The University Of Sheffield.

The University of Sheffield Research IT

C++ Programming – Classes

Michael Griffiths

Corporate Information and Computing Services

The University of Sheffield

Email m.griffiths@sheffield.ac.uk

The University of Sheffield Research IT

Presentation Outline

- Differences between C and C++
- Object oriented programming
- Classes
- Data Abstraction

The University of Sheffield Research IT

C++ AS "BETTER" C

- More robust
  - Stronger type checking
- Hides complexity
- Enables user-extensible language

The University of Sheffield Research IT

C++ ANSI STANDARD

- Standard library
- Standard Template library
  - Wide range of collection classes
- Exception handling
- Namespaces

The University Of Sheffield.

The University of Sheffield Research IT

STANDARD LIBRARY

- Header file names no longer maintain the **.h** extension typical of the C language and of pre-standard C++ compilers
- Header files that come from the C language now have to be preceded by a **c** character in order to distinguish them from the new C++ exclusive header files that have the same name. For example **stdio.h** becomes **cstdio**
- All classes and functions defined in standard libraries are under the **std** *namespace* instead of being global.

ANSI C LIBRARIES AND ANSI C++ EQUIVALENTS

ANSI-C++	ANSI-C
<cassert>	<assert.h>
<cctype>	<ctype.h>
<cfloat>	<float.h>
<cmath>	<math.h>
<csetjmp>	<setjmp.h>
<csignal>	<signal.h>
<cstdarg>	<stdarg.h>
<cstddef>	<stddef.h>
<cstdio>	<stdio.h>
<cstdlib>	<stdlib.h>
<cstring>	<string.h>

The University Of Sheffield.

The University of Sheffield Research IT

FEATURES

- Comments
- Stream input and output
- Constants
  - The const keyword
- Declarations as statements
- Inline expansion
- References

MORE FEATURES

- Dynamic memory allocation (new and delete)
- Function and operator overloading
- User defined types
- Data abstraction and encapsulation
  - Use classes to incorporate new types
- Scoping and the Scope Resolution Operator
- Template

The University of Sheffield Research IT

COMMENTS

- Traditional C comment (can be used in C++)

  /*This is a traditional C comment*/

- C++ Easier comment mechanism

  //This is a C++ comment

The University Of Sheffield.

The University of Sheffield Research IT

STREAM INPUT AND OUTPUT

- C++ Provides operators for input and output of standard data types and strings
  - Stream insertion operator <<
    - Alternative to printf
  - Stream extraction operator >>
    - Alternative to scanf
- To use C++ stream operators include the header iostream

STREAM INPUT AND OUTPUT EXAMPLES

- cout<< "Hello World\n";
  - Displays Hello World
  - Note use of the escape characters used in C
- cin>>a>>b;
  - Assigns data read from the input stream to the variables a and b
- Can still use printf and scanf but it is not good to mix usage of these with stream operators

The University Of Sheffield.

The University of Sheffield Research IT

READING FILES USING IFSTREAM

```
#include <iostream>
#include <fstream>
using namespace std;
 int main ()
{
      ifstream infile;
      infile.open ("test.txt", ifstream::in);
      while (infile.good()) cout << (char) infile.get();
      infile.close();
      return 0;
}
```

The University Of Sheffield.

The University of Sheffield Research IT

USING OFSTREAM TO WRITE TO FILES

```
// ofstream::open
#include <fstream>
using namespace std;
int main ()
{
      ofstream outfile;
      outfile.open ("test.txt", ofstream::out | ofstream::app); outfile << "This sentence is appended to
            the file content\n";
      outfile.close();
      return 0;
 }
```

The University Of Sheffield.

The University of Sheffield Research IT

THE CONST KEYWORD

- In C constants normally defined using
  - #define PI 3.14159265358979323846
- In C++ use the const keyword
  - Declare any variable as const
  - const double PI = 3.14159265358979323846;
- A **const** object may be initialized, but its value may never change.

The University of Sheffield Research IT

ADVANTAGES OF USING CONST

- Allows the compiler to ensure that constant data is not modified generate more efficient code
- Allows type checking

The University Of Sheffield.

