C++

Session #4

#4

- Variables
- Class
- Overloading

Variables

A variable provides us with named storage that our programs can manipulate. Each variable in C++ has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

Variables

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C++ is case-sensitive −

Variables

- Bool
- Char
- Int
- Float
- Double
- Void
- wchar_t

Variables

The primitives types work the same way as available in C, this is the reason why C libraries can be imported into a C++ program.

Variables

You have also noticed that C++ has other types of variable like “string” that you have already used on your previous program.

This type can be re-implemented by using those primitives.

Variables

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type, and contains a list of one or more variables of that type as follows

```
type variable_list;
```

Variables

or

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

Variables

or

```
extern int d = 3, f = 5;   // declaration of d and f.
int d = 3, f = 5;          // definition and initializing d and f.
byte z = 22;               // definition and initializes z.
char x = 'x';
```

Variables

When you create a variable, you MUST assign a value as soon as possible simply because variable might use a “dirty” part of your memory.

Scope

A scope is a region of the program and broadly speaking there are three places, where variables can be declared:

- Inside a function or a block which is called local variables,
- In the definition of function parameters which is called formal parameters.
- Outside of all functions which is called global variables.

Scope

- Local Variables

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code.

Scope

- Global Variables

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the life-time of your program.

Constants/Literals

Constants refer to fixed values that the program may not alter and they are called literals.

Constants can be of any of the basic data types and can be divided into Integer Numerals, Floating-Point Numerals, Characters, Strings and Boolean Values.

Again, constants are treated just like regular variables except that their values cannot be modified after their definition.

Constants/Literals

Example: integer literals

```
212         // Legal (int)
215u        // Legal (unsigned int)
0xFeeL      // Legal (hexadecimal)
078         // Illegal: 8 is not an octal digit
032UU       // Illegal: cannot repeat a suffix
```

Constants/Literals

Constants can be defined within 2 way:

- Using **#define** preprocessor.
- Using **const** keyword.

Constants/Literals

Following is the form to use #define preprocessor to define a constant

#define identifier value

Constants/Literals

You can use const prefix to declare constants with a specific type as follows

```
const type variable = value;
```

Constants/Literals

You can use const prefix to declare constants with a specific type as follows

```
const type variable = value;
```

Class

The main purpose of C++ programming is to add object orientation to the C programming language and classes are the central feature of C++ that supports object-oriented programming and are often called user-defined types.

Class

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

Class

```
class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};
```

Class

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box:

```
Box Box1;          // Declare Box1 of type Box
Box Box2;          // Declare Box2 of type Box
```

Class

- **Class Member Functions**
  A member function of a class is a function that has its definition or its prototype within the class definition like any other variable.
- **Class Access Modifiers**
  A class member can be defined as public, private or protected. By default members would be assumed as private.
- **Constructor & Destructor**
  A class constructor is a special function in a class that is called when a new object of the class is created. A destructor is also a special function which is called when created object is deleted.

Class

- **Copy Constructor**
  The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously.
- **Friend Functions**
  A friend function is permitted full access to private and protected members of a class.
- **this Pointer**
  Every object has a special pointer this which points to the object itself.
- **Pointer to C++ Classes**
  A pointer to a class is done exactly the same way a pointer to a structure is. In fact a class is really just a structure with functions in it.
- **Static Members of a Class**
  Both data members and function members of a class can be declared as static.

Class

Canonical class Implementation:

- Cls::Cls();
- Cls::Cls(const Cls& rhs);
- Cls& Cls::operator=(const Cls& rhs);
- ~Cls::Cls();

Overloading

C++ allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.

Overloading

When you call an overloaded function or operator, the compiler determines the most appropriate definition to use, by comparing the argument types you have used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called overload resolution.

Overloading

Function Overloading in C++:

You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You cannot overload function declarations that differ only by return type.

Overloading

```
class printData {
   public:
      void print(int i) {
        cout << "Printing int: " << i << endl;
      }
      void print(double  f) {
        cout << "Printing float: " << f << endl;
      }
      void print(char* c) {
        cout << "Printing character: " << c << endl;
      }
};
```

Overloading

Operators Overloading in C++

You can redefine or overload most of the built-in operators available in C++. Thus, a programmer can use operators with user-defined types as well.

Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

Box operator+(const Box&);

Overloading

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []