C/C++ language

DIT Part 1st

CHAPTER 1:
INTRODUCTION

MAIN TOPIC COVERED

- HISTORY C / C++ LANGUAGE
- DIFFERENCES B/W C AND C++
- STRUCTURE OF C PROGRAM
- CHARACTERISTICS OF C
- CHARACTER SET OF C
- CONSTANTS & VARIABLES
- ESCAPE SEQUENCES
- INPUT / OUTPUT FUNCTION
- FORMAT SPECIFIRES
- COMMENTS

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

INTRODUCTION

History of C

C is a general-purpose, procedural, computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system.

During the 60s, while computers were still in an early stage of development, many new programming languages appeared. Among them, ALGOL 60, was developed as an alternative to FORTRAN but taking from it some concepts of structured programming which would later inspire most procedural languages, such as CPL and its successors (like C++). ALGOL 68 also influenced directly in the development of data types in C. Nevertheless ALGOL was an unspecific language and its abstraction made it little practical to solve most commercial tasks.In 1963 the CPL (Combined Programming language) appeared with the idea of being more specific for concrete programming tasks of that time than ALGOL or FORTRAN. Nevertheless this same specificity made it a big language and, therefore, difficult to learn and implement. In 1967, Martin Richards developed the BCPL (Basic Combined Programming Language) that signified a simplification of CPL but kept the most important features the language offered. Although it continued being an abstract and some what large language.

The initial development of C occurred at AT&T Bell Labs between 1969 and 1973 It was named "C" because many of its features were derived from an earlier language called "B," which was version of the BCPL programming language.

The development of UNIX was the result of programmers desire to play the *Space Travel* computer game. They had been playing it on their company's mainframe, but as it was underpowered and had to support about 100 users, Thompson and Ritchie found they did not have sufficient control over the spaceship to avoid collisions with the wandering space rocks. This led to the decision to shift the game to an idle PDP-7 computer in the office. As PDP-7 lacked an operating system, so the two scientists set out to develop one, based on several ideas from their colleagues. Eventually it was decided shift the operating system from PDP-7 to the office's PDP-11computer , but faced with the taught task of translating a large amount of custom-written assembly language code from PDP-7 TO PDP-11, the programmers began considering using a portable, high-level language so that the OS could be shifted easily from one computer to another. They looked at using B, but it lacked functionality to take advantage of some of the PDP-11's advanced features. This led to the development of an early version of the C programming language.

The original version of the UNIX system was developed in assembly language. Later, nearly all of the operating system was rewritten in C, a dangerous move at a time when nearly all operating systems were written in assembly. By 1973, the C language had become powerful enough that most of the Unix kernel, originally written in PDP-11 assembly language, was rewritten in C. This was one of the first operating system kernels implemented in a language other than assembly. (Earlier it was Multics system (written in PL/I), and MCP (Master Control Program) for the Burroughs B5000 written in ALGOL in 1961.)“The C Programming Language" by Brian Kernighan and Denis Ritchie, known as the White Book, and that served as standard until the publication of formal ANSI standard (ANSI X3J11 committee) in 1989.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

In 1980, Bjarne Stroustrup, from Bell labs, began the development of the C++ language. In October 1985, the first commercial release of the language appeared as well as the first edition of the book "The C++ Programming Language" by Bjarne Stroustrup.

During the 1980's the C++ language was being refined until it became a language with its own personality. All that with very few changes of code of original C language, In fact, the ANSI standard for the C language published in 1989 took good part of the contributions of C++ to structured programming.

From 1990 on, ANSI committee X3J16 began the development of a specific standard for C++. In the period elapsed until the publication of the standard in 1998, C++ lived a great expansion in its use and today is the preferred language to develop professional applications on all platforms.

Differences B/w C and C++:

1. C does not have any classes or objects. It is procedure and function driven. There is no concept of access through objects and structures are the only place where there is a access through a compacted variable. c++ is object oriented.

2. C structures have a diferent behaviour compared to c++ structures. Structures in c do not accept functions as their parts.

3. C input/output is based on library and the prcesses are carried out by including functions. C++ i/o is made through console commands cin and cout.

4. C functions do not support overloading. Operator overloading is a process in which the same function has two or more different behaviours based on the data input by the user.

5. C does not support new or delete commands. The memory operations to free or alllocate memory in c are carried out by malloc() and free().

6. Undeclared functions in c++ are not allowed. The function has to have a prototype defined before the main() before use in c++ although in c the functions can be declared at the point of use.

7. After declaring structures and enumerators in c we cannot declare the variable for the structure right after the end of the structure as in c++.

8. For an int main() in c++ we may not write a return statement but the return is mandatory in c if we are using int main().

9. In C++ identifiers are not allowed to contain two or more consecutive underscores in any position. C identifiers cannot start with two or more consecutive underscores, but may contain them in other positions.

10. C has a top down approch whereas c++ has a bottom up approach.

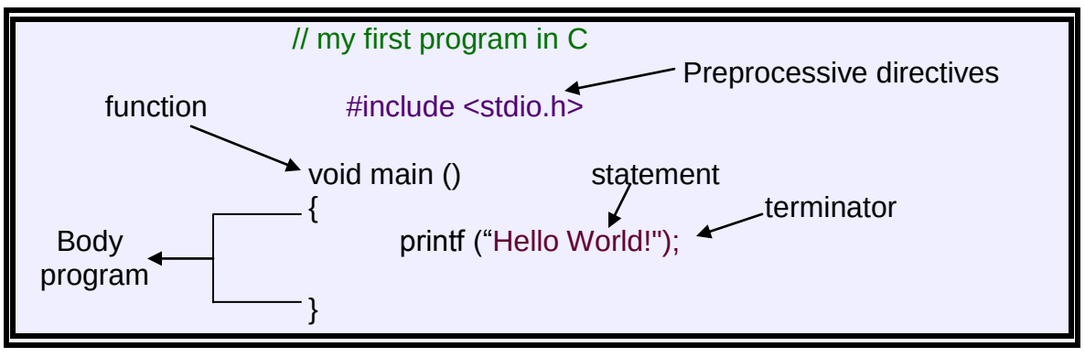
11. in c a character constant is automatically elevated to and integer whereas in c++ this is not the case.

12. In c declaring the global variable several times is allowed but this is not allowed in c++.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Structure of C Program

The basic fundamental components that every C program have is known as structure of program.



// my first program in C

This is a comment line. All lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program. The programmer can use them to include short explanations or observations within the source code itself. In this case, the line is a brief description of what our program is. Comments are parts of the source code. They simply do nothing to compiler . Their purpose is only to allow the programmer to insert notes or descriptions embedded within the source code.

C supports two ways to insert comments:

```
// line comment
/* block comment */
```

The first of them, known as line comment, discards everything from where the pair of slash signs (//) is found up to the end of that same line. The second one, known as block comment, discards everything between the /* characters and the first appearance of the */ characters, with the possibility of including more than one line.

#include <stdio.h>

Lines beginning with a pound sign (#) are directives for the preprocessor. They are not regular code lines with expressions but indications for the compiler's preprocessor. In this case the directive #include <stdio.h>(standered input output) tells the preprocessor to include the stdio.h standard file. This specific file (stdio.h) includes the declarations of the basic standard input-output library in C , and it is included because its functionality is going to be used later in the program.

void main ()

This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C programs start their execution. It does not matter whether there are other functions with other names defined before or after it - the instructions contained within this function's definition will always be the first ones to be executed in any C program. The word main is followed in the code by a pair of parentheses ( ). That is because it is a function declaration, In C, what differentiates a function declaration from other types of expressions are these parentheses that follow its name. Optionally, these parentheses may enclose a list of parameters within them.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Program body { }

Right after these parentheses we can find the body of the main function enclosed in braces ({}). What is contained within these braces is what the function does when it is executed.

Printf("Hello World)";

This line is a C statement. A statement is a simple or compound expression that can actually produce some effect. Printf() represents the standard output character in C, and the meaning of the entire statement is to insert a sequence of characters (in this case the Hello World sequence of characters) into the standard output (which usually is the screen).Printf is declared in the stdio.h standard file, so that's why we needed to include that specific file and to declare that we were going to use this specific namespace earlier in our code.Notice that the statement ends with a semicolon character (;). This character is used to mark the end of the statement and in fact it must be included at the end of all expression statements in all C programs.

Structure of a C++ program

```
1  // my first program in C++
2
3  #include <iostream>
4  using namespace std;
5
6  int main ()
7  {
8    cout << "Hello World!";
9    return 0;
10 }
```

```
Hello World!
```

The first panel shows the source code for our first program. The second one shows the result of the program once compiled and executed. To the left, the grey numbers represent the line numbers - these are not part of the program, and are shown here merely for informational purposes.

The way to edit and compile a program depends on the compiler you are using. Depending on whether it has a Development Interface or not and on its version. Consult the compilers section and the manual or help included with your compiler if you have doubts on how to compile a C++ console program.

// my first program in C++

This is a comment line. All lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program. The programmer can use them to include short explanations or observations within the source code itself. In this case, the line is a brief description of what our program is.

#include <iostream>

Lines beginning with a hash sign (#) are directives for the preprocessor. They are not regular code lines with expressions but indications for the compiler's preprocessor. In this case the directive #include <iostream> tells the preprocessor to include the iostream standard file. This specific file (iostream) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later in the program.

using namespace std;

All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name *std*. So in order to access its functionality we declare with this expression that we will be using these entities. This line is very

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

frequent in C++ programs that use the standard library, and in fact it will be included in most of the source codes included in these tutorials.

int main ()

This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C++ programs start their execution, independently of its location within the source code. It does not matter whether there are other functions with other names defined before or after it - the instructions contained within this function's definition will always be the first ones to be executed in any C++ program. For that same reason, it is essential that all C++ programs have a *main* function.

The word main is followed in the code by a pair of parentheses (()). That is because it is a function declaration: In C++, what differentiates a function declaration from other types of expressions are these parentheses that follow its name. Optionally, these parentheses may enclose a list of parameters within them.

Right after these parentheses we can find the body of the main function enclosed in braces ({}). What is contained within these braces is what the function does when it is executed.

cout << "Hello World!";

This line is a C++ statement. A statement is a simple or compound expression that can actually produce some effect. In fact, this statement performs the only action that generates a visible effect in our first program.

cout is the name of the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters (in this case the Hello World sequence of characters) into the standard output stream (*cout*, which usually corresponds to the screen).

cout is declared in the *iostream* standard file within the *std* namespace, so that's why we needed to include that specific file and to declare that we were going to use this specific namespace earlier in our code.