The University of Sheffield Research IT

CLASSES

- Data Abstraction
- Data encapsulation
- Data hiding
- Objects

The University Of Sheffield.

The University of Sheffield Research IT

ENCAPSULATION AND ABSTRACT DATA TYPES (ADT)

- Model defines an abstract view of a problem
- Represent model as a collection of abstract data types
- Abstract Data type
  - Abstract data structure
  - Operations defining an interface
- Operations of the interface are the only ones allowing access to the types data structure

The University Of Sheffield.

The University of Sheffield Research IT

PRIMARY CHARACTERISTICS OF OBJECT ORIENTED PROGRAMS

- Encapsulation
  - Mechanism for enforcing data abstraction
- Inheritance
  - Allow objects to be derived from previously defined objects
- Polymorphism
  - Different types of objects respond to the same instruction to perform a method that is specific to the particular kind of object

The University of Sheffield Research IT

OBJECT ORIENTED PROGRAMMING (OOP)

- Successful OOP is dependent on good design, this means
  - Selecting classes that are relevant that provide the solution for the particular problem under investigation
  - Always try and get the right class structure for the application
  - In the software lifecycle sometimes necessary to refine classes and introduce new classes

The University Of Sheffield.

The University of Sheffield Research IT

OBJECT ORIENTED DESIGN

- Use OO design techniques such as UML (Unified Modelling Language)
  - Use Case Modelling
  - Class diagrams
  - Sequence diagrams
  - Activity diagrams
- Aim to minimise the number of iterations in the development cycle

The University Of Sheffield.

The University of Sheffield Research IT

CLASSES

- Class is a data type
- Like a structure but includes
  - Methods i.e. functions to manipulate the members
- An object is an instantiation of a class
- Each Class instantitation i.e. each object
  - Own copy of data members
  - Share methods with objects of same type

The University Of Sheffield.

The University of Sheffield Research IT

CLASS DECLARATION

```
class classname {
public:
        classname();
        ~classname();
        vartype m_var;
        vartype method(var list);
private:
        private members and methods here

}
```

The University Of Sheffield.

The University of Sheffield Research IT

MEMBER FUNCTIONS

- Constructor function
  - Methods to perform when object created
  - Initializes object data
- Destructor function
  - Methods to perform when object destroyed
  - Frees any memory used by the object

The University of Sheffield Research IT

DATA MEMBER TYPES

- public (default)
  - Members declared as public can be used by any other objects
- private
  - Members declared as private can only be read, written by members of this class
- protected (see later)
  - Data members can be accessed by members of this class and classes that are derived from this class

SIMPLE CLASS DECLARATION

```
//Program8.cpp
//C++ Class example illustrating
//class constructors and destructors
#include <iostream>
#include <cstring> //strcpy and strlen
class WrgComputeNode {
    public:
        WrgComputeNode(const char *sName ); //Constructor
        ~WrgComputeNode();                  //Destructor
        char *GetName(){ return m_sName; }
    private:
        char *m_sName;
};
```

MEMBER FUNCTIONS FOR WRGCOMPUTENODE CLASS

- Function name preceeded by class name and a scope resolution operator

```
WrgComputeNode::WrgComputeNode(const char *sName )
{
        m_sName = new char [strlen(sName)];
        strcpy(m_sName, sName);
}
//Delete memory that was allocated for the
//creation of the node name
WrgComputeNode::~WrgComputeNode()
{
        delete [] m_sName;
}
```

The University of Sheffield Research IT

USING CLASSES

- Initialising an object

```
WrgComputeNode Sheffield("Titania");
WrgComputeNode Leeds1("Maxima");
WrgComputeNode York("Pascali");
```

- Accessing Members and Calling Methods

```
cout << "Sheffield grid node is " << Sheffield.GetName() <<endl;
cout << "Leeds grid node 1 is " << Leeds1.GetName() << endl;
cout << "York grid node is " << York.GetName() << endl;
```

