Operators

Objectives of this session

- Operators in C++
- Scope Resolution Operator
- Memory Management Operator
- Manipulators
- Type Cast Operator

Operators in C++

New operators in C++ :

- << Insertion Operator
- >> Extraction Operator
- : : Scope Resolution Operator
- : : * Pointer-to-member declaration
- ->* Pointer-to-member Operator
- .* Pointer-to-member Operator
- delete Memory Release Operator
- endl Line Feed Operator
- new Memory Allocation Operator
- setw Field Width Operator

All C operators are valid in C++

Scope Resolution Operator

```
.........
.........
{
    int x = 10;
    .........
    .........
}
.........
.........
{
    int x = 1;
    .........
    .........
}
```

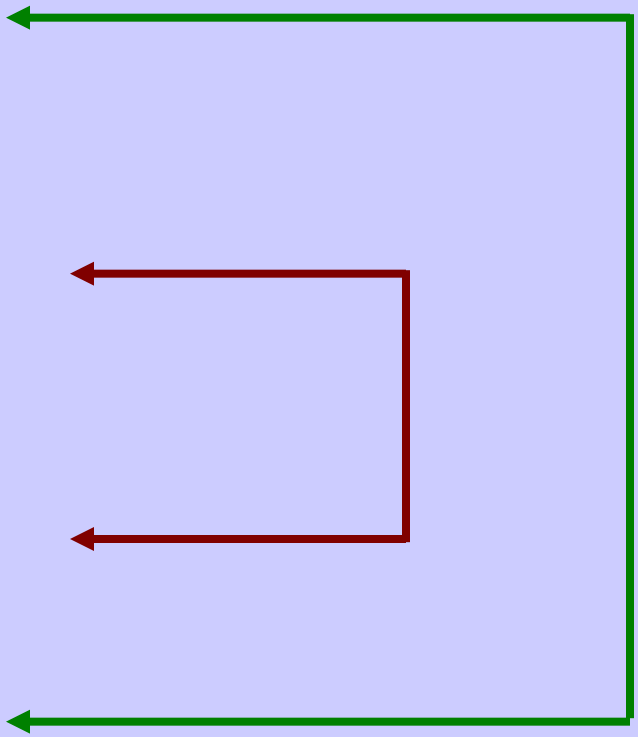
Scope Resolution Operator

continue…

```
.....
.....
{
    int x = 10;
    .....
    .....
    {
        int x = 1;
        .....
        .....
    }
    .....
    .....
}
```

Block1

Block 2



Scope Resolution Operator

continue…

The scope resolution operator (::) can be used to uncover a hidden variable.

: : variable-name

This operator allows access to the global version of a variable.

Scope Resolution Operator

continue…

```
#include<iostream.h>
#include<conio.h>
int m = 10;
void main()
{
   int m = 20;
   clrscr();
   cout << "m_local          = " << m << "\n";
   cout << "m_global         = " <<::m << "\n";
   getch();
}
```

```
m_local          = 20
m_global         = 10
```

Memory Management Operators

malloc( ) and **calloc( )** functions are used to allocate memory dynamically at run time.

The function **free( )** to free dynamically the allocated memory.

C & C++

The unary operators **new** and **delete** perform the task of allocating and freeing the memory.

C++

Memory Management Operators

continue…

- **new** to create an object
- **delete** to destroy an object

A data object created inside a block with **new**, will remain in existence until it is explicitly destroyed by using **delete**.

Thus the life time of an object is directly under our control and is unrelated to the block structure of the program.

Memory Management Operators

continue…

- new – create an object

pointer-variable = **new** *data-type*;

The *data-type* may be any valid data type

pointer-variable is a pointer of type *data-type*

The **new** operator allocates sufficient memory to hold a data object of type *data-type* and returns the address of the object

The pointer-variable holds the address of the memory space allocated

Memory Management Operators

continue…

- *pointer-variable* = **new** *data-type;*

```
p = new int;      // p is a pointer of type int
q = new float;    // q is a pointer of type float
```

Here p and q must have already been declared as pointers of appropriate types.

Alternatively, we can combine the declaration of pointers and their assignments as:

```
int *p = new int;
float *q = new float;
```

Memory Management Operators

continue…

```
int *p = new int;
float *q = new float;
*p = 25;  // assign 25 to the newly created int object
*q = 7.5; // assign 7.5 to the newly created float object
```

pointer-variable = **new** *data-type (value);*

```
int *p = new int ( 25 );
float *q = new float ( 7.5 );
```

Memory Management Operators

continue…

new can be used to create a memory space for any data type including user-defined types such as arrays, structures and classes.

pointer-variable = **new** *data-type [size];*

int *p = new int [10];

When creating multi-dimensional arrays with new, all the array sizes must be supplied.

array_ptr = new int[3][5][4]; //legal

array_ptr = new int[m][5][4]; //legal

array_ptr = new int[3][5][ ]; //illegal

array_ptr = new int[ ][5][4]; //illegal

Memory Management Operators

continue…

- **delete** to destroy an object

delete *pointer-variable;*

When a data object is no longer needed, it is destroyed to release the memory space for reuse.

```
delete p;
delete q;
```

delete *[size] pointer-variable;*

The size specifies the number of elements in the array to be freed.

```
delete [ ]p;  // delete the entire array pointed to by p
```

Memory Management Operators

continue…

If sufficient memory is not available for allocation, malloc( ) and new returns a null pointer.

```
…….
…….
p = new int;
if (!p)
{
   cout << “Allocation failed \n”;
}
…….
…….
```

Memory Management Operators

continue…

Advantages of new over malloc( ):

- It automatically computes the size of the data object. No need to use sizeof operator.
- It automatically returns the correct pointer type. No need to use type cast.
- It is possible to initialize the object while creating the memory space.
- Like any operator, new and delete can be overloaded.

Manipulators

Manipulators are operators that are used to format the data display.

Commonly used manipulators are:

- endl // causes a line feed when used in an
  // output statement

- setw // to specify field width and force the
  // data to be printed right-justified

Manipulators

continue…

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
void main( )
{
 int m, n, p;
 m = 2597;
 n = 14;
 p = 175;
 clrscr( );

 cout <<setw(10) << "First = " <<setw(10) << m << endl
      <<setw(10) << "Second = " << setw(10) << n << endl
      <<setw(10) << "Third = " << setw(10) << p << endl;
 getch( );
}
```

```
    First =      2597
   Second =        14
    Third =       175
```

Manipulators

continue…

We can create our own manipulators as follows:

```
# include < iostream.h>
ostream & symbol (ostream & output)
{
    output << "\tRs. ";
    return output;
}
```

Manipulators

continue…

```
#include<iostream.h>
#include<conio.h>

ostream & symbol (ostream & output)
{
 output << "Rs. ";
 return output;
}

void main( )
{
 clrscr( );

 cout << symbol << "5,000/-"  <<endl;
 getch( );
}
```

Type Cast Operator

C++ permit explicit type conversion of variables or expressions using the type cast operator.

- (type-name) expression // C notation

- type-name (expression) // C++ notation
  // like a function call
  // notation

eg:- average = sum /(float) i; // C notation

average = sum / float(i); // C++ notation

Type Cast Operator

continue…

p = int * (q); // is illegal

The type name should be an identifier

p = (int *) q; // is legal

Alternatively, we can use typedef to create an identifier of the required type.

typedef int * int_pt;

p = int_pt (q);

Thank You