Chapter 3

Tokens, Expression & Control Structure

Tokens

As we know, the smallest individual; units in a program are known as Tokens. C++ has following Tokens.

1. Keyword
2. Identifier
3. Constants
4. String
5. Operators

1) Keywords

The keywords implements specific C++ language features. They are explicitly reserved identifier (Variable) and can not be used as names for program variable or other user define program elements.

asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	register	typedef
class	friend	return	union
const	goto	short	unsigned
continue	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

2) Identifier

Identifier are refers to the name of variable, function, array, structure, class etc. created by the programmer. Each language has its own rules for naming these identifiers. C program recognize only the first 32 characters in a variable name but in C++ places no limits on its length.

Basic Data Type

C++ complier supports all the built in data type (basic or fundamental). Data types in C++ can be classified under various categories as shown in Fig.

The type Void was introduced in C programming. The normal use of void are

1. To specify return type of a function when it is not returning any value.
2. To indicate an empty argument list to the function.

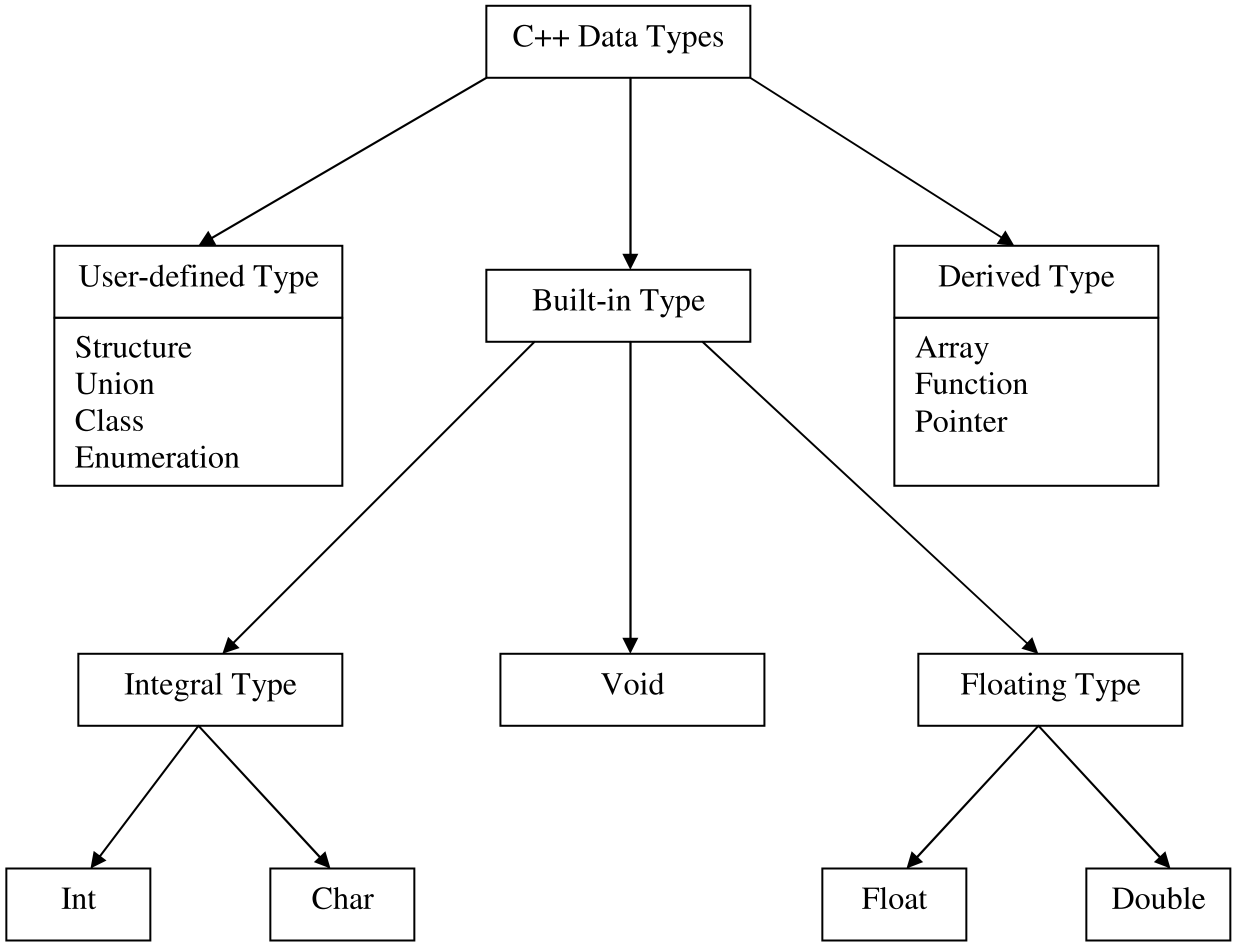

Fig. Hierarchy of C++ Data types

Type	**Bytes**	**Range**
Char	1	-128 to 128
Unsigned char	1	0 to 255
Signed char	1	-128 to 127
Int	2	-32768 to 32767
Unsigned int	2	0 to 65535
Signed int	2	-32768 to 32767
Short int	2	-32768 to 32767
