Basic program constructs in C++

Introduction to C++

C++ is an object oriented programming language. It was developed by Bjarne Stroustrup at AT and T Bell Lab USA. It was developed on the base of C programming language and the first added part to C is concept of ***Class***. Hence originally this language was named “C with class”. Later name was changed to C++ on the basis of idea of increment operator available in C. since it is the augmented or incremented version of C, it is named C++.

The first C++ program

Following is a sample program in C++ that prints a string on the screen(monitor).

```
#include<iostream.h>
int main()
{
        cout<<"Hi Everybody";
        return 0;
}
```

Like in C, functions are the basic building block in C++. The above example consists of a single function called main ().

When a C++ program executes, the first statement that executes will be the first statement in main () function. The main function calls member functions of various classes (using objects) to carry out the real work. It may also call other stand-alone functions.

In C++, the return type of the main function is ‘int’. So, it returns one integer value to the operating system. Since the return type int is default, the keyword ‘int’ in main () is optional. So,

```
        main ()
        {
           …….// also valid
        }
```

Compilers generate error or warning if no value is returned. Many operating systems test the return values. If the exit value is zero (0), the operating system will understand that the program ran successfully. If the returned value (exit value) is non zero, it would mean that there was problem.

Comment syntax

In C++, comments start with a double slash (//) symbol and end at the end of the line. A comment may start at the beginning of a line or anywhere in the line and whatever follows till the end of that line is ignored. There is no closing symbol. If we need to comment multiple lines, we can write as

```
// this is an
// example
// of multi line comments
```

The C comment style /*…………*/ may also be used for multi line comments.

The output operator (Output using "*cout*")

The statement: *cout<<"Hi Everybody";* in the above example prints the phrase in quotation marks on the screen.

Here, the identifier 'cout', pronounced as "see out", is a predefined object of standard stream in C++. The operator << is called 'insertion' or 'put to' operator. It inserts the content on its right to the object on its left.

```
cout<<a;
```

In the above case, the statement will display the content of the variable 'a'.

The input operator(input using *cin*)

A statement

```
cin>>a;
```

is an input statement. This causes the program to wait for the user to type and give some input. The given input is stored in the variable a.

Here, the identifier 'cin', pronounced as 'see in' is an object of standard input stream. The operator '>>' is called 'extraction' or 'get from' operator. It extracts or gets value from keyboard and assigns it to the variable on its right.

Cascading I/O operators(multiple input/output)

The i/o operators can be used repeatedly in a single i/o statements as follows.

```
cout<<a<<b<<c;
cin>>x>>y>>z;
```

These are perfectly legal. The above cout statement first sends the value of 'a' to cout, then sends the value of b and then sends the value of c. Similarly, the cin statement first reads a value and stores in x, then reads again and stores in y and then in z. The multiple uses of i/o operators in one statement is called cascading.

The iostream header file

The directive '#include<iostream.h>' causes the preprocessor to add the contents of iostream.h file to the program. It contains the declarations of identifiers cout, cin and the operators << and >>. So, the header file iostream should always be included at the beginning if we need to use cin, cout, << and >> operators in our program.

Tokens: Tokens are the smallest individual units in a program. Keywords, identifiers, constants, strings and operators are tokens in C++.

Keywords: Keywords are explicitly reserved identifiers and can not be used as names for the program variables or other user-defined program elements. Some keywords are **int**, **auto**, **switch**, **case**, **do**, **else**, **public**, **default**, **continue** etc.

Functions
A function is a single comprehensive unit that performs a specified task. This specified task is repeated each time the function is called. Functions break large programs into smaller tasks. They increase the modularity of the programs and reduce code redundancy.
Like in C, C++ programs also should contain a main function, where the program always begins execution. The main function may call other functions, which in turn will again call other functions.
When a function is called, control is transferred to the first statement of the function body. Once the statements of the function get executed (when the last closing bracket is encountered) the program control return to the place from where this function was called.

