Introduction (Lecture - 1)

Welcome to Object-Oriented

Object Oriented are the programming languages that provide tools for implementing high cohesion and loose coupling in the program of the application which makes the code more modular and easy to collaborate.

C++ is an object oriented programming language which provides the efficiency of C language with the object oriented features of a modern language that makes it high in efficiency and performance.

Addressing Complexity

Large applications are complex, to address this, we identify the most important feature of the problem and examine it.

Programming Languages

C, C++, Java, Python, etc. are currently the most popular programming languages.

The difference between C, C++ and Java is that C is not object-oriented, Java is purely object-oriented and C++ is a hybrid of both the languages.

Features of C++

- nearly a perfect superset of C.
- a multi-paradigm language
    - procedural language (code written as a sequence of instructions - line by line)
    - object-oriented language (code divided into classes and objects)
- realistic, effective and flexible enough for big demanding projects (large applications, game programming, operating system)
- clean enough for presenting basic concepts
- comprehensive enough for presenting advanced programming concepts

Type Safety

C++ is a type-safe language which means that it finds/traps syntax errors at compile-time which diminishes the amount of buggy code that escapes at the end. C compilers are more tolerant to type errors while comparing with C++ compilers.

Namespaces

A namespace is a container that holds a collection of identifiers (such as variable names, function names, or class names) and helps avoid naming conflicts between different parts of a program. Namespaces are used to organize and manage the scope and visibility of these identifiers.

Defining a namespace:

```
namespace identifier {

 }
```

The identifier after the `namespace` keyword is the name of the scope. The pair of braces encloses and defines the scope.

For example, to define x in two separate `namespaces` (english and french), we write

```
namespace english {

    int x = 2;
    // ...

 }

 namespace french {

    int x = 3;
    // ...

 }
```

To access a variable defined within a namespace, we precede the variable identifier with the namespace's identifier and separate them with a double colon (::). We call this double colon the scope resolution operator.

For example, to increment the `x` in namespace `english` and to decrement the `x` in namespace `french`, we write:

```
    english::x++;
    french::x--;
```

Each prefix uniquely identifies each variable's namespace.

Namespaces hide their entities. To expose an identifier to the current namespace, we insert the using declaration into our code before referring to the identifier.

For example, to expose one of the `x`'s to the current namespace, we write:

```
using french::x;
```

After which, we can simply write:

```
x++; // increments french::x but not english::x
```

To expose all of a namespace's identifiers, we insert the `using` directive into our code before referring to any of them.

For example, to expose all of the identifiers within namespace `english`, we write:

```
using namespace english;
```

Afterwards, we can write:

```
x++; // increments english::x but not french::x
```

Exposing a single identifier or a complete namespace simply adds the identifier(s) to the hosting namespace.

Common uses of namespaces:

By far the most common use of namespaces is for classifying

- struct-like types

- function types

Writing C++ code for a C programmer:

The the following example we will display a phrase “Welcome to Object-Oriented” in C as well as C++ language.

C - procedural code:

```
#include <stdio.h>

int main(void)
{
    printf("Welcome to Object-Oriented\n");
}
```

The two functions - main() and printf() - specify activities. These identifiers share the global namespace.

C++ - Procedural code:

```
#include <cstdio>
using namespace std;

int main ( ) {
    printf("Welcome to Object-Oriented\n");
}
```

The file extension for any C++ source code is .cpp. `<cstdio>` is the C++ version of the C header file `<stdio.h>`. This header file declares the prototype for printf() within the std (standard) namespace.

The directive `using namespace std;` exposes all of the identifiers declared within the std namespace to the global namespace. The libraries of standard C++ declare most of their identifiers within the std namespace.

C++ - hybrid code:

```
#include <iostream>
using namespace std;

int main ( ) {
    cout << "Welcome to Object-Oriented" << endl;
}
```

The object-oriented syntax consists of:

1. The directive `#include <iostream>` inserts the `<iostream>` header file into the source code. The `<iostream>` library provides access to the standard input and output objects.
2. The object `cout` represents the standard output device.
3. The object `<<` inserts whatever is on its right side into whatever is on its left side.
4. The manipulator `endl` represents an end of line character along with a flushing of the output buffer. Note the absence of a formatting string. The `cout` object handles the output formatting itself.

That is, the complete statement `cout << "Welcome to Object-Oriented" << endl;` inserts into the standard output stream the string `"Welcome to Object-Oriented"` followed by a newline character and a flushing of the output buffer.

First Input and Output Example

The following object-oriented program accepts an integer value from standard input and displays that value on standard output:

```
#include <iostream>
using namespace std;

int main() {
    int i;

    cout << "Enter an integer : ";
    cin >> i;
    cout << "You entered " << i << endl;
}
```

Output

```
Enter an integer : 65
You entered 65
```

The object-oriented input statement includes:

1. The object `cin` represents the standard input device.
2. The extraction operator `>>` extracts the data identified on its right side from the object on its left-hand side.
3. Note the absence of a formatting string. The `cin` object handles the input formatting itself.
4. That is, the complete statement `cin >> i;` extracts an integer value from the input stream and stores that value in the variable named `i`.
5. The type of the variable `i` defines the rule for converting the text characters in the input stream into byte data in memory. Note the absence of the address of operator on `i` and the absence of the conversion specifier, each of which is present in the C language.