CS 201
Introduction to c++ (1)

Debzani Deb

History of C++

- Extension of C (C++ for better C)
- Early 1980s: Bjarne Stroustrup (Bell Laboratories)
- Provides capabilities for object-oriented programming
  - Objects – reusable software components : model things in real world
  - Object-oriented programs : Easy to understand, correct and modify
- Hybrid language
  - C-like style
  - Object-oriented style

What is C++?

- General purpose, biased towards system programming:
  - A better C
  - Supports data abstraction
  - Supports object-oriented programming
  - Supports generic programming.
- C++ programs
  - Built from pieces called classes and functions.
- C++ standard Library
  - Rich collection of existing classes and functions.

Data abstraction (Encapsulation)

- Define program entity in terms of related data.
- Define operations on entity.
- Separate implementation of program from its data structures (data hiding via object interfaces).
- Program relies more on abstract properties of classes, than on properties of particular objects.

Object oriented …

- Extends data abstraction.
- Facilitates code reuse through inheritance.
- Runtime dynamic binding.
- C++ is not “truly object-oriented”
  - Hybrid approach
  - Not everything is a class.

Major Domains of C++ Applications

- Operating system kernels and components
- Device drivers
- Real-time/ deterministic systems
- Critical systems
- Parallel computing
- Numerical calculations
- 3D games and graphics

A better C ..

- Symbolic constants
- Inline code submission
- Default function arguments
- Function and operator overloading
- References
- Namespace management
- Exception handling
- Templates

C++ Syntax features (1)

- Symbolic constants
  ```
  const int arraySize = 10;
  int a [arraySize];
  ```
- Inline functions: C++ compiler places the function inline rather than generate the code for calling the routine. Eliminates the overhead of function calling in case of small functions.
  ```
  inline double cube (const double s) { return s*s*s; }
  ```
- Default function arguments
  ```
  void foo (int a, int b =0) { ...}
  foo(a); /* called function foo with arguments (a,0) */
  ```


C++ Syntax features (2)

- Declaration inside the loop control
  ```
  for( int i = 0; i <max; i++) {... }
  ```
  i is undefined outside the loop.
- A reference is an alias (an alternative name) for an object. When you initialize a reference with an object, you *bind* that reference to that object.
  ```
  int num1 = 10;
  int num2 = 20;
  int & RefOne = num1; // valid
  int & RefOne = num2; // error, two definitions of RefOne
  int & RefTwo; // error, uninitialized reference
  int & RefTwo = num2; // valid
  ```


C++ Syntax features (3)

- Pass-by-reference in C++
  - Passing the address of an argument in the calling function to a corresponding parameter in the called function. In C, the corresponding parameter in the called function must be declared as a pointer type. In C++, the corresponding parameter can be declared as reference type.

```
void swapnum (int & i, int & j) {
  int temp = i; i = j; j = temp;
}
int main(void) {
  int a = 10; int b = 20;
  swapnum (a, b);
  return 0;
}
```

```
void swapnum(int *i, int *j) {
   int temp = *i;
   *i = *j;
   *j = temp;
} // C Syntax
```

```
swapnum(&a, &b); // C Syntax
```


A simple C++ program (1)

- Input/output
  - cin: standard input stream, normally keyboard
  - cout: standard output stream, normally screen
  - cerr: standard error stream, display error messages
- Simple C++ program

```
// myFirst.cpp
#include <iostream>
int main () {
   std :: cout << "Welcome to C++! \n";
   return 0;
}
```

Preprocessor directive to include input/output stream header file <iostream>

Stream insertion operator

Binary scope resolution operator

Name cout belongs to namespace std

A simple C++ program (2)

```
// myFirst.cpp
#include <iostream>
using namespace std;
int main () {
   cout << "Welcome to C++! \n";
   return 0;
}
```

Library file names in namespace std do not require a .h extension

- Namespace
  - std :: cout is removed through use of **using** statements which specifies that we will be using objects (such as cout) located in a special region/namespace called **std**.
  - C++ standard library is defined in the namespace **std**. So every program generally include using namespace std.

Namespace (1)

- Namespaces allow to group entities like classes, objects and functions under a single name.
- This way the global scope can be divided in "sub-scopes", each one with its own name.
- The format of namespaces is:

  namespace identifier { entities }
  - where identifier is any valid identifier and entities is the set of classes, objects and functions that are included within the namespace. For example:

  namespace myNamespace { int a, b; }
  - In order to access these variables from outside the myNamespace namespace we have to use the scope operator ::. For example, myNamespace :: a

Namespace (2)

- The functionality of namespaces is especially useful in the case that there is a possibility that a global object or function uses the same identifier as another one, causing redefinition errors. For example:

```
#include <iostream>
using namespace std;
namespace first { int var = 5; }
namespace second { double var = 3.1416; }
int main () {
  cout << first :: var;
  cout << second::var;
  return 0;
}
```

In this case, there are two global variables with the same name: var. One is defined within the namespace first and the other one in second. No redefinition errors happen thanks to namespaces.

Namespace (3)

- The keyword using is used to introduce a name from a namespace into the current declarative region. For example:

```
#include <iostream>
using namespace std;
namespace first { int x = 5; int y = 10; }
namespace second { double x = 3.1416; double y = 2.7183; }
int main () {
    using first::x;
    using second::y;
    cout << x;
    cout << y;
    cout << first::y;
    cout << second::x;
    return 0;
}
```

Notice how in this code, x (without any name qualifier) refers to first::x whereas y refers to second::y, exactly as our using declarations have specified.

We still have access to first::y and second::x using their fully qualified names.

Output:
```
5
2.7183
10
3.1416
```


Namespace (4)

- using and using namespace have validity only in the same block in which they are stated or in the entire code if they are used directly in the global scope. For example, if we had the intention to first use the objects of one namespace and then those of another one, we could do something like:

```
#include <iostream>
using namespace std;
namespace first { int x = 5; }
namespace second { double x = 3.1416; }
int main () {
  {
    using namespace first;
    cout << x;
  }
  {
    using namespace second;
    cout << x;
  }
  return 0;
}
```

Output:
```
5
3.1416
```


Structure and their Applicability (1)

- New data types are built by using elements of other types.

```
struct Time {
  int hour;
  int minute;
  int second;
};
```

Structure members

- Structure member naming
  - In same structure must have unique name.
  - Different structure can have members of same name.

Structure and their Applicability (2)

- Structure member can be pointer to instance of enclosing struct (self-referencial structure) – used for linked lists, queues, stack and trees.
- Structure member can not be instance of enclosing structure.
- Few uses:

```
Time timeObject;
Time timeArray[10];
Time *timePtr;
Time &timeRef = timeObject;
```

- Member access operator
  - Dot (.) for structure/class members
  - Arrow (->) for pointer to structure/class.`

Implementing user defined type Time with a structure

- C-style structure
  - No "interface"
    - ➢If implementation of this data structure changes, all programs using that structure must change accordingly.
  - Can not print as a whole
    - ➢Must print/format member by member
  - Can not compare entirely
    - ➢Must compare member by member.

Implementing Time abstract data type with a class

- Classes in C ++
  - Model objects
    - ➢Attributes (data members)
    - ➢Behaviors (member functions)
  - Defied by using keyword class
- Member access specifiers
  - Public:
    - ➢Accessible whenever object of the class is in scope
  - Private
    - ➢Accessible only within the member functions of a class
  - Protected:

Implementing Time abstract data type with a class

- Constructor Function
  - Special member function
    - ➢Initializes data members
    - ➢Same name as the class
    - ➢Called when object of a class is instantiated
    - ➢Can have several constructor with different number and type of parameters
      - – Function overloading
    - ➢N return type

Class Time definition

```
class Time {

public:
    Time();                          // constructor
    void setTime (int h, int m, int s ); // set hr, min and sec
    void printUniversal ();          // in universal time format
    void printStandard ();           // in standard time format

private:
    int hour;                        // 0 .. 23
    int minute;                      // 0 .. 59
    int second;                      // 0 .. 59
};
```


Implementing Time ADT with a class

- Objects of a class
  - After class definition
    - ➢Class name can be treated as new type specifier.
    - ➢Objects, arrays, pointers and references of a specific class can be declared.
    - ➢Examples:

```
Time sunset;                  // object of type Time
Time arrayOfTimes[ 10 ];      // Array of Time objects
Time *pointerToTime;          // Pointer to a Time object
Time &dinnerTime = sunset;    // Reference to a Time object
```


Implementing Time ADT with a class

- Member Functions defined outside class
  - Binary scope resolution operator (::)
    - ➢Ties member name to class name
    - ➢Uniquely identify functions of particular class
    - ➢Different classes can have members of same name.
  - Format for defining member functions

ReturnType ClassName :: MemberFunctionName (Arguments) { ....}

Does not change whether function is declared as public or private.

- Member Functions defined inside class
  - Do not need a scope resolution operator or a class name to be specified.

```
// Time Class
#include <iostream>
using std:: cout;
class Time {                              // Time ADT definition
public:
    Time();                               // constructor
    void setTime (int h, int m, int s );  // set hr, min and sec
    void printUniversal ();               // in universal time format
    void printStandard ();                // in standard time format
private:
        int hour;                         // 0 .. 23
        int minute;                       // 0 .. 59
        int second;                       // 0 .. 59
};                                        // End of Class Time
```



```
// Time constructor initializes each data member to zero and ensures
    that all Time objects start with a consistent state.
Time :: Time() {
    hour = minute = second = 0;
}
// setTime() sets new Time values using universal method, performs
    validity checks on the data values and set invalid values to zero.
void Time :: setTime (int h, int m, int s) {
    hour = ( (h >= 0) && (h < 24) ) ? h : 0;
    minute = ( (m >= 0) && (m < 60) ) ? m : 0;
    second = ( (s >= 0) && (s < 60) ) ? s : 0;
} // end function setTime()
..... // other member function definitions here
int main() {
    Time t; // instantiate object t of class Time
    ....
}
```


Advantages of using Class

- Simplify programming
- Interfaces
  - Hide implementation
- Software Reuse
  - Aggregation
    - Class objects can be members of other classes.
  - Inheritance
    - New classes derived from old.

Key features of C++

- Classes , single/multiple inheritance
- Streamed I/O
- Exceptions
- Templates
- Namespaces
- Run-time type information
- Standard library

C++ vs. Java

• Hybrid	• Everything is a class
• Real machine code	• Byte code
• Minimal standard library	• Massive APIs
• Memory management	• Garbage collection
• Single/multiple inheritance	• Single inheritance
• Easy to access low level system information	• Difficult to access system information.

Important !!!

- Take home exercise will be given in the next class.
- Last class – Quiz 4
- All lab related problem should be taken care of by this Friday. After that, nothing can be done.