Unsigned short int	2	0 to 65535
Signed short int	2	-32768 to 32767
Long int	4	-2147483648 to 2147483647
Signed long int	4	-2147483648 to 2147483647
Unsigned long int	4	0 to 4294967295
Float	4	3.4E – 38 to 3.4E + 38
Double	8	1.7E – 308 to 1.7E + 308
Long double	10	3.4E – 4932 to 1.1E + 4932

Table Size and range of C++ basic data types

User Defined Data Type

a) Structure & Classes

We have used user defined data type such as structure in C. While this data types are legal in C++. Some more features have been added to make suitable for object oriented programming. C++ also permits us to another user defined data type known as class which can used just like any other basic data types, to declare variable. The class variable is known as objects, which are the central focus of OOP'S.

b) Enumerated Data Type

An enumerated data type is another user defined data type, which provides a way for attaching names to numbers, thereby increasing comprehensibility of the code. The enum keyword automatically enumerate a list of words by assigned the value 0,1,2,…. The syntax of enum statement is similar to that of structure statement.

e.g.

```
enum shape
{
   rectangle,
   triangle
   square
};
```

Write a Program to Demonstrate Enum Data Type.

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

enum shape
{
  rectangle,
  triangle,
  square
};

main()
{
int code;
clrscr();
cout<<"Enter the code = ";
cin>>code;
switch(code)
{
case rectangle:
           int area,br,l;
           cout<<"Enter length & bradth = ";
           cin>>l>>br;
           area=l*br;
```