Notice that the statement ends with a semicolon character (;). This character is used to mark the end of the statement and in fact it must be included at the end of all expression statements in all C++ programs (one of the most common syntax errors is indeed to forget to include some semicolon after a statement).

return 0;

The return statement causes the main function to finish. return may be followed by a return code (in our example is followed by the return code with a value of zero). A return code of *0* for the *main* function is generally interpreted as the program worked as expected without any errors during its execution. This is the most usual way to end a C++ console program. The program has been structured in different lines in order to be more readable, but in C++, we do not have strict rules on how to separate instructions in different lines. For example, instead of

```
1  int main ()
2  {
3    cout << " Hello World!";
4    return 0;
5  }
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

We could have written:

```
int main () { cout << "Hello World!"; return 0; }
```

All in just one line and this would have had exactly the same meaning as the previous code.

In C++, the separation between statements is specified with an ending semicolon (;) at the end of each one, so the separation in different code lines does not matter at all for this purpose. We can write many statements per line or write a single statement that takes many code lines. The division of code in different lines serves only to make it more legible and schematic for the humans that may read it. Let us add an additional instruction to our first program:

```
1  // my second program in C++
2
3
4  #include <iostream>
5
6
7
8  using namespace std;
9
10
11 int main ()
12 {
     cout << "Hello World! ";
     cout << "I'm a C++ program";
     return 0;
   }
```

```
Hello World! I'm a C++ program
```

In this case, we performed two insertions into cout in two different statements. Once again, the separation in different lines of code has been done just to give greater readability to the program, since *main* could have been perfectly valid defined this way:

```
int main () { cout << " Hello World! "; cout << " I'm a C++ program "; return 0; }
```

We were also free to divide the code into more lines if we considered it more convenient:

```
1 int main ()
2
3
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

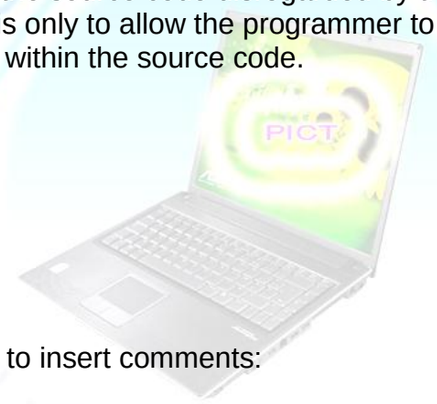
```
{
  cout <<
    "Hello World!";
  cout
    << "I'm a C++ program";
  return 0;
}
```

And the result would again have been exactly the same as in the previous examples.

Preprocessor directives (those that begin by #) are out of this general rule since they are not statements. They are lines read and processed by the preprocessor and do not produce any code by themselves. Preprocessor directives must be specified in their own line and do not have to end with a semicolon (;).

Comments

Comments are parts of the source code disregarded by the compiler. They simply do nothing. Their purpose is only to allow the programmer to insert notes or descriptions embedded within the source code.



C++ supports two ways to insert comments:

```
// line comment
/* block comment */
```

The first of them, known as line comment, discards everything from where the pair of slash signs (//) is found up to the end of that same line. The second one, known as block comment, discards everything between the /* characters and the first appearance of the */ characters, with the possibility of including more than one line. We are going to add comments to our second program:

```
/* my second program in C++
   with more comments */

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

```
Hello World! I'm a C++ program
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

```
12 int main ()
   {
     cout << "Hello World! ";         // prints Hello World!
     cout << "I'm a C++ program"; // prints I'm a C++ program
     return 0;
   }
```

If you include comments within the source code of your programs without using the comment characters combinations //, /* or */, the compiler will take them as if they were C++ expressions, most likely causing one or several error messages when you compile it.

Characteristics of C

Programs

Nowadays computers are able to perform many different tasks, from simple mathematical operations to sophisticated animated simulations. But the computer does not create these tasks by itself, these are performed following a series of predefined instructions call a program which is contains different expression, variable, constants and operator.

Language

A programming language is a set of instructions and a series of conventions specifically designed to order computers what to do.

C has certain characteristics over other programming languages. The most remarkable ones are:

Object-oriented programming

The possibility to orientate programming to objects allows the programmer to design applications from a point of view more like a communication between objects rather than on a structured sequence of code. In addition it allows a greater reusability of code in a more logical and productive way.

Portability

You can practically compile the same C code in almost any type of computer and operating system without making any changes. C is the most used and ported programming languages in the world.

Brevity

Code written in C is very short in comparison with other languages, since the use of special characters is preferred to key words, saving some effort to the programmer (and prolonging the life of our keyboards!).

Modular programming

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

An application's body in C can be made up of several source code files that are compiled separately and then linked together. Saving time since it is not necessary to recompile the complete application when making a single change but only the file that contains it. In addition, this characteristic allows to link C code with code produced in other languages, such as Assembler or C.

C Compatibility

C is backwards compatible with the C language. Any code written in C can easily be included in a C program without making any change.

Speed

The resulting code from a C compilation is very efficient, due indeed to its duality as high-level and low-level language and to the reduced size of the language itself.

Character set of C

The basic building blocks in C program are following character.

Numeric digits

Thses are numerical digits from 0 to 9.

Alphabets

They are of lower case a to z and upper case A to Z

Special characters

These are also known as punctuation symbols the are 32 in number with a blank space as 33rd character they are:

~ ! @ # $ ` % ^ & * ( ) _ + | \ = [ ] –‘ ; : “ ,{ } . / < > ?

and a blank space.

Constant and Variables in C

Variable:

The quantity which changes its value during the execution of program is known as variable.

Rules of variables name

1. The first character must be a letter.
2. No reserved word or predefined is allowed in variable name.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

3. The underscore "_" counts as a letter.
4. Don't begin variable names with underscore names.
5. Upper and lower case letters are different, so x and X are two different names
6. Use lower case for variable names, and all upper case for symbolic constants.

Types of variables

Integer variables

The data type which is used for integers without decimal point is know as integers data types. The integer types come in different sizes, with varying amounts of memory usage and range of represent able numbers. Modifiers are used to designate the size: short, long and long. The standard header file limits.h defines the minimum and maximum values of the integral primitive data types, amongst other limits.

Symbol used

The int is used symbol for integer variable

Types of integer variable

There are three types of decimal integers are used in C language each take different size in computer's memory location

The following table provides a list of the integral types and their *typical* storage sizes and acceptable ranges of values, which may vary from one compiler and platform to another. For integer types of *guaranteed* sizes, ranging from 8 to 64 bits,.

Typical limits of integer types					
Implicit Specifier(s)	**Explicit Specifier**	**Bits**	**Bytes**	**Minimum Value**	**Maximum Value**
short	signed short int	16	2	−32,767	32,767
unsigned short	unsigned short int	16	2	0	65,535
int	signed long int	32	4	−2,147,483,647	2,147,483,647
int	unsigned long int	32	4	0	4,294,967,295
long	signed long int	64	8	−9,223,372,036,854,775,807	9,223,372,036,854,775,807
unsigned long	unsigned long int	64	8	0	18,446,744,073,709,551,615

Rules of integer variables

1. Decimal point is not allowed.
2. The unsigned integer is considered positive.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

3. The negative integer must have "-"sign before variable
4. Special characters are not allowed.

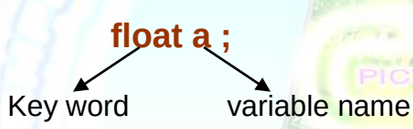
e.g. 0, 09, 99,123, 34567, 10034567 are all integer values

Floating point variable

The floating-point form is used to represent numbers with a fractional component. There are three types of real values, denoted by their symbols single-precision (float), double-precision (double) and double-extended-precision (long double). Each of these may represent values in a different form. Floating-point varibles may be written in decimal notation, e.g. 1.23. Scientific notation may be used by adding e or E followed by a decimal exponent, e.g. 1.23e2 (which has the value 123). Either a decimal point or an exponent is required (otherwise, the number is an integer). Hexadecimal floating-point , which follow similar rules except that they must be prefixed by 0x and use p to specify a hexadecimal exponent. Both decimal and hexadecimal floating-point may be suffixed by f or F to indicate a constant of type float, by l or L to indicate type long double, or left un suffixed for a double The standard header file float.h defines the minimum and maximum values of the floating-point types float, double, and long double. It also defines other limits that are relevant to the processing of floating-point numbers.

Symbol used

The float is used symbol for floating point variable s



Types of floating variable

There are three types of floating point variable which are used in C language each take different size in computer's memory location

Typical limits of floating types					
Implicit Specifier(s)	**symbol**	**Bits**	**Bytes**	**Range**	**Precession level**
float	float	32	4	10^{-38} to 10^{38}	7 digits
Double float	double float	64	8	10^{-308} to 10^{308}	15digits
Long double	long double	80	10	10^{-4932} to 10^{4932}	19 digits

Rules of integer variables

1. Decimal point is allowed.
2. The unsigned floating point variable is consider positive.
3. The negative floating point variable must have "-"sign before variable
4. Special characters are not allowed.

e.g. 0.0, 0.9, 9.9,12.3, 3.4567, 1003.4567 are all floating values

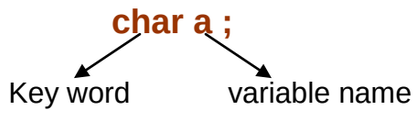
Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Character variables

A single byte, capable of holding one character in the local character set is known as character char is a type distinct from both signed char and unsigned char. It may be a signed type or an unsigned type, depending on the compiler and the character set (C guarantees that members of the C basic character set have positive values).

Symbol used

The char is used as symbol for character variable



Typical limits of character types					
Implicit Specifier(s)	**symbol**	**Bits**	**Bytes**	**Minimum Value**	**Maximum Value**
char	char	8	1	−127 or 0	127 or 255

Boolean variables

Boolean take one value out of two values either it is true or false

Typical limits of Boolean types					
Implicit Specifier(s)	**symbol**	**Bits**	**Bytes**	**Minimum Value**	**Maximum Value**
bool	bool	8	1	False(0)	True (1)

Initialization of variables

When declaring a regular local variable, its value is by default undetermined. But you may want a variable to store a concrete value at the same moment that it is declared. In order to do that, you can initialize the variable. There are two ways to do this in C:

The first one, known as c-like, is done by appending an equal sign followed by the value to which the variable will be initialized:

type identifier = initial_value ;

For example, if we want to declare an int variable called a initialized with a value of 0 at the moment in which it is declared, we could write:

```
int a = 0;
```

The other way to initialize variables, known as constructor initialization, is done by enclosing the initial value between parentheses (()):

type identifier (initial_value) ;

For example:

```
int a (0);
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Both ways of initializing variables are valid and equivalent in C.

```
// initialization of variables

#include <stdio.h>
void main ()
{
  int a=5;               // initial value = 5
  int b(2);              // initial value = 2
  int result;            // initial value
undetermined

  a = a + 3;
  result = a - b;
  printf (“%d”,result) ;

}
```

```
6
```

Constants

Constants are expressions with a fixed value. Constants can be divided in Integer constants, Floating-Point constants, Characters & Strings constants.

Integer constant

Integer constants identify integer decimal values. Notice that to express a numerical constant we do not have to write quotes (") nor any special character. There is no doubt that it is a constant whenever we write 1776 in a program, we will be referring to the value 1776.

```
int a = 1776 ;
```

constant

In addition to decimal numbers (those that all of us are used to use every day) C allows the use of constants as octal numbers (base 8) and hexadecimal numbers (base 16). If we want to express an octal number we have to precede it with a 0 (zero character). And in order to express a hexadecimal number we have to precede it with the characters 0x (zero, x). For example, the following constants are all equivalent to each other:

```
75         // decimal
0113       // octal
0x4b       // hexadecimal
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

All of these represent the same number: 75 (seventy-five) expressed as a base-10 numeral, octal numeral and hexadecimal numeral, respectively.