Data types in C++

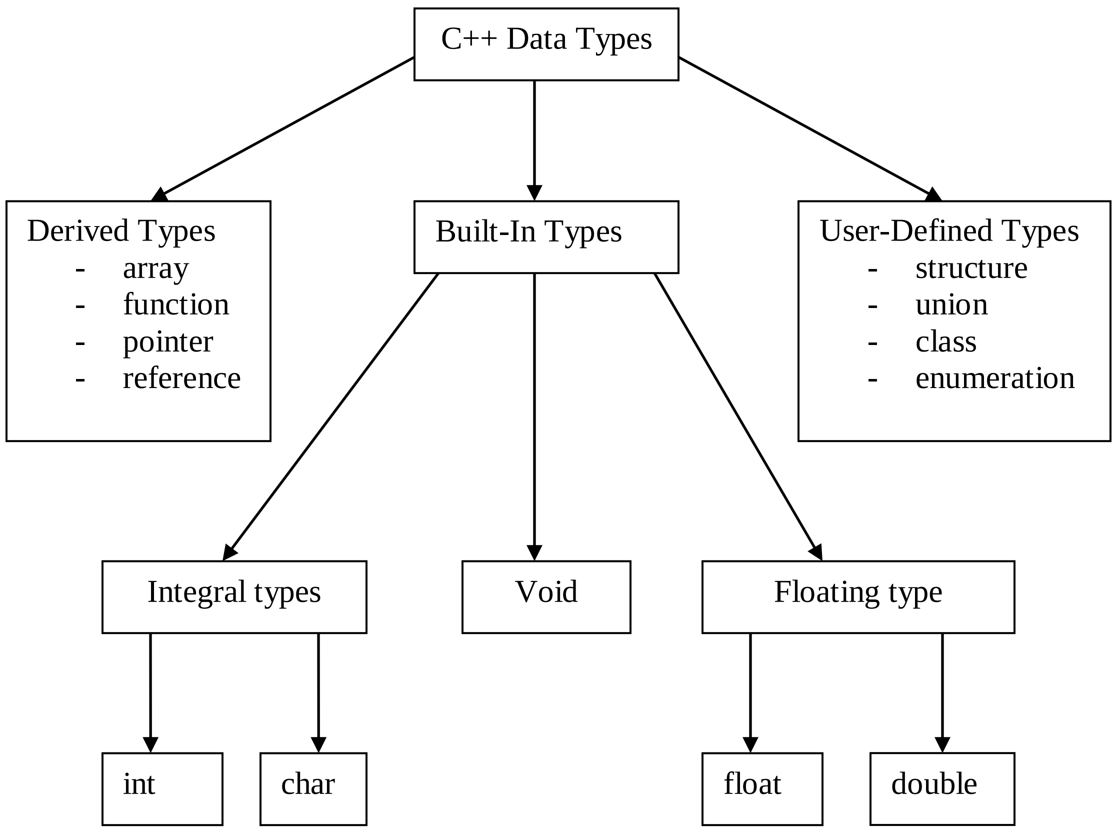


Fig. Hierarchy of C++ Data types

Enumerated Data Types
Like structures, enumerated data type is another user defined data type. Enumerated means that all the values are listed. They are used when we know a finite list of values

that a data type can take on or it is an alternative way for creating symbolic constants. The ‘enum’ keyword automatically lists a list of words and assign them values 0,1,2…

```
Eg.  enum shape  {circle, square, triangle};
     enum Boolean  {true, false};
     enum switch  {on, off};
```

The above example is equivalent to

```
        const  circle = 0;
        const square = 1;
        const triangle = 2;
```

We can even use standard arithmetic operator on enum types. We can also use relational operators when suitable. This is because, the enum data types are internally treated as integers.
Once we specify the data type, we need to define variables of that type.

```
        Eg. shape s1,s2;
```

Now the variables s1, s2 can hold only the members of ‘shape’ data type (those are circle, square and triangle) and can not hold anything except these values. If other values are given, error will be generated.

An example

```
#include<iostream.h>
#include<conio.h>
enum days {sun, mon, tue, wed, thur, fri, sat};
void main()
{
        days d1,d2;
        d1 = sun;
        d2 = thur;
        int diff = d2-d1;  // using arithmetic operator
        cout<<”Days between”<<diff<<endl;
        if(d1<d2)   // using relational operator
                cout<<”d1 comes first”;
        getch();
}
```

Manipulators

The manipulators are operators used with insertion operator “<<’ to format or manipulate the data display. ‘endl’ and ‘setw’ are most common manipulators.

endl manipulator causes a linefeed to be inserted into the output stream. i.e the cursor moves to next line. It is similar to ‘\n’ character.
Eg-

```
                cout<<”Kathmandu”<<endl;
                cout<<”Nepal”;
```

See output

setw manipulator specify a field width to a number or string that follows it and force them to be printed right justified. The field width is given as an argument to this manipulator.
Eg-

```
x = 456; y = 40;
cout<<setw(5)<<x<<setw(5)<<y;
```

The manipulator will specify a field 5 for printing the value of x. The value is right justified within the fields as shown below. For y, it will specify again space of width 5, right justifies and prints.