```
            cout<<"Area = "<<area;
            break;
case triangle:
            int h,ba;
            float area1;
            cout<<"Enter height & base = ";
            cin>>h>>ba;
            area1=0.5*ba*h;
            cout<<"Area = "<<area1;
            break;
case square:
            int s,area2;
            cout<<"Enter the side = ";
            cin>>s;
            area2=s*s;
            cout<<"Area = "<<area2;
            break;
}
getch();
return(0);
}
```

Derived Data Type

a) Array

The application of array in C++ is similar to that in C. The array is the way character arrays are initialized.

b) Function

Functions have undergone major changes in C++. While some of these changes are simple, others require a new way of thinking when organizing our programs. Many of these modifications improvements were driven by the requirement of the object oriented concept of C++. Some of these were introduced to more the C++ program more reliable & readable.

c) Pointer

In general programs, whenever variable is used in any statement to access the its value, we have to search a address by the variable name. The large amount of data is to be access like array. Every time the variable search the address. To reduce this time pointer can be used. A pointer is kind of variable that holds of another variable.

3) Constants

In both C & C++, any value declared as constant can not be modify by the program in any way. The name constants are just like variables except that there values can not be change. C++ requires a constant to initialized.

4) String

Strings are one dimensional arrays of type char. In C++, a string is terminated by a null character or '/0'. String constants are written in double quotation.

5) Operators

Like C, C++ has a rich set of operators. All the C operators are valid in C++.

a) Mathematical Operators

+ , - , * , / , % (modulus operator), ^ (exponential operator)

b) Relational Operators

< , > , <= , >= , = = , != (not equal to)

Note

= is used for assignment operator.
= = is used for comparison of two quantities.

c) Logical Operators

&& (and) , || (or) , ! (not)

In addition to this C++ introduced some new operators there are as follows.

1) : : - Scope resolution operator
2) : : * - Pointer-to-member declarator
3) * - Pointer-to-member operator
4) . * - Pointer-to-member operator
5) new – Memory allocation operator
6) delete - Memory released operator
7) endl – new line operator
8) setw – set width operator

1) Scope Resolution Operator

Like C language C++ also a block structured language. Blocks are generally used while constructing the program. The scope of variable changes from block to block. A variable declared inside a block is called as Local Variable. A single variable with the same name can be declared in different block of program.

If we want to access the global variable C++ provides the Scope resolution operator which represents as ": :". The general form of this operator is

: : Variable name

Write a Program to Demonstrate of Scope Resolution Operator.

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

int m=10;
main()
{
  int m=20;
  {
     int m=30;
     cout<<"Value of M = "<<m;
     cout<<"Gobal Value = "<<::m;
  }
  cout<<"Value of M = "<<m;
  cout<<"Gobal Value = "<<::m;
  getch();
  return(0);
}
```

<u>Manipulators</u>

Manipulators are operators that are used to format the data display. The most commonly used manipulators are endl & setw. The endl manipulator when use an O/P statement causes a line feed to be inserted. It has the same effect as using the new line character "\n". The setw manipulator is used to print the massage & force them to right justified.

Write a Program to Demonstrate of Manipulators.

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

main()
{
int b=1000,h=200,d=300;
cout<<setw(5)<<"Basic"<<setw(5)<<b<<endl;
cout<<setw(5)<<"Da"<<setw(5)<<d<<endl;
cout<<setw(5)<<"Hra"<<setw(5)<<h<<endl;
getch();
return(0);
}
```

<u>2) New & Delete (Memory Management Operators)</u>

The operators new & delete perform the task of allocate & free the memory in a better & easier way. Since these operators manipulate memory on free store, they are also known as free store operators. The new operator can be used for allocate the memory of any data type. The general form is as follows.

Pointer variable = new data type

Here pointer variable is a pointer of type data type. The new operator allocates sufficient memory to hold a data object of type data type & returns the address of the object. The data type may be any valid data type. The pointer variable holds the address of the memory space allocated.

There are some advantages of new operator over the malloc function.

1. It automatically computes the size of the data object. There is no need to use the size of operator.
2. It automatically returns the correct pointer type. There is no need to use type casting.
3. Like any other operator new & delete can be overloaded.
4. It is possible to initialize the object while creating the memory space.

We can also initialize the memory using the new operator is as follows.

```
        Pointer name = new data type (Value)
```

e.g.

```
        1) p = new int;
           q = new char;
        2) p = new int(33);
           q = new float(3.63);
```

When variable is no longer needed, it is destroyed to release the memory space for reuse. The general form is as follows.

```
        Delete Pointer variable;
```

The pointer variable is the pointer that points to a data objects created with the new operator.

e.g.

```
        int *p;
        p = new int;
        . . . .
        delete p;
```

Control Structure

In C++, a large number of programs are used that pass messages and process the data contained in objects. A function is set up to perform a task. The format of program should be easy to trace the flow of execution of statements.

1) If Statement

The if Statement is implemented in three forms.

a) Simple If Statement

Syntax 1)

```
if (Condition)
{
  Execute Statement;
}
```

2)

```
if (Condition)
{
  Group of Statement 1;
}
else
{
  Group of Statement 2;
}
```

b) Multiple If Statement

Syntax

```
if (Condition 1)
   Statement 1;
else
     if (Condition 2)
          Statement 2;
:
:
else
    if (Condition n)
        Statement n;
    else
        Statement x;
```

c) Nested If Statement

Syntax

