Chapter 1

Objective

- C++ History, why and how C++ was developed.
- How C++ improves its code/program design.
- How object oriented concepts are used in C++.
- Key elements of C++ programming language.
- How to use input/output in C++.
- Scope of variables and object in C++.

Unit 1

History Of C++

- Developed by Bjarne Stroustrup in early 80's
- Derived from BCPL and Simula67
- Class concept taken from Simula
- Designed for efficiency and object oriented concepts.

Originally C++ was called C with classes. The Classes concept was taken from Simula and added to C, but later new and improved features were added which were enhancements to the C language. Due to the added features C with classes was renamed as C++.

Improved Program Design

C++ is an Object-oriented programming language that supports following concepts:

Modularity:

•C++ is designed in a very modular and compartmentalized manner.

•Each and every element work like a miniature program.

Extensibility:

•C++ programs can easily be enhanced or modified because code is modularized.

•Modularity of C++ code improves quality and makes the addition of new code very simple.

Reusability:

•Inheritance concept provides code reusability when programming in C++.

•The key characteristic of the C++ language is to build class libraries for code reusability purpose.

The significant feature of C++ language is its object-oriented concepts. Such as:

•Encapsulation of data with user defined functions.

•Incremental development

•Code reusability

Unit 2

Object Oriented Concepts

- Object oriented programming makes the computer language resemble the way things look in the real world.
- Object oriented concepts encapsulate real world objects because they have characteristics and behavior.
- Object oriented language must satisfy following three criteria's:
  - Data abstraction
  - Inheritance
  - Polymorphism

Data Abstraction

- The combination of the data and the operations (functions) on the data creates new data type, called data abstraction.
- An abstract data type behaves just like built in C/C++ data type, except the programmer defines the type.

See example on next page.

```
0   #include <iostream>   // Example 1-1
1   struct Date {
2       int m_month;
3       int m_day;
4       int m_year;
5       void show();    // member function
6   };
7   int main()
8   {
9      Date hire;    // No need to use struct before date
10     hire.m_month = 3;
11     hire.m_day = 14;
12     hire.m_year = 80;
13     hire.show();
14   }
15
16   //Show is a member function of Date (user defined data type)
17   void Date::show()
18   {
19    std::cout << m_month << "/" << m_day << "/" << m_year << "\n";
20   }
```

Output: 3/14/80

Inheritance

- Source code is organized in a hierarchical ordering.
- Hierarchical ordering is designed using class concept.
- Each class completely represents a single concept.
- You specify the relationship between parent (base) and child (sub) classes.

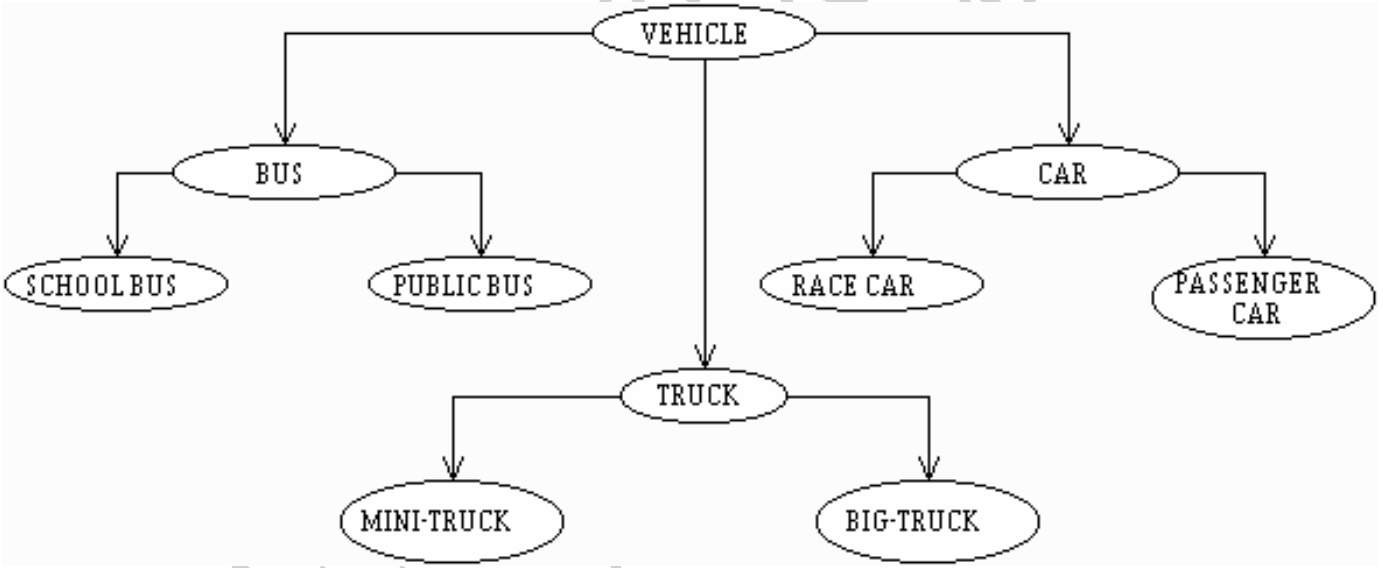

Figure 1

Polymorphism

- Multiple implementation single interface.
- It allows a single operation (method) to behave differently with each object.
  - For example, operation **drive** may be defined to move a car from one location to another; same operation **drive** can be used for truck that characterizes the state of a truck.

