" the iterator to get at what the iterator "points" to.
    - **`++`** move the iterator to the next item in the container.

Console Game Programming

- C++ Vectors and Iteration Example:

```
#include <iostream>
#include <vector>

int main ()
{
    std::vector<int> exampleVector;

    for (int i = 1; i <= 5; ++i)
    {
        exampleVector.push_back(i);
    }

    std::cout << "The vector contains:";

    std::vector<int>::iterator iter = exampleVector.begin();

    while(iter != exampleVector.end())
    {
        std::cout << " " << *iter;
        ++iter;
    }

    std::cout << std::endl;

    return 0;
}
```

Console Game Programming

- C++ Vectors and Iteration Example:

```
#include <iostream>
#include <vector>

int main ()
{
    std::vector<int> exampleVector;

    for (int i = 1; i <= 5; ++i)
    {
        exampleVector.push_back(i);
    }

    std::cout << "The vector contains:";

    std::vector<int>::iterator iter = exampleVector.begin();

    while(iter != exampleVector.end())
    {
        std::cout << " " << *iter;
        ++iter;
    }

    std::cout << std::endl;

    return 0;
}
```

Declare a **`vector`** of **`int`**s. Since it's a local declaration, it will also run the **`vector`**'s constructor.

Push items into the container…

Get an iterator for the start of the container…

While the iterator is not at the end of the container…

Get what the iterator "points to"…

Move to the next item in the container…

Console Game Programming

- Other STL Containers (Data Structures):
  - **`#include <set>`**
  - **`#include <list>`**
  - **`#include <stack>`**
  - **`#include <queue>`**
  - **`#include <deque>`**
  - **`#include <map>`**
  - All have similar functionality to **`std::vector`**…
    - But display the characteristics of their data structure…

Console Game Programming

- STL Algorithms:
  - **`#include <algorithm>`**
  - This gives access to the **`std::sort`** method.
    - Can sort items in a container…
      - Provided the item has an **`operator<`** operation.
  - And many more methods, including but not limited to:
    - **`std::random_shuffle()`**
    - **`std::copy()`**
    - **`std::move()`**
    - **`std::reverse()`**

Exercises

- Week 3:
  - Day 005.1 – C++: Classes and Objects
  - Day 005.2 – Local vs Freestore Objects
  - Day 005.3 – Object Factory
  - Day 005.4 – String manipulations
  - Day 005.5 – STL Containers and Iterators

Summary

- More C++:
  - Classes
  - Constructors, Destructors, Methods
  - Encapsulation
  - Inheritance, Overloading
  - Virtual Methods, Pure Virtual
  - Object Copying
- The C++ Standard Template Library
  - Containers, Algorithms