		4	5	6

x

			4	0

y

Setprecision() The setprecision ( ) is used to control the number of digits of an output stream display of a floating point value. The setprecision ( ) manipulator prototype is defined in the header file <iomanip.h>.

The general syntax of the setprecision manipulator is

Setprecision (int p)

Which sets the precision for floating point insertions to p. The default precision is 6

PROGRAM

A program to use the setprecision manipulator function while displaying a floating point value onto the screen.

```
/ /using setprecision manipulator

#include <iostream.h>

#include <iomanip.h>

void main (void)

{

  float  a,b,c;

  a = 5;

  b = 3;
```

```
    c = a/b;

    cout << setprecision (1) << c << endl;

    cout << setprecision (2) << c << endl;

    cout << setprecision (3) << c << endl;

    cout << setprecision (4) << c << endl;

    cout << setprecision (5) << c << endl;

    cout << setprecision (6) << c << endl;

  }
```

Output of the program

```
  1.7

  1.67

  1.667

  1.6667

  1.66667

  1.666667
```

Type Conversion (Data Conversion)

We use the assignment operator (=) to assign value of one variable to another. For example

```
x = y;
```

Where x and y are integer variables. We have also noticed that = assigns the value of one user defined object to another, provided that they are of the same type. For example,

```
d2 = d1;
```

Normally, when the value of one object is assigned to another of the same type, the values of all the member data items are simply copied into the new object. The compiler does not need any special instructions to use = for the assignment of user-defined objects such as distance objects.

The assignments between types, whether they are basic types or user-defined types, are handled by the compiler with no effort on our part, provided that the same data type is used on both sides of the equal sign.

But if the variables on different sides of the = are of different types, then the type of variable on the right side of = needs to be converted to the type of left side variable before the assignment takes place.

Type conversion is the conversion of one data type to another data type.

Conversion Between Basic Types

Consider the statement,

```
intvar = floatvar;
```

where intvar is of type int and floatvar is of type float. Here the compiler will call a special routine to convert the value of floatvar, which is expressed in floating point format, to an integer format so that it can be assigned to intvar. There are many such conversions: from **float** to **double**, **char** to **float** and so on. Each such conversion has its own routine, built into the compiler and called up when the data types on different sides of the = sign so dictate. Such conversions are **implicit conversion** or **automatic conversion.**

Sometimes we want to force the compiler to convert one type to another. This is known as **explicit** or **type conversion**. For example,

```
int total = 400;
float avg;
avg = float(total) / 5;  // converts value of total to float before division takes place.
```

Arrays in C++ Programming

Array is a collection of data of same types stored in sequential memory location. It is a linear data structure, where data is stored sequentially one after the other. The elements in an array is accessed using an index. For example, In an array of n elements, the first element has index *zero* and the last element has index *(n-1)*. Elements with consecutive index (i.e. i and i+1) are stored in consecutive memory location in the system.

Array can be divided into following types:

One Dimensional Array

Multi Dimensional Array

One Dimensional Array

An array in which data are arranged linearly in only one dimension is called one dimensional array. It is commonly known as 1-D array

Syntax and Declaration of One Dimensional Array

datatype array_name[size];

Here, array_name is an array of type datatype and the number of elements in array_name is equal to size.

For example,

int x[10]; // declares an integer array with 10 elements

float arr[5]; // declares an float array with 5 elements

char n[50]; // declares an character array with 50 elements

Arrangement of One Dimensional Array

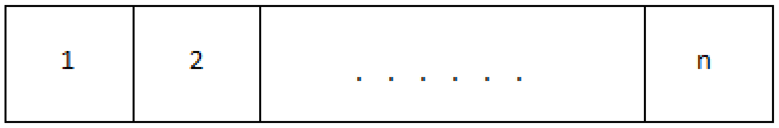


Fig: One Dimensional Array

Example of One Dimensional Array

C++ program to ask 10 numbers from user and display the sum.

```
#include <iostream>
#include <conio.h>
using namespace std;

int main()
{
   int arr[10],sum=0,i;
   cout<<"Enter 10 numbers"<<endl;
   for(i=0;i<10;i++)
   {
      cin>>arr[i];
```

```
        sum = sum+arr[i];
    }
    cout<<"Sum = "<<sum;
    getch();
    return 0;
}
```

Here, a one dimensional array *arr* of size 10 is declared. Ten numbers are entered by user which is stored in the array arr. Then the sum of these numbers is calculated using a for loop. Finally the sum is displayed outside the loop.