- It simplifies the syntax for performing the same task differently based on a type of object.

Unit 3

Key Elements Of C++ Language

Object-oriented programming mainly centers around concepts. There are three key elements that describe the object-oriented phenomena.

- Classes
- Objects
- Methods

Classes

•Keyword "**class**" declares a new user-defined type.
•The central concept of object-oriented programming "encapsulation" is implemented within the class.
•A class specifies the behavior and creation of its object.
•Class contains a complete description of one kind of object.
•Class can produce as many copies of object as needed.
•Objects are representative (instances) of a class.
•Classes can be organized hierarchically with subclasses. (**see figure 1**)

Classes (Continued)

- Class declares data member and member function same as structure.

```
class CDate {
        void Init();
        void Show();
        int m_nMonth;
        int m_nDay;
        int m_nYear;
};
```

- The subtle difference between class and structure lies in the visibility of members.
- The class members default to **private**, and structure member's default to **public**.
- C++ keywords **private** and **public** invokes data member or member function according to the visibility of declaration.
- **Private** members can not be accessed directly through an instance of the class or a structure.
- **Public** members can be accessed using dot operator in relation to an instance of the class or structure.
- Once either **public** or **private** keyword is used in a class or a structure, all members will be **public** or **private** unless otherwise reset to **public** or **private**.

See example on next page.

Classes (Continued)

```
0   #include <iostream>       // Example 1-2
1   class CDate {
2     public:
3       void Init();
4       void Show();
5
6     private:
7       int m_nMonth;
8       int m_nDay;
9       int m_nYear;
10  };
11
12  int main()
13  {
14    CDate Hire;
15    Hire.Init();
16    Hire.Show();
17  }
```

Classes (Continued)

```
18
19   void CDate::Init()
20   {
21     m_nMonth = 9;
22     m_nDay = 17;
23     m_nYear = 92;
24   }
25
26   void CDate::Show()
27   {
28    std::cout << m_nMonth << "/" << m_nDay << "/" << m_nYear << "\n";
29   }
```

OUTPUT:

```
9/17/92
```

Objects

- An object is a variable of a user-defined type "**class**".
- It encapsulates data members and member function to performs some actions.
- Any thing that has characteristic and behavior is considered an object.
- No need to know the internals of other objects.
- Objects are not scalar

Methods

- Methods are the member functions of a class.
- These member functions process messages for objects.

Comments Delimiters In C++

- There are two ways to comment the C++ source code.
  a. /* This is C style commenting. */
  b. // This is new way to comment in C++ programs.

// Use following header in your programs to separate functions from each other.

```
/*****
* Function: ProcessOrder
* Purpose:
*       To process the purchase orders for overseas
*       customers.
* Parameters:
*       pCost: total purchase cost.  <Passed by reference>.
*       nInvoiceId: purchase invoice id
*                number.  <Passed by value>.
* Local Variables:
*       nDollar: Holds amount to purchase
*                office equipment.
*       nItem : Holds office equipment item purchased.
******/
```

Unit 4

Quick View Of Input/Output Streams

- Commonly known as iostreams or streams.
- This provides all the functionality of **stdio.h** library in C.
- A stream refers to the flow of data from producer to consumer.
- Classes are provided to support input/output streams.

Facilities	Header Files	Classes
Console Output	constrea.h	conbuf, constream
Memory Buffers	iostream.h	ios, iostream, iostream_withassign,istream, istream_withassign, ostream, ostream_withassign, streambuf.
Files	fstream.h	filebuf, fstream, ifstream, ofstream, fstreambase.
Strings	strstrea.h	istrstream, ostrstream, strstream, strstreambase, strstreambuf.

Figure 2

Quick View Of Input/Output Streams (Continued)