```
if (Condition 1)
{
      Group of Statement 1;
       if (Condition 2)
          Group of Statement 2;
       else
          Group of Statement 3;
}
else
{
      Group of Statement 4;
}
```

<u>2) Do-While Loop</u>

The do-while is an exit-controlled loop. Based on a condition, the control is transferred back to a particular point in the program.

Syntax

```
do
{
    Group of Statement;
     Increment Loop Counter;
}while (Condition);
```

3) While Loop

This is also a loop structure, but is an entry-controlled one.

Syntax

```
Initialize Loop Counter
while (Condition)
{
      Group of Statement;
      Increment Loop Counter;
}
```

4) For Loop

The for is an entry-enrolled loop and is used when an action is to be repeated for a predetermined number of times.

Syntax

```
for (Initialization;Condition;Increment/Decrement)
{
      Group of Statement;
}
```

5) Switch Statement

This is a multiple-branching statement where, based on a condition, the control is transferred to one of the many possible points.

Syntax

```
switch (Integer Expression)
{
     case Constant 1:
                  Group of Statement;
                  break;
     case Constant 2:
                  Group of Statement;
                  break;
     :
     :
     :
     default:
             Default Statement
}
```

Type Case Operator

C++ permits explicit type conversion of variables using the type cast operator. A type-name behaves as if it is a function for converting values to a designed type. The function call notation usually leads to simplest expressions.

Syntax

```
(Type-name)Variable name;        C Notation
Type-name(Variable name);        C++ Notation
```

e.g.