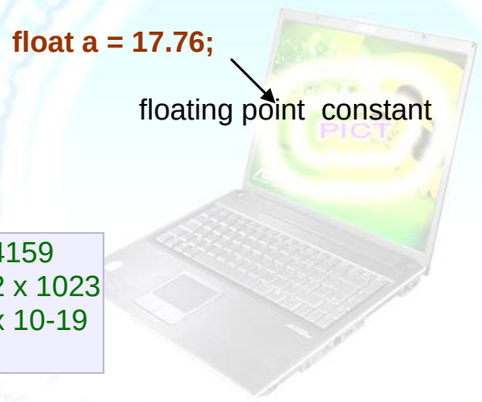
Constants, like variables, are considered to have a specific data type. By default, integer literals are of type int.

```
75         // int
75u        // unsigned int
75l        // long
75ul       // unsigned long
```

In both cases, the suffix can be specified using either upper or lowercase letters.

Floating Point constant

They express numbers with decimals and/or exponents. They can include either a decimal point, an e character (that expresses "by ten at the Xth height", where X is an integer value that follows the e character), or both a decimal point and an e character:




```
3.14159    // 3.14159
6.02e23    // 6.02 x 1023
1.6e-19    // 1.6 x 10-19
3.0        // 3.0
```

These are four valid numbers with decimals expressed in C. The first number is PI, the second one is the number of Avogadro, the third is the electric charge of an electron (an extremely small number) and the last one is the number three expressed as a floating-point numeric literal.

The default type for floating point literals is double. If you explicitly want to express a float or long double numerical literal, you can use the f or l suffixes respectively:

```
3.14159L   // long double
6.02e23f   // float
```

Any of the letters than can be part of a floating-point numerical constant (e, f, l) can be written using either lower or uppercase letters without any difference in their meanings.

Character and string constants

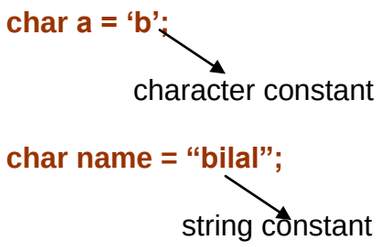
There also exist non-numerical constants, like:

```
'z'
'p'
"Hello world"
"How do you do?"
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

The first two expressions represent single character constants, and the following two represent string literals composed of several characters. Notice that to represent a single character we enclose it between single quotes (') and to express a string (which generally consists of more than one character) we enclose it between double quotes (").

When writing both single character and string literals, it is necessary to put the quotation marks surrounding them to distinguish them from possible variable identifiers or reserved keywords



Reserved word keywords

C language specific the following words as standard reserved keywords they are predefined in compiler they are:

asm, auto, bool, break, case, catch, char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while

Escape Sequences

These are special characters that are used in the source code for formatting the output of C program, like new line (\n) or tab (\t). All of them are preceded by a backslash (\). Here you have a list of some of such escape codes:

\n	newline
\r	carriage return
\t	Tab
\v	vertical tab
\b	backspace
\f	form feed (page feed)
\a	alert (beep)
\'	single quote (')
\"	Double quote (")
\?	question mark (?)
\\	backslash (\)

For example:

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

```
'\n'
'\t'
"Left \t Right"
"one\ntwo\nthree"
```

Additionally, you can express any character by its numerical ASCII code by writing a backslash character (\) followed by the ASCII code expressed as an octal (base-8) or hexadecimal (base-16) number. In the first case (octal) the digits must immediately follow the backslash (for example \23 or \40), in the second case (hexadecimal), an x character must be written before the digits themselves (for example \x20 or \x4A).

String literals can extend to more than a single line of code by putting a backslash sign (\) at the end of each unfinished line.

```
"string expressed in \
two lines"
```

You can also concatenate several string constants separating them by one or several blank spaces, tabulators, newline or any other valid blank character:

Variables and Data Types in C++.

we can define a variable as a portion of memory to store a determined value.

Each variable needs an identifier that distinguishes it from the others. For example, in the previous code the variable identifiers were a, b and result, but we could have called the variables any names we wanted to invent, as long as they were valid identifiers.

Identifiers

A *valid identifier* is a sequence of one or more letters, digits or underscores characters (_). Neither spaces nor punctuation marks or symbols can be part of an identifier. Only letters, digits and single underscore characters are valid. In addition, variable identifiers always have to begin with a letter. They can also begin with an underline character (_), but in some cases these may be reserved for compiler specific keywords or external identifiers, as well as identifiers containing two successive underscore characters anywhere. In no case can they begin with a digit.

Another rule that you have to consider when inventing your own identifiers is that they cannot match any keyword of the C++ language nor your compiler's specific ones, which are *reserved keywords*. The standard reserved keywords are:

asm, auto, bool, break, case, catch, char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while
Additionally, alternative representations for some operators cannot be used as identifiers since they are reserved words under some circumstances:
and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor, xor_eq
Your compiler may also include some additional specific reserved keywords.
Very important: The C++ language is a *"case sensitive"* language. That means that

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

an identifier written in capital letters is not equivalent to another one with the same name but written in small letters. Thus, for example, the RESULT variable is not the same as the result variable or the Result variable. These are three different variable identifiers.

Fundamental data types

summary of the basic fundamental data types in C++, as well as the range of values that can be represented with each one:

Name	Description	Size*	Range*
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	Wide character.	2 or 4 bytes	1 wide character

Declaration of variables

In order to use a variable in C++, we must first declare it specifying which data type we want it to be. The syntax to declare a new variable is to write the specifier of the desired data type (like int, bool, float...) followed by a valid variable identifier. For example:

```
1 int a;
2 float mynumber;
```

These are two valid declarations of variables. The first one declares a variable of type *int* with the identifier *a*. The second one declares a variable of type *float* with the identifier *mynumber*. Once declared, the variables *a* and *mynumber* can be used within the rest of their scope in the program.

If you are going to declare more than one variable of the same type, you can declare all of them in a single statement by separating their identifiers with commas. For example:

```
1 int a, b, c;
```

This declares three variables (*a*, *b* and *c*), all of them of type *int*, and has exactly the same meaning as:

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

```
1  int a;
2  int b;
3  int c;
```

The integer data types *char*, *short*, *long* and *int* can be either signed or unsigned depending on the range of numbers needed to be represented. Signed types can represent both positive and negative values, whereas unsigned types can only represent positive values (and zero). This can be specified by using either the specifier *signed* or the specifier *unsigned* before the type name. For example:

```
1 unsigned short int NumberOfSisters;
2 signed int MyAccountBalance;
```

By default, if we do not specify either *signed* or *unsigned* most compiler settings will assume the type to be signed, therefore instead of the second declaration above we could have written:

```
int MyAccountBalance;
```

with exactly the same meaning (with or without the keyword signed)

An exception to this general rule is the *char* type, which exists by itself and is considered a different fundamental data type from *signed char* and *unsigned char*, thought to store characters. You should use either signed or unsigned if you intend to store numerical values in a *char*-sized variable.

```
1  // operating with variables
2
3  #include <iostream>
4  using namespace std;
5
6  int main ()
7  {
8    // declaring variables:
9    int a, b;
10   int result;
11
12   // process:
13   a = 5;
14   b = 2;
15   a = a + 1;
16   result = a - b;
17
18   // print out the result:
19   cout << result;
20
21   // terminate the program:
22   return 0;
23 }
```

```
4
```

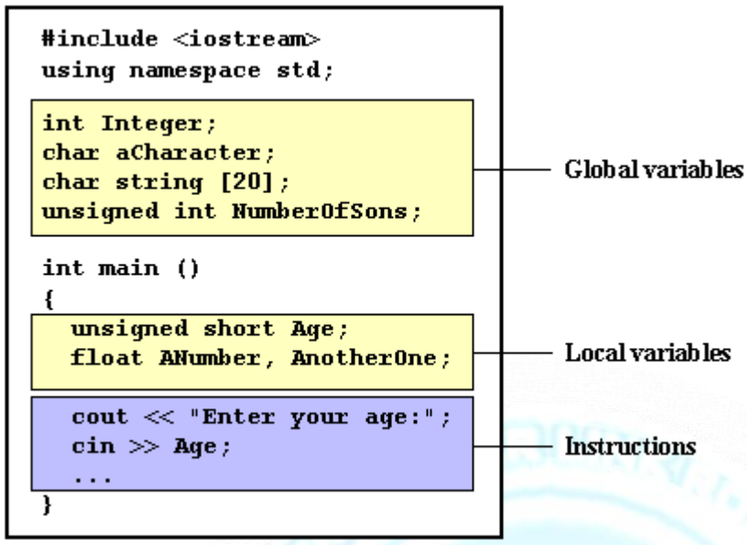
Scope of variables

All the variables that we intend to use in a program must have been declared with its type specifier in an earlier point in the code, like we did in the previous code at the

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

beginning of the body of the function main when we declared that *a*, *b*, and *result* were of type *int*.

A variable can be either of global or local scope. A global variable is a variable declared in the main body of the source code, outside all functions, while a local variable is one declared within the body of a function or a block.



Global variables can be referred from anywhere in the code, even inside functions, whenever it is after its declaration.
The scope of local variables is limited to the block enclosed in braces ({}) where they are declared. For example, if they are declared at the beginning of the body of a function (like in function *main*) their scope is between its declaration point and the end of that function. In the example above, this means that if another function existed in addition to *main*, the local variables declared in *main* could not be accessed from the other function and vice versa.

```
1   // initialization of variables
2
3   #include <iostream>
4   using namespace std;
5
6   int main ()
7   {
8     int a=5;               // initial value = 5
9     int b(2);              // initial value = 2
10    int result;            // initial value
11  undetermined
12
13    a = a + 3;
14    result = a - b;
15    cout << result;
16
17    return 0;
    }
```

```
6
```

Introduction to strings

Variables that can store non-numerical values that are longer than one single character are known as strings.

The C++ language library provides support for strings through the standard string class. This is not a fundamental type, but it behaves in a similar way as fundamental types do in its most basic usage.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

A first difference with fundamental data types is that in order to declare and use objects (variables) of this type we need to include an additional header file in our source code: <string> and have access to the std namespace (which we already had in all our previous programs thanks to the using namespace statement).

```
1   // my first string
2   #include <iostream>
3   #include <string>
4   using namespace std;
5
6   int main ()
7   {
8     string mystring = "This is a string";
9     cout << mystring;
10    return 0;
11  }
```

```
This is a string
```


Input and output Functions of C and C++:

Output - printf

printf's name comes from **print f**ormatted. It generates output which consists of characters to be printed and also special character sequences which request that other arguments be fetched, formatted, and inserted into the string. Our very first program was nothing more than a call to printf, printing a constant string:

```
printf("Hello, world!\n");
```

Our second program also featured a call to printf:

C uses a convenient abstraction called *streams* to perform input and output operations in sequential media such as the screen or the keyboard. A stream is an object where a program can either insert or extract characters to/from it. The standard C library includes the header file iostream, where the standard input and output stream objects are declared.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Input – Scanf

Scanf() is use reads characters from out side the world through key board. According to the specification in format, and stores the results. Scanf() ignores blanks and tabs in its format string. Furthermore, it skips over white space (blanks, tabs,
new lines, etc.) as it looks for input values.