Output

```
Enter 10 numbers
13
52
4
-41
32
11
19
7
2
25
Sum = 124
```

Multi-Dimensional Array

An array in which data are arranged in the form of array of arrays is called multi-dimensional array. An array can have as much dimensions as required. However, two dimensional and three dimensional array are commonly used.

Syntax and Declaration of Multi-Dimensional Array

datatype array_name[d1][d2]...[dn];

Here, array_name is an array of type datatype and it has n dimensions. The number of elements in a multi dimensional array is equal to the product of size of all dimensions i.e. total number of elements in array array_name is d1*d2* ... dn.

Two Dimensional Array

Two dimensional array is where the data is stored in a list containing 1-D array.

Syntax and Declaration of Two Dimensional Array

datatype array_name[d1][d2];

Here, array_name is an array of type datatype and it has 2 dimensions. The number of elements in array_name is equal to d1*d2.

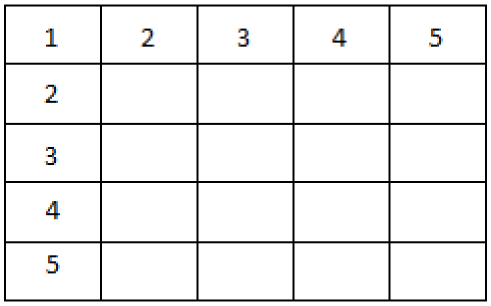
For example,

int a[10][10]; // declares an integer array with 100 elements

float f[5][10]; // declares an float array with 50 elements

char n[5][50]; // declares an character array with 250 elements

Arrangement of Two Dimensional Array



1	2	3	4	5
2				
3				
4				
5				

Fig: Two dimensional Array

Example of Two Dimensional Array

C++ program to enter elements of a matrix and display them.

```
#include <iostream>

#include <conio.h>

using namespace std;


int main()
```

```
{
   int arr[10][10],row,col,i,j;
   cout<<"Enter size of row and column: ";
   cin>>row>>col;
   cout<<"Enter elements of matrices(row wise)"<<endl;
   for(i=0;i<row;i++)
      for(j=0;j<col;j++)
         cin>>arr[i][j];
   cout<<"Displaying matrix"<<endl;
   for(i=0;i<row;i++)
   {
      for(j=0;j<col;j++)
         cout<<arr[i][j]<<"  ";
      cout<<endl;
   }
   getch();
   return 0;
}
```

In this program, a two dimensional array is used to store the content of a matrix. The size of row and column is entered by user. Nested for loop is used to ask the content of elements of matrix and display them. The number of elements in the array (matrix) is equal to the product of size of row and column.

Output

```
Enter size of row and column: 2 3

Enter elements of matrices(row wise)

12 31 51

19 13 24

Displaying matrix

12  31  51
```

```
19  13  24
```

Example of Three Dimensional Array

C++ program to show the concept of three dimensional array.

```
#include <iostream>
#include <conio.h>
using namespace std;

int main()
{
   int arr[10][10][10],d1,d2,d3,i,j,k;
   cout<<"Enter size of three dimensions: ";
   cin>>d1>>d2>>d3;
   cout<<"Enter elements of array"<<endl;
   for(i=0;i<d1;i++)
      for(j=0;j<d2;j++)
         for(k=0;k<d3;k++)
         {
            cout<<"a["<<i<<"]["<<j<<"]["<<k<<"] = ";
            cin>>arr[i][j][k];
         }
   cout<<"Displaying elements of array"<<endl;
   for(i=0;i<d1;i++)
      for(j=0;j<d2;j++)
         for(k=0;k<d3;k++)
            cout<<"a["<<i<<"]["<<j<<"]["<<k<<"] = "<<arr[i][j][k]<<endl;
   getch();
   return 0;
}
```

This example show how data are stored and accessed from a three dimensional array. The values of size of three dimensions: d1, d2 and d3 are entered by user. According to these values, a nested loop is created to enter the value of elements of array and display them. The outermost loop runs d1 times, middle loop runs d2 times and the innermost loop runs d3 times.