CLASS CONSTRUCTORS AND THE NEW OPERATOR

- In C++ the new function takes the place of malloc().
- To specify how the object should be initialized, one declares a *constructor* function as a member of the class,
  - Name of the constructor function is the same as the class name
- Methods for declaring a stack class

```
stack s1(17);
stack *s2 = new stack(2);
```

DESTRUCTORS AND THE DELETE OPERATOR

- Just as new is the replacement for malloc(), the replacement for free() is delete.
- To get rid of the Stack object we allocated above with new, one can do:

  delete s2;
- This will deallocate the object, but first it will call the *destructor* for the Stack class, if there is one.
- This destructor is a member function of Stack called ~Stack()

STACK CLASS DEFINITION

```
class stack{
    public:
      stack(int sz=10);
      ~stack();
          void push(int value);
          int pop();
    public:
          int m_size;
          int m_count;
          int *m_stack
};
```

The University Of Sheffield.

The University of Sheffield Research IT

STACK CONSTRUCTOR

```
stack::stack(int sz)
{
    m_size=(sz>0?sz:10);
    m_stack = new int[m_size];
    m_count=0;
}
```

STACK DESTRUCTOR

```
stack::~stack()
{
        delete [] m_stack;
        cout << "Stack deleted." << endl;
}
```

EXAMPLE OF USING THE STACK CLASS

```
int main()
{
        stack s1(17);
        stack *s2 = new stack(2);
        s1.push(5);
        s2->push(8);
        s2->push(7);
        //pop items from the stack
        cout << "Popping stack s1 gives " << s1.pop() << endl;
        cout << "Popping stack s2 gives " << s2->pop() << endl;
        cout << "Popping stack s2 gives " << s2->pop() << endl;

        cout << "deleting stack s2: ";
        delete s2;
        return 0;
}
```

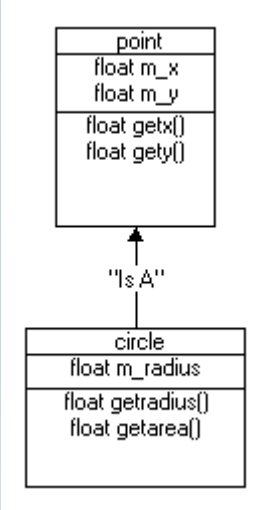
The University Of Sheffield.

The University of Sheffield Research IT

INHERITANCE

- Inheritance is a mechanism by which base classes can inherit data and methods from a base class
- The new class is called a derived class
- An object of a derived class may be treated as an object of the base class

A SHAPE DEFINED BY A OBJECT DEFINING ITS POSITION



The University Of Sheffield.

The University of Sheffield Research IT

BASE CLASS DEFINITION

```
class point {
public:
        point(float x=0, float y=0);  //Default constructor
        void setpoint(float x, float y);
        float getx() const {return m_x;}
        float gety() const {return m_y;}
protected:   //accessible only by derived classes
        float m_x;
        float m_y;
};
```

The University of Sheffield Research IT

DERIVED CLASS DEFINITION

```
class circle : public point {   //circle inherits from point
public:
        circle(float r, float x, float y); //default constructor
        void setradius(float r);
        float getradius() const;
        float getarea() const;
protected:
      float m_r;
};
```

CREATING INSTANCES

```
point *pointPtr;
point p(2.1, 3.2);
circle *circlePtr;
circle c(3.2, 1.7, 3.9);

pointPtr = (point *) &c;
circlePtr = &c;
```

The University Of Sheffield.

The University of Sheffield Research IT

CALLING OBJECT METHODS AND MEMBERS

```
cout << "Point p: x=" << p.getx() << " y=" << p.gety() << endl;
cout << "Circle c: radius=" << c.getradius() <<" x=" << c.getx()
        << " y=" << c.gety() << " area=" << c.getarea() << endl << endl;
```

TREATING A CIRCLE OBJECT AS A POINT OBJECT

```
cout << "Using point pointer to access circle base class information"
        << endl;
cout << "Circle c: x=" << pointPtr->getx()
        << " y=" << pointPtr->gety() << endl;
```

TREATING A CIRCLE OBJECT AS A CIRCLE BASE CLASS

```
cout << "Using circle pointer to access circle base class information"
        << endl;
cout << "Circle c: x=" << circlePtr->getx()
        << " y=" << circlePtr->gety() << endl;
```

The University Of Sheffield.

The University of Sheffield Research IT

VIRTUAL FUNCTIONS AND POLYMORPHISM

- Through virtual functions and polymorphism it is easier to extend systems and to reuse objects
- programs take on a simplified appearance with less branching logic
- Easier to maintain applications

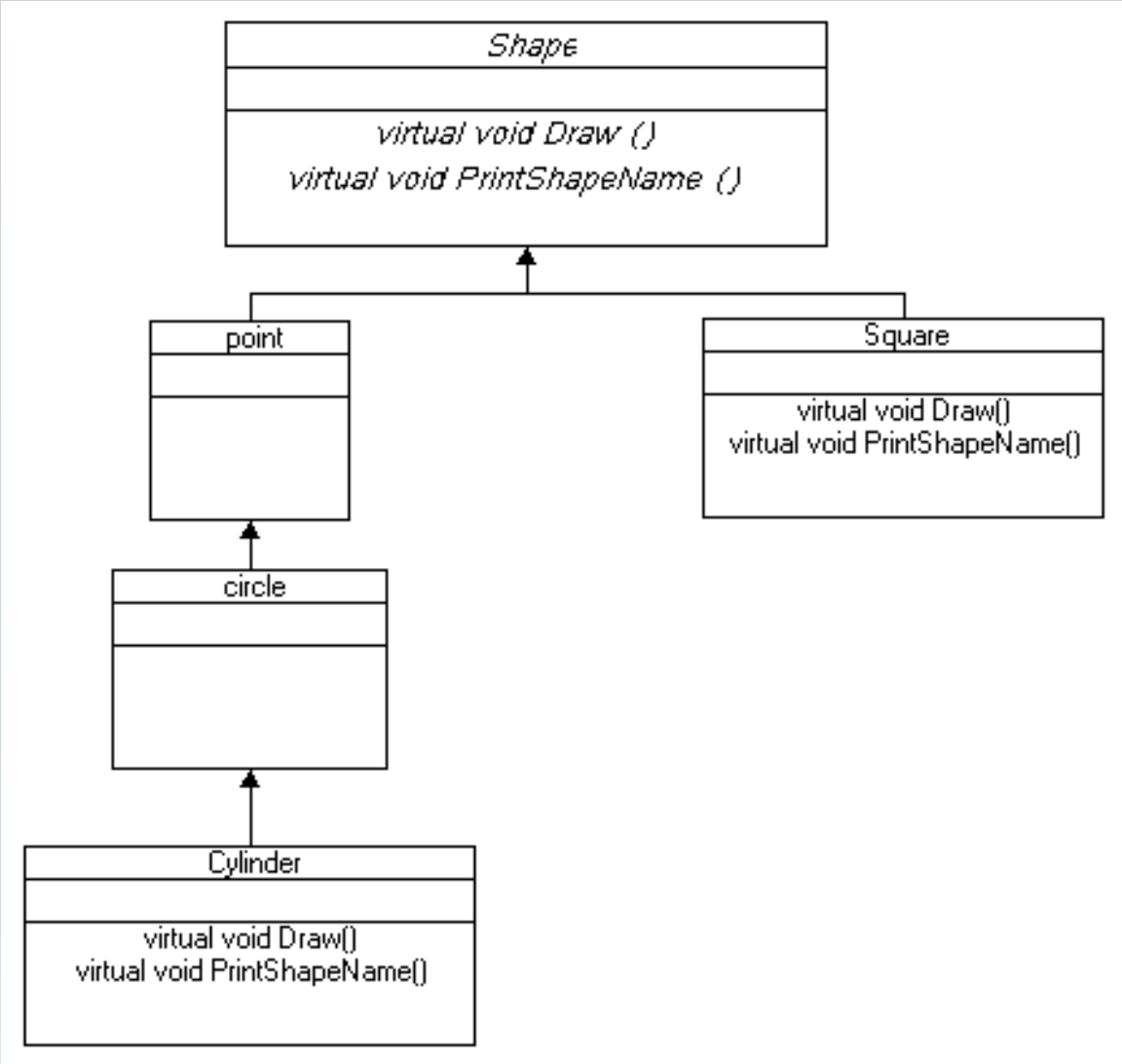
The University Of Sheffield.