```
#include <stdio.h>
void main ()
{
        int a;
        printf("input number : “  );
        scanf(“%d”,& a);
        printf (“The number u enter is %d”);
}
```

The out put of program will be
Input number :10
The number is 10
Where & is the address operator used to store the value of variable at specific location in the above case it will be a.

Standard Output (cout)

By default, the standard output of a program is the screen, and the C stream object defined to access it is cout.

cout is used in conjunction with the *insertion operator*, which is written as << (two "less than" signs).

```
cout << "Output sentence"; // prints Output sentence on screen
cout << 120;               // prints number 120 on screen
cout << x;                 // prints the content of x on screen
```

The << operator inserts the data that follows it into the stream preceding it. In the examples above it inserted the constant string Output sentence, the numerical constant 120 and variable x into the standard output stream cout. Notice that the sentence in the first instruction is enclosed between double quotes (") because it is a constant string of characters. Whenever we want to use constant strings of characters we must enclose them between double quotes (") so that they can be clearly distinguished from variable names. For example, these two sentences have very different results:

```
cout << "Hello";  // prints Hello
cout << Hello;    // prints the content of Hello variable
```

The insertion operator (<<) may be used more than once in a single statement:

```
cout << "Hello, " << "I am " << "a C++ statement";
```

This last statement would print the message Hello, I am a C++ statement on the screen. The utility of repeating the insertion operator (<<) is demonstrated when we want to print out a combination of variables and constants or more than one variable:

```
cout << "Hello, I am " << age << " years old and my zipcode is " << zipcode;
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

If we assume the age variable to contain the value 24 and the zipcode variable to contain 90064 the output of the previous statement would be:

```
Hello, I am 24 years old and my zipcode is 90064
```

It is important to notice that cout does not add a line break after its output unless we explicitly indicate it, therefore, the following statements:

```
cout << "This is a sentence.";
cout << "This is another sentence.";
```

will be shown on the screen one following the other without any line break between them:

This is a sentence.This is another sentence.

even though we had written them in two different insertions into cout. In order to perform a line break on the output we must explicitly insert a new-line character into cout. In C++ a new-line character can be specified as \n (backslash, n):

```
cout << "First sentence.\n ";
cout << "Second sentence.\nThird sentence.";
```

This produces the following output:

First sentence.
Second sentence.
Third sentence.

Additionally, to add a new-line, you may also use the endl manipulator. For example:

```
cout << "First sentence." << endl;
cout << "Second sentence." << endl;
```

would print out:

First sentence.
Second sentence.

The endl manipulator produces a newline character, exactly as the insertion of '\n' does, but it also has an additional behavior when it is used with buffered streams: the buffer is flushed. Anyway, cout will be an unbuffered stream in most cases, so you can generally use both the \n escape character and the endl manipulator in order to specify a new line without any difference in its behavior.

Standard Input (cin)

The standard input device is usually the keyboard. Handling the standard input in C++ is done by applying the overloaded operator of extraction (>>) on the cin stream. The operator must be followed by the variable that will store the data that is going to be extracted from the stream. For example:

```
int age;
cin >> age;
```

The first statement declares a variable of type int called age, and the second one waits for an input from cin (the keyboard) in order to store it in this integer variable.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

cin can only process the input from the keyboard once the RETURN key has been pressed. Therefore, even if you request a single character, the extraction from cin will not process the input until the user presses RETURN after the character has been introduced.

You must always consider the type of the variable that you are using as a container with cin extractions. If you request an integer you will get an integer, if you request a character you will get a character and if you request a string of characters you will get a string of characters.

```
// i/o example
#include <stdio.h>
void main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
}
```

```
Please enter an integer value: 702
The value you entered is 702 and its double is 1404.
```

The user of a program may be one of the factors that generate errors even in the simplest programs that use cin (like the one we have just seen). Since if you request an integer value and the user introduces a name (which generally is a string of characters), the result may cause your program to misoperate since it is not what we were expecting from the user. So when you use the data input provided by cin extractions you will have to trust that the user of your program will be cooperative and that he/she will not introduce his/her name or something similar when an integer value is requested. A little ahead, when we see the stringstream class we will see a possible solution for the errors that can be caused by this type of user input.

You can also use cin to request more than one datum input from the user:

```
cin >> a >> b;
```

is equivalent to:

```
cin >> a;
cin >> b;
```

In both cases the user must give two data, one for variable a and another one for variable b that may be separated by any valid blank separator: a space, a tab character or a newline.

cin and strings

We can use cin to get strings with the extraction operator (>>) as we do with fundamental data type variables:

```
cin >> mystring;
```

However, as it has been said, cin extraction stops reading as soon as if finds any blank space character, so in this case we will be able to get just one word for each extraction. This behavior may or may not be what we want; for example if we want to get a sentence from the user, this extraction operation would not be useful.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

In order to get entire lines, we can use the function getline, which is the more recommendable way to get user input with cin:

```
// cin with strings
#include <iostream.h>
#include <string.h>
  void main ()
{
  string mystr;
  cout << "What's your name? ";
  getline (cin, mystr);
  cout << "Hello " << mystr << ".\n";
  cout << "What is your favorite team? ";
  getline (cin, mystr);
  cout << "I like " << mystr << " too!\n";
}
```

```
What's your name? Juan Soulié
Hello Juan Soulié.
What is your favorite team? The Isotopes
I like The Isotopes too!
```

Notice how in both calls to getline we used the same string identifier (mystr). What the program does in the second call is simply to replace the previous content by the new one that is introduced.

Format specifiers

Printf() function always prints the string but some time we need any integer or floating point variables to be printed with the strings for that C language uses special characters called format specifiers. It replaced the characters of integer variable simply by writhing its specific format spericfier symbol with strings e.g. %d with the value of the variable i.

```
printf("i is %d\n", i);
```

There are quite a number of format specifiers for printf. Here are the basic ones :

%d	print an int argument in decimal
%ld	print a long int argument in decimal
%c	print a character
%s	print a string
%f	print a float or double argument
%e	same as %f, but use exponential notation
%g	use %e or %f, whichever is better
%o	print an int argument in octal (base 8)
%x	print an int argument in hexadecimal (base 16)
%%	print a single %

It is also possible to specify the width and precision of numbers and strings as they are inserted a notation like %3d means to print an int in a field at least 3 spaces wide; a notation like %5.2f means to print a float or double in a field at least 5 spaces wide, with two places to the right of the decimal.)

To illustrate with a few more examples: the call

```
printf("%c %d %f %e %s %d%%\n", '1', 2, 3.14, 56000000., "eight", 9);
```

would print

```
1 2 3.140000 5.600000e+07 eight 9%
```

The call

```
printf("%d %o %x\n", 100, 100, 100);
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

would print

100 144 64

Successive calls to printf just build up the output a piece at a time, so the calls

```
printf("Hello, ");
printf("world!\n");
```

would also print Hello, world! (on one line of output).

There isn't really much difference between a character and an integer in C; most of the difference is in whether we choose to interpret an integer as an integer or a character. printf is one place where we get to make that choice: %d prints an integer value as a string of digits representing its decimal value, while %c prints the character corresponding to a character set value. So the lines

```
char c = 'A';
int i = 97;
printf("c = %c, i = %d\n", c, i);
```

would print c as the character A and i as the number 97. But if, on the other hand, we called

```
printf("c = %d, i = %c\n", c, i);
```

we'd see the decimal value (printed by %d) of the character 'A', followed by the character (whatever it is) which happens to have the decimal value 97.

Comments

Comments are parts of the source code disregarded by the compiler. They simply do nothing. Their purpose is only to allow the programmer to insert notes or descriptions embedded within the source code.

C supports two ways to insert comments:

```
// line comment
/* block comment */
```

The first of them, known as line comment, discards everything from where the pair of slash signs (//) is found up to the end of that same line. The second one, known as block comment, discards everything between the /* characters and the first appearance of the */ characters, with the possibility of including more than one line. We are going to add comments to our second program:

```
/* my second program in C
with more comments */
#include <stdio.h>
void main ()
{
printf( "Hello World! ");       // prints Hello World!
printf( "I'm a C program"; // prints I'm a C program
}
```

```
Hello World! I'm a C program
```

If you include comments within the source code of your programs without using the comment characters combinations //, /* or */, the compiler will take them as if they were C expressions, most likely causing one or several error messages when you compile it.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Chapter 2.
OPERATORS IN C AND C++

MAIN TOPIC COVERED

- ASSIGMENT OPERATORS
- ARITHMETIC OPERATORS
- INCREMENT / DECREMENT
- RELATIONAL OPERATORS
- LOGICAL OPERATORS
- COMMA OPERATOR
- BITWISE OPERATORS
- SIZE OF OPERATOR
- EXPLICIT TYPE CASTING OPERATOR
- PRECEDENCE LEVEL OF OPEATORS

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Operators

Assignment (=) Operator

The assignment operator assigns a value to a variable.

expression	is equivalent to
+ =	Increase R.H.S of equality
- =	Decrease R.H.S of equality
/ =	Divide R.H.S of equality
* =	Multiply R.H.S of equality
% =	Remainder R.H.S of equality

This statement assigns the integer value 5 to the variable a. The part at the left of the assignment operator (=) is known as the *lvalue* (left value) and the right one as the *rvalue* (right value). The lvalue has to be a variable whereas the rvalue can be either a constant, a variable, the result of an operation or any combination of theseThe most important rule when assigning is the *right-to-left* rule: The assignment operation always takes place from right to left, and never the other way:

```
a = b;
```

This statement assigns to variable a (the lvalue) the value contained in variable b (the rvalue). The value that was stored until this moment in a is not considered at all in this operation, and in fact that value is lost. Consider also that we are only assigning the value of b to a at the moment of the assignment operation. Therefore a later change of b will not affect the new value of a. For example, let us have a look at the following code - I have included the evolution of the content stored in the variables as comments:

```
// assignment operator

#include <stdio.h>
void main ()
{
  int a, b;         // a:?,  b:?
  a = 10;           // a:10, b:?
  b = 4;            // a:10, b:4
  a = b;            // a:4,  b:4
  b = 7;            // a:4,  b:7
  printf( " a:");
  printf( "%d", a);
  printf( " " b:");
  printf( "%d", b);
}
```

```
a:4 b:7
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

This code will give us as result that the value contained in a is 4 and the one contained in b is 7. Notice how a was not affected by the final modification of b, even though we declared a = b earlier (that is because of the *right-to-left rule*).

A property that C has over other programming languages is that the assignment operation can be used as the rvalue (or part of an rvalue) for another assignment operation. For example:

```
a = 2 + (b = 5);
```

is equivalent to:

```
b = 5;
a = 2 + b;
```

that means: first assign 5 to variable b and then assign to a the value 2 plus the result of the previous assignment of b (i.e. 5), leaving a with a final value of 7.