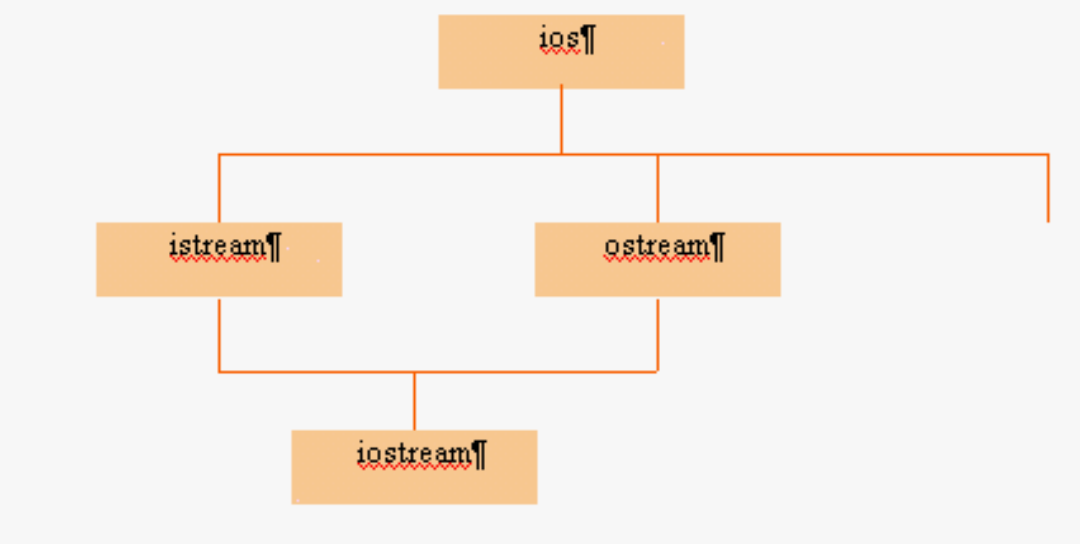

Figure 3

Stream Objects and Operators

- Stream output is accomplished with insertion operator <<
- Stream input is accomplished with extraction operator >>
- To redirect the output **cout** object is used from ostream class
- The **cout** object interprets different data types correctly.
- The **cerr** object is used to output error messages to screen.
- Using **cerr** object programs can have their standard output redirected to another file or device while error messages are sent to console.
- To read data from the keyboard **cin** object is used from istream class.

Insertion Operator Usage

```
0   // Display message
1   #include <iostream>   // Example 1-3
2   int main()
3   {
4     std::cout << "Welcome to C++\n";
5   }
6
```

OUTPUT:

```
Welcome to C++
```

Extraction Operator Usage

```
0   // Convert inches into centimeter, Prompt for inches input
1   #include <iostream>   // Example 1-4
2   void main()
3   {
4     int nInch = 0;
5     std::cout << "inches =";
6     std::cin >> nInch;
7     std::cout << nInch << " in = ";
8     std::cout << nInch * 2.54 << " cm\n";
9   }
10
```

OUTPUT:

```
inches = 12
12 in = 30.48 cm
```

Usage Of cerr object

```
0 // error.cpp    (redirect error message to consol)
1 #include <iostream>     // Example 1-5
2 int main()
3 {
4    char cAlpha = 'A';
5
6    if (cAlpha == 'B')
7        std::cout << "Selected Alphabet Is: " << cAlpha << "\n";
8    else {
9       std::cerr << "Letter A and B are not same\n";
10       std::cout << "Current Alphabet is: " << cAlpha << "\n";
11   }
12 }
13
```

Figure 1.3 redirects the output to a outfile

```
C:\>error > outfile
```

OUTPUT:

Letter A and B are not same // This output goes direct to the screen

Current Alphabet is: A // This output goes into outfile

Declarations and Scope Of Identifiers

- Declaration of a variable can be done any place within a block.
- Every identifier declared has a scope.
- The scope of the declared variable extends to the end of the block.
- Scope is the portion of a program in which identifiers can be used to access data objects.

See example on next page.

Unit 5

Declarations and Scope Of Identifiers (Continued)

```
0 #include <iostream>     // Example 1-6
1 void Show();    // Function Prototype
2 int nCats = 20;    // Global declaration    (not a good idea)
4 int main()
5 {
6   std::cout << "total cats = " << nCats << "\n";    // nCats = 20
7   int nCats = 10;
8   {
9     int nCats = 30;
10     std::cout << "Block cats = " << nCats << "\n";  // nCats = 30
11     std::cout << "global cats = " << ::nCats << "\n"; // nCats = 20
12  } // end of block
13
14  std::cout << "main cats = " << nCats << "\n";    // nCats = 10
15  Show();
16 } // end main function
17
```

Declarations and Scope Of Identifiers (Continued)

```
18 void Show()
19 {
20     std::cout << "function cats = " << nCats << "\n";    // nCats = 20;
21 }
```

OUTPUT:

```
total cats = 20
Block cats = 30
global cats = 20
main cats = 10
function cats = 20
```

Declaring Identifier Within for Loop

Initialization part of the “for loop” can be used as declarations in C++.

```
0   #include <iostream>   // Example 1-7
1   int main()
2   {
3     for (int i = 0, j = 10; i < 5; i++, j++)
4        std::cout << "i = " << i << "j = " << j << "\n";
5   }
6
```

OUTPUT:

```
i = 0; j = 10
i = 1; j = 11
i = 2; j = 12
i = 3; j = 13
i = 4; j = 14
```

Declaring Identifier Within for Loop

```
0   // Nested for loop
1   #include <iostream>   // Example 1-8
2   int main()
3   {
4     for (int nRow = 0; nRow < 6; nRow++) {
5       std::cout << nRow << ":";
6       for (char cAlpha = 'A'; cAlpha < 'A' + 6; cAlpha++)
7         std::cout << cAlpha;
8       std::cout << "\n";
9     } // end of for loop
10  } // end of main
```

OUTPUT:

```
0:ABCDEF
1:ABCDEF
2:ABCDEF
3:ABCDEF
4:ABCDEF
5:ABCDEF
```