The University of Sheffield Research IT

THE SHAPES EXAMPLE

- treat all shapes as a generic instances of the base class shape
- When a shape is drawn we then simply call the Draw method of the shape base class
- program determines dynamically at execution time which Draw method should be used
- Extend our circle class by introducing different shapes and a base class called shape

POLYMORPHISM – ABSTRACT CLASSES



The University Of Sheffield.

The University of Sheffield Research IT

ABSTRACT SHAPE CLASS

```
class shape {
public:
        virtual float area() const {return 0.0;}
        virtual float volume() const {return 0.0;}
        virtual void printShapeName()=0;  //pure virtual
};
```

A CONCRETE CLASS

```
class square : public shape{
public:
        square(float x=0);  //Default constructor
        void setside(float x);
        float getside() const {return m_side;}
        virtual void printShapeName(){cout << "Square: ";}
        virtual float area() const {return(m_side*m_side);}
protected:   //accessible only by derived classes
        float m_side;
};
```

CODE FRAGMENT ILLUSTRATING POLYMORPHISM

```
void main()
{
        int i;
        shape *shapearray[3];
        point p(2.1, 3.2);
        circle c(3.2, 1.7, 3.9);
        square s(4.5);
        shapearray [0] = (shape *)&p;
        shapearray [1] = (shape *)&c;
        shapearray [2] = (shape *)&s;
        for(i=0; i<3; i++)
        {
          cout << "Area of ";
          shapearray[i]->printShapeName();
           cout << shapearray[i]->area() << endl;
        }
}
```

ABSTRACT CLASSES AND VIRTUAL FUNCTIONS

- abstract class called shape which is inherited by two new shapes cylinder and square
- since shape is abstract and it does not express a concrete realisation of a shape we cannot define the Draw method
  - Draw function of the shape base class is said to be vitual

The University Of Sheffield.

The University of Sheffield Research IT

POLYMORPHIC BEHAVIOUR THROUGH ABSTRACTION

- Virtual functions in an abstract base class.
    - Desired behaviour obtained dyamically by selecting the correct methods for concrete realisations of a class
- Method is defined in each of the derived classes.
- In this sense a collection of objects referenced using the shape class exhibit polymorphic behaviour
    - behaviour dependent on the specific methods defined for the derived classes.

The University Of Sheffield.

The University of Sheffield Research IT

RECOMMENDATION

- Do not mix C and C++ Programming Styles
  - External c libraries can still be called
- For C++ programs use the ANSI C++ header nad not the ANSI C headers standard libraries
- Keep things simple
  - Build Create new classes when necessary
  - Take care when using inheritance
- Avoid re-inveting the wheel, use SDK's
  - See the code repository references below

The University Of Sheffield.

The University of Sheffield Research IT

CONCLUSION

- Advantages
  - Powerful development language for HPC applications
  - Portable
  - Reusable
  - Requires careful design
  - Wide range of libraries enabling high performance and rapid development.
- Disadvantage
  - Very easy to develop sloppy and inefficient code!!!!!!!!

The University Of Sheffield.

The University of Sheffield Research IT

REFERENCES

- http://www.cplusplus.com
- Standard template libraries
  - http://www.sgi.com/tech/stl/table_of_contents.html
- Bjarn Stroustrups homepage
  - http://www.research.att.com/~bs/homepage.html
- Documentation for Sun Compilers
  - http://developers.sun.com/prodtech/cc/compilers_index.html

The University Of Sheffield.

The University of Sheffield Research IT

CODE RESPOSITORIES

- Github
- http://sourceforge.net/
- http://www.netlib.org/
- http://archive.devx.com/sourcebank/
- http://www.boost.org/