The following expression is also valid in C:

```
a = b = c = 5;
```

It assigns 5 to the all the three variables: a, b and c.

C++ example, let us have a look at the following code - I have included the evolution of the content stored in the variables as comments:

```
// assignment operator

#include <iostream>

using namespace std;


int main ()

{

  int a, b;         // a:?,  b:?

  a = 10;           // a:10, b:?

  b = 4;            // a:10, b:4

  a = b;            // a:4,  b:4

  b = 7;            // a:4,  b:7


  cout << "a:";

  cout << a;
```

```
a:4 b:7
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

```
  cout << " b:";
  cout << b;


  return 0;
}
```

This code will give us as result that the value contained in a is 4 and the one contained in b is 7. Notice how a was not affected by the final modification of b, even though we declared a = b earlier (that is because of the right-to-left rule).

A property that C++ has over other programming languages is that the assignment operation can be used as the rvalue (or part of an rvalue) for another assignment operation. For example:

```
a = 2 + (b = 5);
```

is equivalent to:

```
1 b = 5;
2
  a = 2 + b;
```

that means: first assign 5 to variable b and then assign to a the value 2 plus the result of the previous assignment of b (i.e. 5), leaving a with a final value of 7.

Arithmetic operators (+, -, *, /, %)

The five arithmetical operations supported by the C language are:

+	addition
-	subtraction
*	multiplication

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

/	division
%	modulo

Operations of addition, subtraction, multiplication and division literally correspond with their respective mathematical operators. The only one that you might not be so used to see may be *modulo*; whose operator is the percentage sign (%). Modulo is the operation that gives the remainder of a division of two values. For example, if we write:

Operator	Operation	Result
Addition	a = 12 + 3;	15
Subtraction	a = 12 - 3;	9
Multiplication	a = 12 * 3;	36
Division	a = 12 / 3;	4
Modulus or Remainder	a = 11 % 3;	0

Increment and Decrement (++, --) Operators

Shortening even more some expressions, the increase operator (++) and the decrease operator (--) increase or reduce by one the value stored in a variable. They are equivalent to +=1 and to -=1, respectively. Thus:

```
c++;
c+=1;
c=c+1;
```

are all equivalent in its functionality: the three of them increase by one the value of c.

In the early C compilers, the three previous expressions probably produced different executable code depending on which one was used. Nowadays, this type of code optimization is generally done automatically by the compiler, thus the three expressions should produce exactly the same executable code.

A characteristic of this operator is that it can be used both as a prefix and as a suffix. That means that it can be written either before the variable identifier (++a) or after it (a++). Although in simple expressions like a++ or ++a both have exactly the same meaning, in other expressions in which the result of the increase or decrease operation is evaluated as a value in an outer expression they may have an important difference in their meaning: In the case that the increase operator is used as a prefix (++a) the value is increased before the result of the expression is evaluated and therefore the increased value is considered in the outer expression; in case that it is used as a suffix (a++) the value stored in a is increased after being evaluated and therefore the value stored before the increase operation is evaluated in the outer expression. Notice the difference:

Example 1	Example 2
B=3; A=++B; // A contains 4, B contains 4	B=3; A=B++; // A contains 3, B contains 4

In Example 1, B is increased before its value is copied to A. While in Example 2, the value of B is copied to A and then B is increased.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Relational and equality operators (==, !=, >, <, >=, <=)

In order to evaluate a comparison between two expressions we can use the relational and equality operators. The result of a relational operation is a Boolean value that can only be true or false, according to its Boolean result.

We may want to compare two expressions, for example, to know if they are equal or if one is greater than the other is. Here is a list of the relational and equality operators that can be used in C:

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Here there are some examples:

```
(7 == 5)     // evaluates to false.
(5 > 4)      // evaluates to true.
(3 != 2)     // evaluates to true.
(6 >= 6)     // evaluates to true.
(5 < 5)      // evaluates to false.
```

Of course, instead of using only numeric constants, we can use any valid expression, including variables. Suppose that a=2, b=3 and c=6,

```
(a == 5)     // evaluates to false since a is not equal to 5.
(a*b >= c)   // evaluates to true since (2*3 >= 6) is true.
(b+4 > a*c)  // evaluates to false since (3+4 > 2*6) is false.
((b=2) == a) // evaluates to true.
```

Be careful! The operator = (one equal sign) is not the same as the operator == (two equal signs), the first one is an assignment operator (assigns the value at its right to the variable at its left) and the other one (==) is the equality operator that compares whether both expressions in the two sides of it are equal to each other. Thus, in the last expression ((b=2) == a), we first assigned the value 2 to b and then we compared it to a, that also stores the value 2, so the result of the operation is true.

Logical operators (! , &&, ||)

The Operator ! is the C operator to perform the Boolean operation NOT, it has only one operand, located at its right, and the only thing that it does is to inverse the value of it, producing false if its operand is true and true if its operand is false. Basically, it returns the opposite Boolean value of evaluating its operand. For example:

```
!(5 == 5)    // evaluates to false because the expression at its right (5 == 5) is true.
!(6 <= 4)    // evaluates to true because (6 <= 4) would be false.
!true        // evaluates to false
!false       // evaluates to true.
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

The logical operators && and || are used when evaluating two expressions to obtain a single relational result. The operator && corresponds with Boolean logical operation AND. This operation results true if both its two operands are true, and false otherwise. The following panel shows the result of operator && evaluating the expression a && b:

AND (&&) OPERATOR

a	b	a && b (a.b)
True(1)	True(1)	True(1)
True(1)	False(0)	False(0)
False(0)	True(1)	False(0)
False(0)	False(0)	False(0)

The operator || corresponds with Boolean logical operation OR. This operation results true if either one of its two operands is true, thus being false only when both operands are false themselves. Here are the possible results of a || b:

OR (||) OPERATOR

a	b	a && b (a+b)
True(1)	True(1)	True(1)
True(1)	False(0)	False(0)
False(0)	True(1)	False(0)
False(0)	False(0)	False(0)

For example:

```
( (5 == 5) && (3 > 6) )  // evaluates to false ( true && false ).
( (5 == 5) || (3 > 6) )  // evaluates to true ( true || false ).
```

Conditional operator (?)

The conditional operator evaluates an expression returning a value if that expression is true and a different one if the expression is evaluated as false. Its format is:

condition ? result1 : result2

If condition is true the expression will return result1, if it is not it will return result2.

```
7==5 ? 4 : 3     // returns 3, since 7 is not equal to 5.
7==5+2 ? 4 : 3   // returns 4, since 7 is equal to 5+2.
5>3 ? a : b      // returns the value of a, since 5 is greater than 3.
a>b ? a : b      // returns whichever is greater, a or b.
```

```
// conditional operator
#include <iostream.h>

void main ()
{
  int a,b,c;
```

```
7
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

```
  a=2;
  b=7;
  c = (a>b) ? a : b;

  printf( " c;
}
```

In this example a was 2 and b was 7, so the expression being evaluated (a>b) was not true, thus the first value specified after the question mark was discarded in favor of the second value (the one after the colon) which was b, with a value of 7.

Comma operator (,)

The comma operator (,) is used to separate two or more expressions that are included where only one expression is expected. When the set of expressions has to be evaluated for a value, only the rightmost expression is considered.

For example, the following code:

```
a = (b=3, b+2);
```

Would first assign the value 3 to b, and then assign b+2 to variable a. So, at the end, variable a would contain the value 5 while variable b would contain value 3.

Bitwise Operators (&, |, ^, ~, <<, >>)

Bitwise operators modify variables considering the bit patterns that represent the values they store.

operator	asm equivalent	description
&	AND	Bitwise AND
|	OR	Bitwise Inclusive OR
^	XOR	Bitwise Exclusive OR
~	NOT	Unary complement (bit inversion)
<<	SHL	Shift Left
>>	SHR	Shift Right

Explicit type casting operator

Type casting operators allow you to convert a datum of a given type to another. There are several ways to do this in C. The simplest one, which has been inherited from the C language, is to precede the expression to be converted by the new type enclosed between parentheses (()):

```
int i;
float f = 3.14;
i = (int) f;
```

The previous code converts the float number 3.14 to an integer value (3), the remainder is lost. Here, the typecasting operator was (int). Another way to do the same thing in C is using the functional notation: preceding the expression to be converted by the type and enclosing the expression between parentheses:

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

```
i = int ( f );
```

Both ways of type casting are valid in C.

sizeof( )

This operator accepts one parameter, which can be either a type or a variable itself and returns the size in bytes of that type or object:

```
a = sizeof (char);
```

This will assign the value 1 to a because char is a one-byte long type. The value returned by sizeof is a constant, so it is always determined before program execution.

Precedence of operators

When writing complex expressions with several operands, we may have some doubts about which operand is evaluated first and which later. For example, in this expression:

```
a = 5 + 7 % 2
```

we may doubt if it really means:

```
a = 5 + (7 % 2)    // with a result of 6, or
a = (5 + 7) % 2    // with a result of 0
```

The correct answer is the first of the two expressions, with a result of 6. There is an established order with the priority of each operator, and not only the arithmetic ones (those whose preference come from mathematics) but for all the operators which can appear in C. From greatest to lowest priority, the priority order is as follows:

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Level	Operator	Description	Grouping
1	: :	scope	Left-to-right
2	( ) [ ] . -> ++ -- dynamic_cast static_cast reinterpret_cast const_cast typeid	postfix	Left-to-right
3	++ -- ~ ! sizeof new delete	unary (prefix)	Right-to-left
	* &	Indirection and reference (pointers)	
	+ -	unary sign operator	
4	(type)	type casting	Right-to-left
5	.* ->*	pointer-to-member	Left-to-right
6	* / %	multiplicative	Left-to-right
7	+ -	additive	Left-to-right
8	<< >>	shift	Left-to-right
9	< > <= >=	relational	Left-to-right
10	== !=	equality	Left-to-right
11	&	bitwise AND	Left-to-right
12	^	bitwise XOR	Left-to-right
13	|	bitwise OR	Left-to-right
14	&&	logical AND	Left-to-right
15	||	logical OR	Left-to-right
16	?:	conditional	Right-to-left
17	= *= /= %= += -= >>= <<= &= ^= !=	assignment	Right-to-left
18	,	comma	Left-to-right

Grouping defines the precedence order in which operators are evaluated in the case that there are several operators of the same level in an expression.

All these precedence levels for operators can be manipulated or become more legible by removing possible ambiguities using parentheses signs (and), as in this example:

```
a = 5 + 7 % 2;
```

might be written either as:

```
a = 5 + (7 % 2);
```

or

```
a = (5 + 7) % 2;
```

depending on the operation that we want to perform.

So if you want to write complicated expressions and you are not completely sure of the precedence levels, always include parentheses. It will also become a code easier to read.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

CHAPTER 3.
DECISION STRUCTURES

Main Topics Covered

- THE IF STATEMENT
- THE IF-ELSE STATMENT
- THE ELSE IF STATEMENT
- SWITCH STATEMENT

\

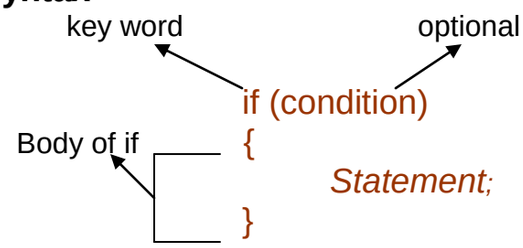
Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

DECISION STRUCTURES

If Statement

The if-else statement is used to express decisions.

Syntax

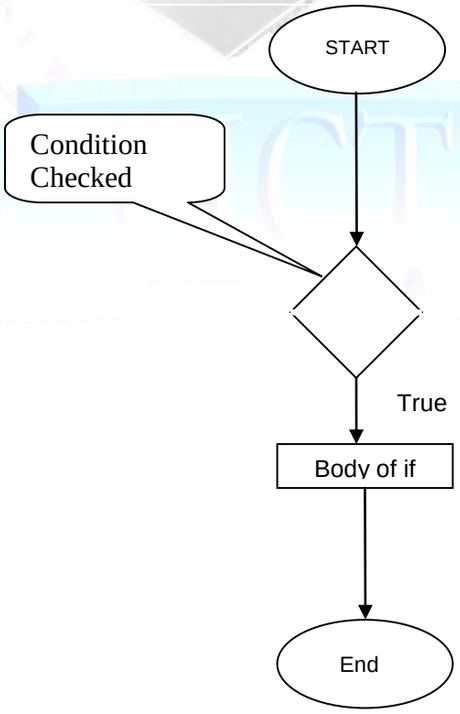


Where

condition conditionis the expression that is being evaluated. If this condition is true, statement is executed. If it is false, statement is ignored (not executed) and the program continues right after this conditional structure. For example, the following code fragment prints x is 100 only if the value stored in the x variable is indeed 100:

```
if (x == 100)
{
   printf ( "x is  %d");
}
```

Flow chart



IF statement Flow chart

In the above flow

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

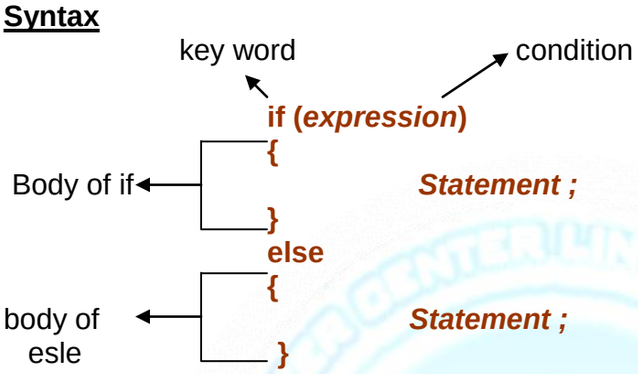
chart following steps takes place

1. Initialization takes place.
2. Condition is checked.
3. If condition is true the body of if is executed.
4. If condition is false then body will not be executed and program will end.
5. The program will end.

If-else statement

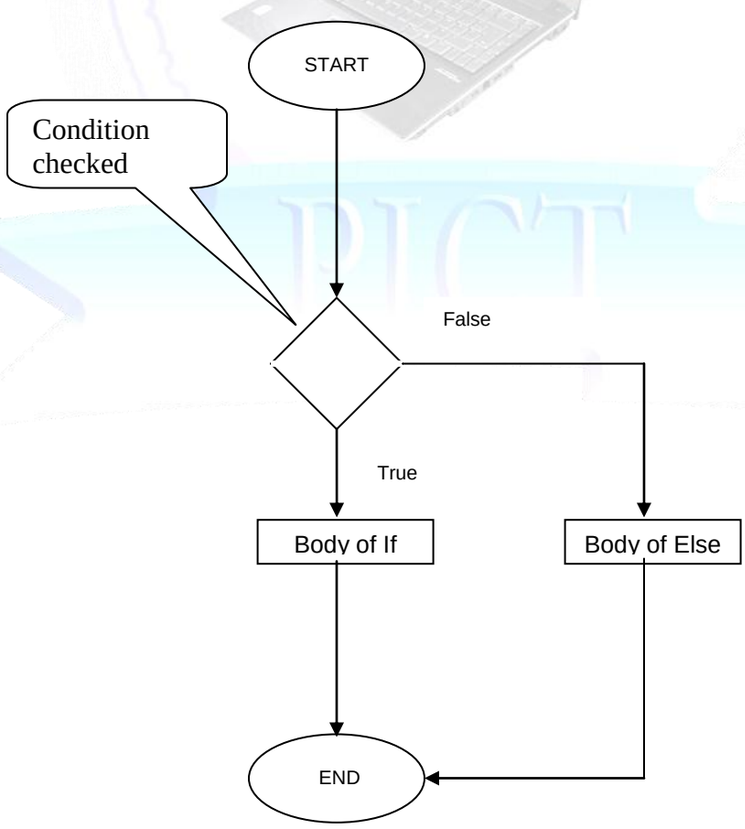
The if-else statement is used when if expression doesn't fulfill the condition.

Syntax



Where condition is the expression that is being evaluated. If this condition is true, statement of **if** is executed if it is false then the statement of **else** is executed.

Flow chart



In the above flow chart following steps takes place

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

1. Initialization takes place.
2. Condition is checked.
3. If condition is true the body of if is executed.
4. If condition is false then body of else is executed.
5. The program will end.

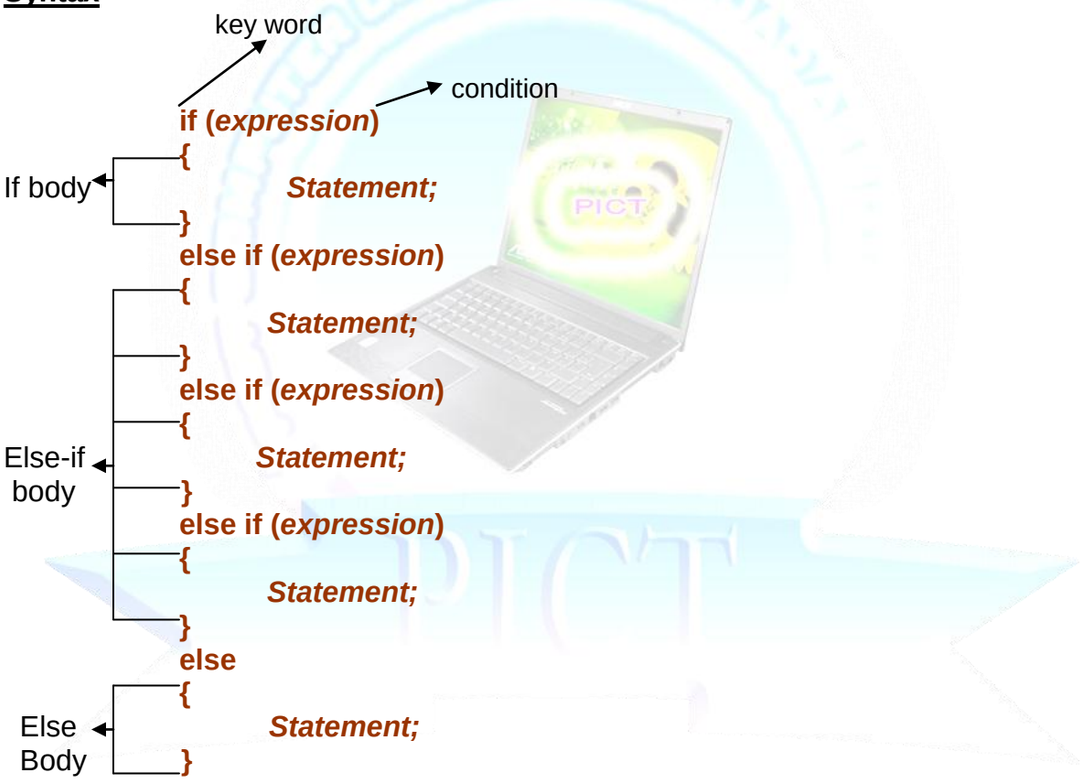
```
if (x == 100)
{
printf ( "x is 100");
}
else
{
printf ( "x is not 100");
}
```

It will prints on the screen x is 100 if the condition is true. If it is false then it will printout x is not 100.

Else-if satatment

The else-if statement is used for multi-way decision making

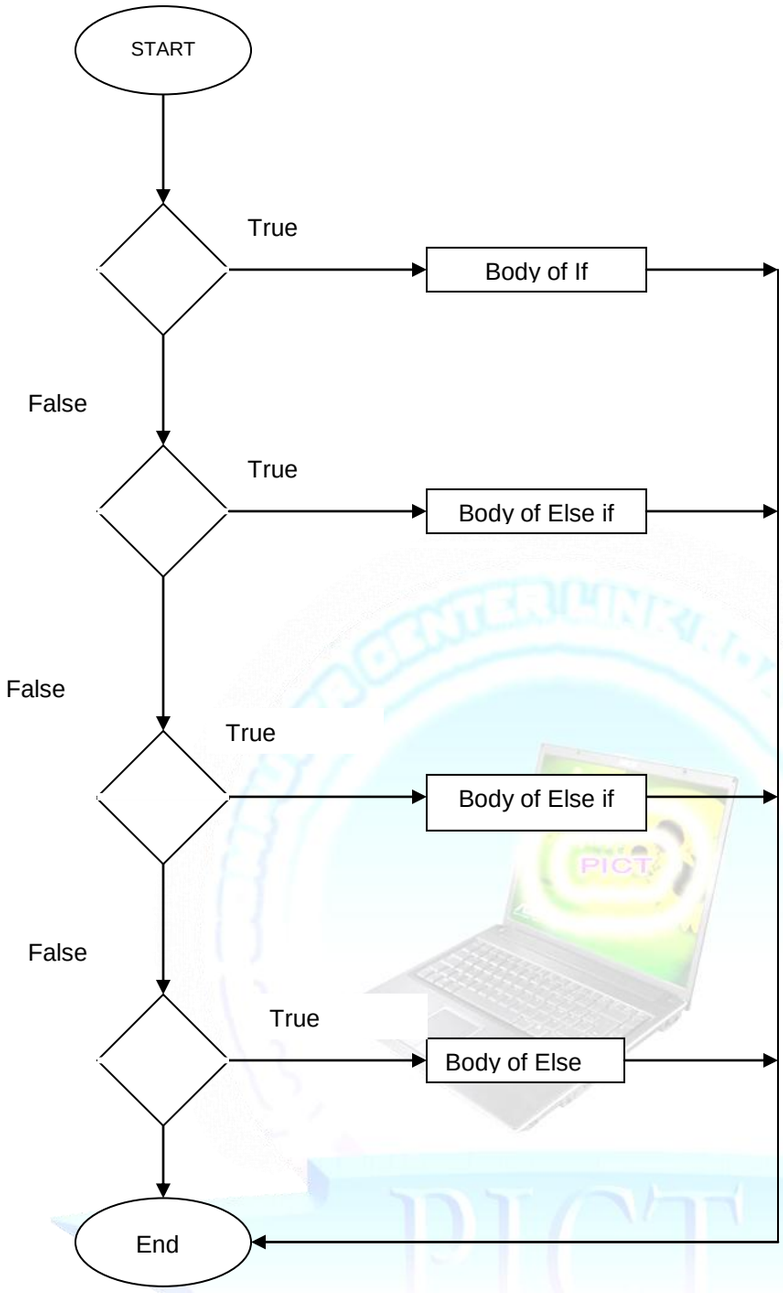
Syntax



This sequence of if statements is the most general way of writing a multi-way decision. The *expressions* are evaluated in order; if an *expression* is true, the *statement* associated with it is executed, and this terminates the whole chain. As always, the code for each *statement* is either a single statement, or a group of them in braces. The last else part handles the ``none of the above'' or default case where none of the other conditions is satisfied.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Flow chart



In the above flow chart following steps takes place

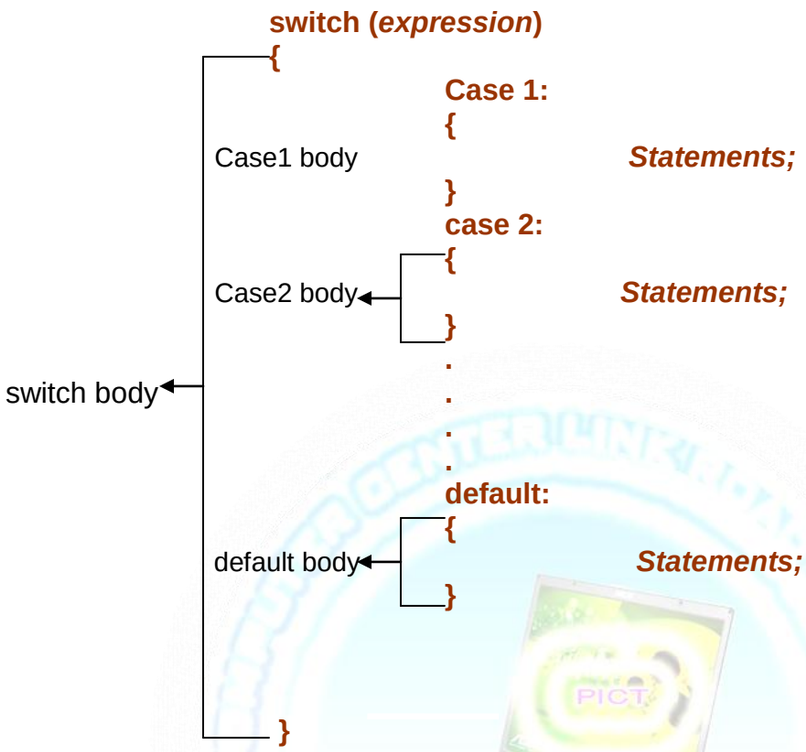
1. Initialization takes place.
2. Condition is checked.
3. If condition is true the body of if is executed.
4. If condition is false then next condition is checked and body of if-else is executed.
5. The condition is checked again and again until the condition come true.
6. If none of else-if condition is true the body of else is executed.
7. The program will end.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Switch Statement

The switch statement is a multi-way decision making technique that tests whether an expression matches one of a alternatives or not.

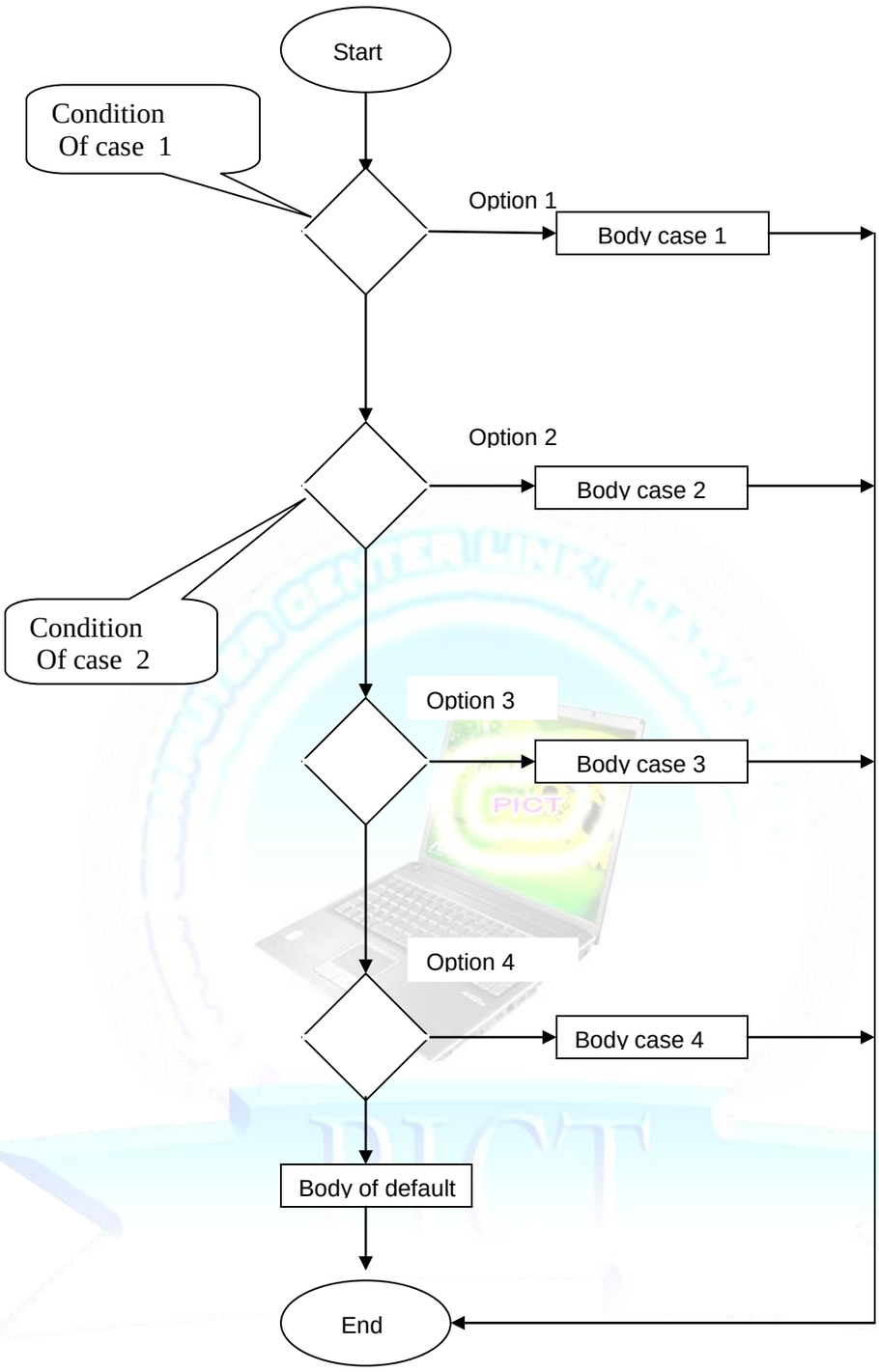
Syntax



Each case is labeled by one or more integer expressions. If a case matches the expression value, execution starts at that case. All case expressions must be different. The case labeled default is executed if none of the other cases are satisfied. A default is optional; if it isn't there and if none of the cases match, no action at all takes place. Cases and the default clause can occur in any order.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Flow chart



1. Initialization takes place.
2. Case checking takes place.
3. If the matched case is found then body of case is executed.
4. If the case is not found the body of default is executed.
5. The program will end.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

CHAPTER4.
LOOPS

Main Topics Covered

- THE FOR LOOP
- THE WHILE LOOP
- THE DO WHILE LOOP

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

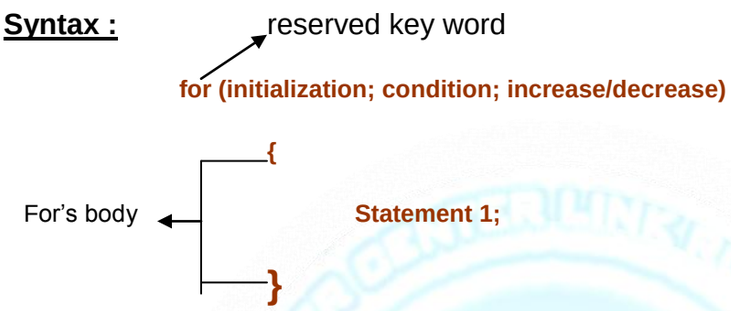
LOOPS

Loops have as purpose to repeat a statement for a certain number of times or while a condition is true . They are also known as repetations and Iteration structures.

For loop

The for loop is used when number of repetitions are known to the programmer.

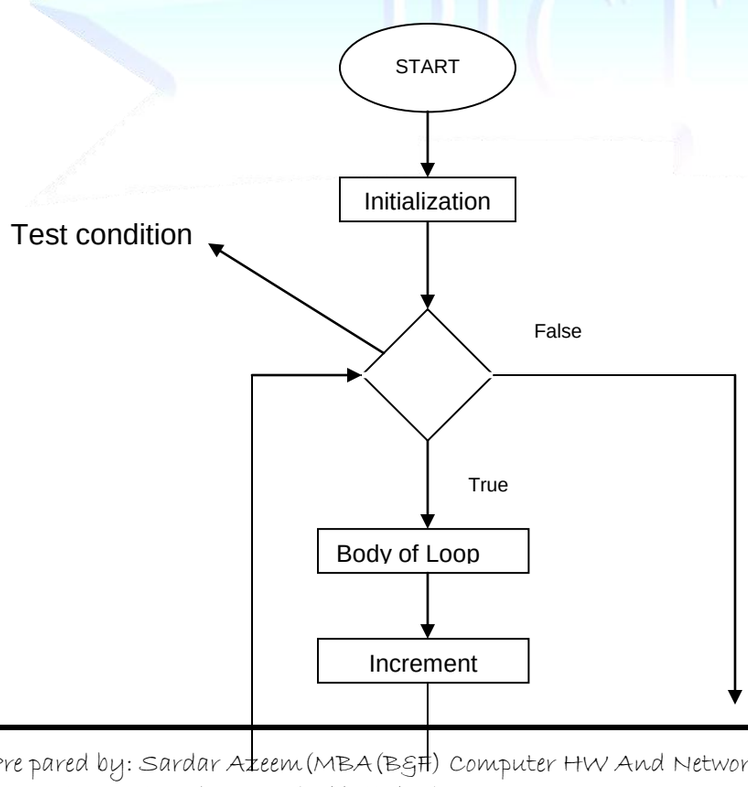
Syntax :



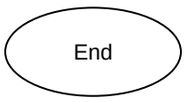
The main function of loop is to repeat statement while condition remains true. It works in the following way:

1. *For :* The for is reserved key word
2. *Initialization:* It is an initial value setting for a counter variable. This is executed only once.
3. *Condition : It* is used to check the value . If it is true the loop continues, otherwise the loop ends and statement is skipped (not executed).
4. *Statement :* it is executed when condition is true it can be either a single statement or a block enclosed in braces { }.
5. *Increase or Decrease :* it is to increase or decrease the value initialized value

Flow chart :



Pre pared by: Sardar Azeem (MBA (BGH) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com



1. Initialization takes place.
2. Condition is checked.
3. If condition is true the body of for loop is executed.
4. Increment or decrement takes place.
5. the condition is checked again
6. Step 2 and 3 are checked again until the condition become false.
7. The program will end.

Example C:

```c
// countdown using a for loop
#include <stdio.h>
void main ()
{
  for (int n=10; n>0; n--)
    {
      printf( n << ", ");
    }
  printf( "FIRE!\n");
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2,
1, FIRE!
```

Example C++

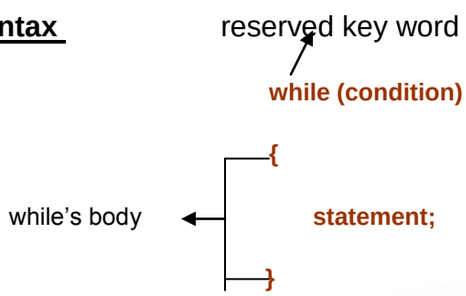
```cpp
// countdown using a for loop
#include <iostream>
using namespace std;
int main ()
{
  for (int n=10; n>0; n--) {
    cout << n << ", ";
  }
  cout << "FIRE!\n";
  return 0;
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
FIRE!
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

While loop: The while loop is used when number of repetitions are not known to the programmer.