```
int a=10;
float b=1000.5,sum;
sum=b+(float)a;            C Notation
sum=b+float(a);            C++ Notation
```

Operator Precedence

Although C++ enables us to add multiple meaning to the operators, their association and precedence remain the same. For example, the multiplication operator will continue having precedence then the add operator. The following table gives the precedence and associativity of all the C++ operators.

Table :- Operator precedence and associativity

Operator	**Associativity**
: :	Left to Right
-> . ( ) [ ] postfix ++ postfix --	Left to Right
prefix ++ prefix -- ~ ! unary + unary – unary * unary & (type) sizeof new delete	Left to Right
->* *	Left to Right
* / %	Left to Right
+ -	Left to Right
<< >>	Left to Right
< <= > >=	Left to Right
= = !=	Left to Right
&	Left to Right
^	Left to Right
|	Left to Right
&&	Left to Right
||	Left to Right
?:	Left to Right
= *= /= %= += -= <<= >>= &= ^= |=	Left to Right
,	Left to Right

Expressions and Implicit Conversion

An expression is a combination of operators, constants and variables. An expression may consist of one or more operands and zero or more operators to produce a value. Expression may be of four types, namely, Constant expression, Integral expression, Float expression and Pointer expression.

Constant expression consist of only constant values.

e.g.

5
20 + 5 / 2
'x'

Integral expression are those produce integer results after implementing all the automatic and explicit type conversion.

e.g.

m
m * n – 5
5 + int(2.0)

Where m and n are integer variables.

Float expression are those which, after all conversion, produce floating-point results.

e.g.

x + y
5 + float(10)
10.75

Where x and y are float variables.

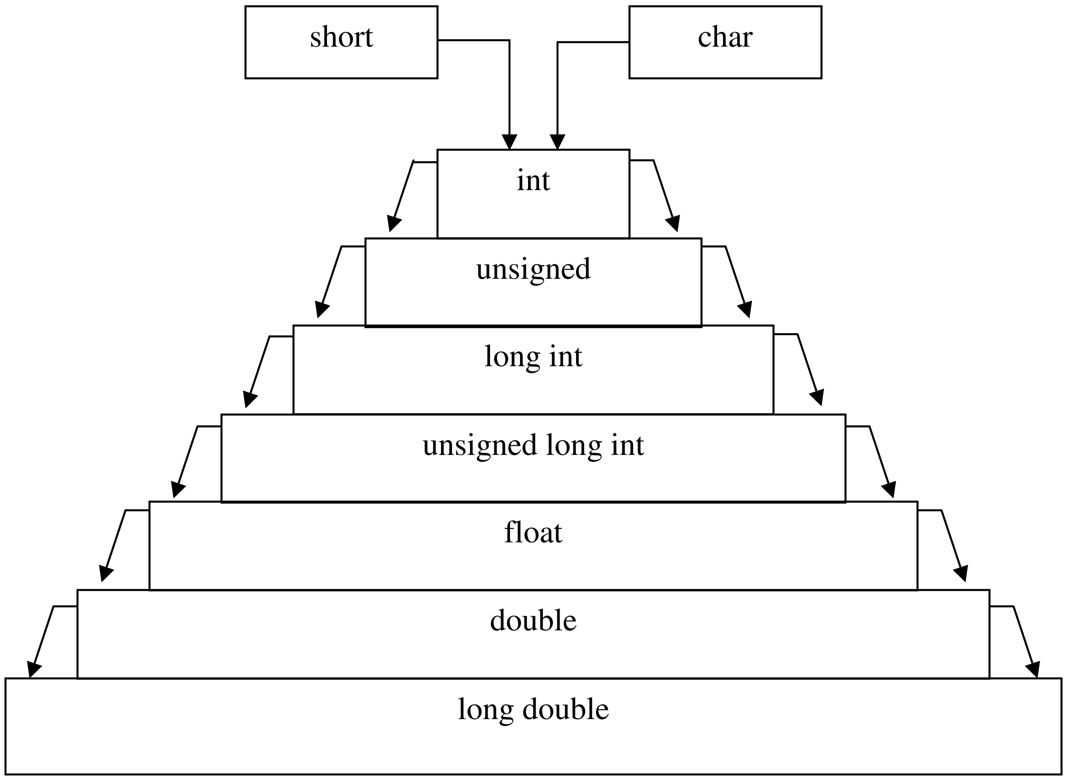

Fig. Water-fall model of type conversion

Pointer expression produce address values.

e.g.

&m
ptr + 1

Where m is variable & ptr is pointer.

Wherever data types are mixed in an expression, C++ performs the conversions automatically. This process is known as Implicit or Automatic conversion.

For a binary operator, if the operand type differ, the complier convert one of them to match with the other using a certain rule. The rule is that the "smaller" type is converted to the "wider" type. The "water-fall" model shown in above fig. illustrates this rule.

Write a Program to Check Given is Even or Odd.

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

main()
{
int no;
clrscr();
cout<<"Enter the Value = ";
cin>>no;
int r;
r=no%2;
if(r==0)
  cout<<"Number is Even";
else
  cout<<"Number is Odd";
getch();
return(0);
}
```

Write a Program to Print Following Output.

```
1
2  2
3  3   3
4  4   4   4
```

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

main()
{
int a=1,b=1;
clrscr();
for(a=1;a<5;a++)
{
   for(b=1;b<=a;b++)
      cout<<a<<" ";
   cout<<"\n";
}
getch();
return(0);
}
```

Write a Program to Find Largest Value in Given 3 Value.

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

main()
{
int a,b,c;
clrscr();
cout<<"Enter the 3 Value = ";
cin>>a>>b>>c;
if(a>b && a>c)
  cout<<"A is Large";
else
  if(b>a && b>c)
    cout<<"B is Large";
  else
    cout<<"C is Large";
getch();
return(0);
}
```

Write a Program to Calculate Factorial of Given Number.

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

main()
{
int no;
clrscr();
cout<<"Enter the Value = ";
cin>>no;
int fact=1;
for(int a=1;a<=no;a++)
   fact=fact*a;
cout<<"Factorial = "<<fact;
getch();
return(0);
}
```

Write a Program to Find Reverse no of Given Number.

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

main()
{
int no,rev=0,r;
clrscr();
cout<<"Enter the no = ";
cin>>no;
while(no!=0)
{
  r=no%10;
  rev=rev*10+r;
  no=no/10;
}
cout<<"Reverse No = "<<rev;
getch();
return(0);
}
```