Output

```
Enter size of three dimensions: 3 2 2

Enter elements of array

a[0][0][0] = 113

a[0][0][1] = 2

a[0][1][0] = 91

a[0][1][1] = 14

a[1][0][0] = 56

a[1][0][1] = 71

a[1][1][0] = 30

a[1][1][1] = 23

a[2][0][0] = 51

a[2][0][1] = 67

a[2][1][0] = 219

a[2][1][1] = 641

Displaying elements of array

a[0][0][0] = 113

a[0][0][1] = 2

a[0][1][0] = 91

a[0][1][1] = 14

a[1][0][0] = 56

a[1][0][1] = 71

a[1][1][0] = 30

a[1][1][1] = 23

a[2][0][0] = 51
```

a[2][0][1] = 67

a[2][1][0] = 219

a[2][1][1] = 641

Pointers:

Pointer is a variable that stores memory addresses. Unlike normal variables it does not store user given or processed value, instead it stores valid computer memory address.

Pointer allows various magical things to be performed in the programs

- Pointers are more efficient in handling arrays and structures.
- Pointers are used to return multiple values from a function.
- Pointer allows dynamic memory allocation and deallocation (creation and deletion of variables at runtime) in C/C++. Which undoubtedly is the biggest advantage of pointers.
- Pointer allows to refer and pass a function as a parameter to functions.

Syntax to declare pointer variable

<data-type> * <variable-name>

Example of pointer declaration

```
int * integer_pointer;

float * float_ptr

char * charPtr;

long * lptr;
```

Using Pointers in C++

There are few important operations, which we will do with the pointers very frequently. (a) We define a pointer variable. (b) Assign the address of a variable to a pointer. (c) Finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. Following example makes use of these operations −

```
#include<iostream>
using namespace std;
int main()
{int a=25,*p;
p=&a;
cout<<"\nThe value of a is: "<<a;
cout<<"\nThe address of a is:"<<&a;
cout<<"\nThe address p is holding is:"<<p;
cout<<"\nThe value pointed by p is: "<<*p;
}
```

Dynamic program using pointer

```
#include<iostream>
using namespace std;
int main()
{int i,n,*p;
cout<<"How many numbers would you like to input?";
cin>>n;
for(i=0;i<n;i++)
{
   cout<<"Enter number:";cin>>p[i];
}
cout<<"You have entered:";
for(i=0;i<n;i++)
   cout<<p[i]<<",";
}
```

New and delete operators in C++

C uses malloc() and calloc() function to allocate memory dynamically at run time and uses free() function to free dynamically allocated memory. C++ supports these functions and also has two operators **new** and **delete** that perform the task of allocating and freeing the memory in a better and easier way.

Syntax to use new operator: To allocate memory of any data type, the syntax is:

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

Allocate block of memory: new operator is also used to allocate a block(an array) of memory of type *data-type*.

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

Syntax:

// Release memory pointed by pointer-variable

delete pointer-variable;

To free the dynamically allocated array pointed by pointer-variable, use following form of *delete*:

// Release block of memory pointed by pointer-variable

delete[] pointer-variable;

Example:

```
  // It will free the entire array pointed by p.
  delete[] p;
```

Dynamic Program using new and delete operators

```
#include<iostream>
using namespace std;
int main()
{int i,n;
cout<<"How many numbers would you like to input?";
cin>>n;
int *p=new int[n];
for(i=0;i<n;i++)
{
   cout<<"Enter number:";cin>>p[i];
}
cout<<"You have entered:";
for(i=0;i<n;i++)
   cout<<*(p+i)<<",";
delete[] p;
}
```

Constant:

Constant is something that doesn't change. In C language and C++ we use the keyword const to make program elements constant. const keyword can be used in many contexts in a C++ program. It can be used with:

- Variables
- Pointers
- Function arguments and return types
- Class Data members
- Class Member functions
- Objects

Constant Variables in C++

If you make any variable as constant, using const keyword, you cannot change its value. Also, the constant variables must be initialized while they are declared.

```
int main
{   const int i = 10;
   const int j = i + 10;     // works fine
   i++;    // this leads to Compile time error
}
```