Syntax



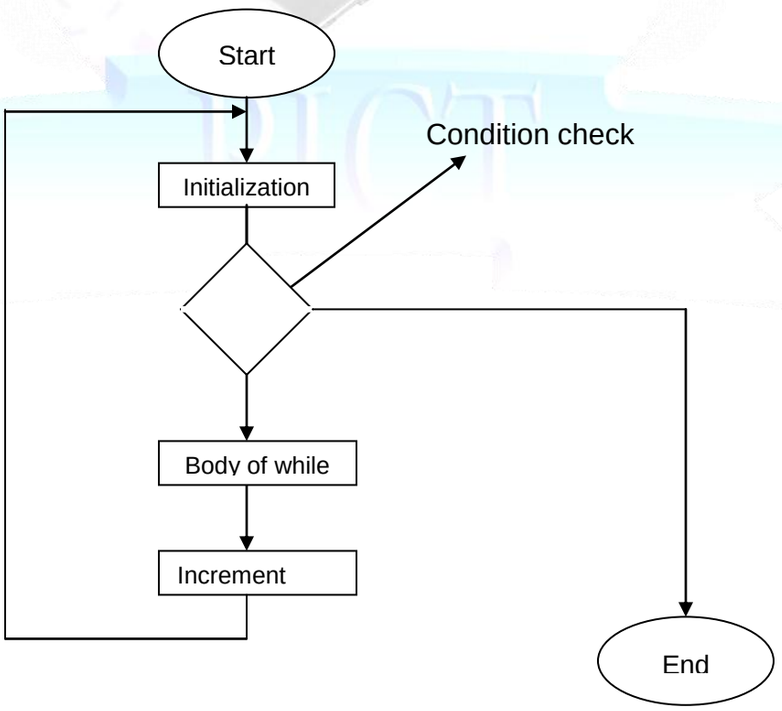
1 .While: Here while is reserved key word.

2. Condition: It is used to check the value . If it is true the loop continues, otherwise the loop ends and statement is skipped (not executed).

3. Statement: it is executed when condition is true it can be either a single statement or a block enclosed in braces { }.

When creating a while-loop, we must always consider that it has to end at some point, therefore we must provide within the block some method to force the condition to become false at some point, otherwise the loop will continue looping forever.

Flow chart



1. Initialization takes place.
2. Condition is checked.
3. If condition is true the body of while loop is executed.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

4. Increment or decrement takes place.
5. the condition is checked again
6. Step 2 and 3 are checked again until the condition become false.
7. The program will end.

Example For example, we are going to make a program to countdown using a while-loop:

```
// custom countdown using while
#include <stdio.h>
void main ()
 {
   int n;
   printf(“ "Enter the starting number  )";
   scanf(“%d”,&n);
   while (n>0)
        {
            printf (“ %d”,&n);
            --n;
         }
  printf ("FIRE!\n");
 }
```

```
Enter the starting number > 8
8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```

When the program starts the user is prompted to insert a starting number for the countdown. Then the while loop begins, if the value entered by the user fulfills the condition n>0 (that n is greater than zero) the block that follows the condition will be executed and repeated while the condition (n>0) remains being true.

The whole process of the previous program can be interpreted according to the following script (beginning in main):

1. User assigns a value to n
2. The while condition is checked (n>0). At this point there are two posibilities:
   * condition is true: statement is executed (to step 3)
   * condition is false: ignore statement and continue after it (to step 5)
3. Execute statement: printf(“n<< ", " ; --n;
   (prints the value of n on the screen and decreases n by 1)
4. End of block. Return automatically to step 2
5. Continue the program right after the block: print FIRE! and end program.

Example C++

```
// custom countdown using while

#include <iostream>
using namespace std;

int main ()
{
  int n;
  cout << "Enter the starting number
> ";
  cin >> n;

  while (n>0) {
    cout << n << ", ";
    --n;
  }

  cout << "FIRE!\n";
```

```
Enter the starting number > 8
8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```

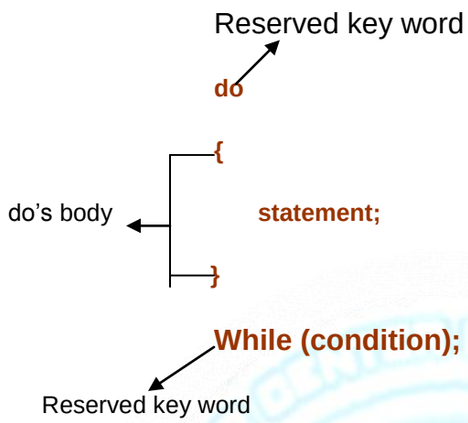

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

```
  return 0;
}
```

The do-while loop

Its functionality is exactly the same as the while loop, except that condition in the do-while loop is executed after the execution of body instead before, granting at least one execution of statement even if condition is never fulfilled.

Syntax:



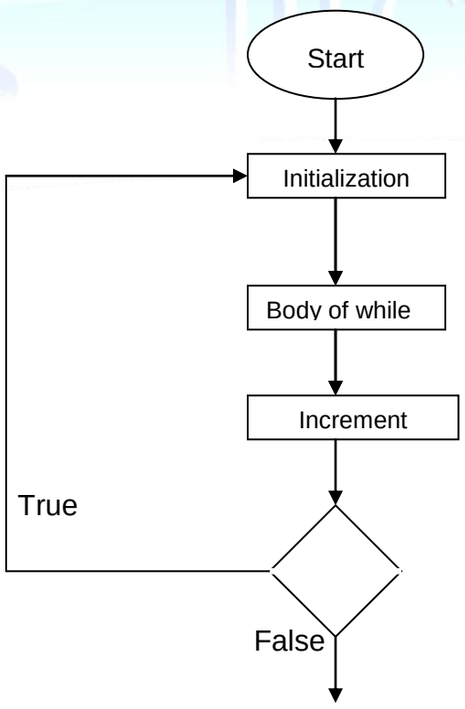
1 . do: Here do is reserved key word.

2. Statement: it is executed when condition is true it can be either a single statement or a block enclosed in braces { }.

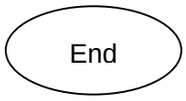
3. While: Here while is reserved key word. Remember while is always terminated in do-while statement by semicolon (;)

4. Condition: It is used to check the value. If it is true the loop continues, otherwise the loop ends and statement is skipped (not executed).

Flow chart



Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com



1. Initialization takes place.
2. The body of for do-while loop is executed.
3. Increment or decrement takes place.
4. Condition is checked.
5. If condition is true then body of loop is executed else it will terminate.
6. If condition is true then Step 2 and 3 are executed again until the condition become false.
7. The program will end

Example

```
// number echoer
#include <stdio.h>
  void main ()
  {
  unsigned long n;
  do
     {
        printf(“ "Enter number 0 to end): ";
        scanf(“%d”,&n);
        printf( "You entered: %d\n" ,n);
     }
  while (n != 0);

  }
```

```
Enter number (0 to end): 12345
You entered: 12345
Enter number (0 to end): 160277
You entered: 160277
Enter number (0 to end): 0
You entered: 0
```

The do-while loop is usually used when the condition that has to determine the end of the loop is determined

C++ Example

```
// number echoer

#include <iostream>
using namespace std;

int main ()
{
  unsigned long n;
  do {
    cout << "Enter number (0 to end):
";
    cin >> n;
    cout << "You entered: " << n <<
"\n";
  } while (n != 0);
  return 0;
}
```

```
Enter number (0 to end): 12345
You entered: 12345
Enter number (0 to end): 160277
You entered: 160277
Enter number (0 to end): 0
You entered: 0
```

The Break Statement:

Sometimes while executing a loop it becomes desirable to skip a part of the loop or quit the loop as soon as certain condition occurs, for example consider searching a particular number in a set of 100 numbers as soon as the search number is found it is desirable to terminate the loop. C language permits a jump from one statement to another within a loop as well as to jump out of the loop. The break statement allows us to accomplish this task. A break

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

statement provides an early exit from for, while, do and switch constructs. A break causes the innermost enclosing loop or switch to be exited immediately. Example program to illustrate the use of break statement.

```
#include < stdio.h

void main()

{
int I, num=0;
float sum=0,average;
printf("Input the marks, -1 to end\n");
while(1)
{
scanf("%d",&I);
if(I==-1)
break;
sum+=I;
num++ ;
}}
```

C++ Program

```
// break loop example

#include <iostream>
using namespace std;

int main ()
{
  int n;
  for (n=10; n>0; n--)
  {
    cout << n << ", ";
    if (n==3)
    {
      cout << "countdown aborted!";
      break;
    }
  }
  return 0;
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, countdown
aborted!
```

Continue statement:

During loop operations it may be necessary to skip a part of the body of the loop under certain conditions. Like the break statement C supports similar statement called continue statement. The continue statement causes the loop to be continued with the next iteration after skipping any statement in between. The continue with the next iteration the format of the continue statement is simply:

```
Continue;
```

Consider the following program that finds the sum of five positive integers. If a negative number is entered, the sum is not performed since the remaining part of the loop is skipped using continue statement.

```
#include < stdio.h >
void main()
{
int I=1, num, sum=0;
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

```
for (I = 0; I < 5; I++)
{
printf("Enter the integer");
scanf("%I", &num);
if(num < 0) zero
{
printf("You have entered a negative number");

continue;

}
sum+=num;
}
printf("The sum of positive numbers entered = %d",sum);
}
```

C++ Example

```
// continue loop example
#include <iostream>
using namespace std;

int main ()
{
  for (int n=10; n>0; n--) {
    if (n==5) continue;
    cout << n << ", ";
  }
  cout << "FIRE!\n";
  return 0;
}
```

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

CHAPTER 5.
STANDARD LIBRARY FUNCTIONS

1. Arithmetic functions:

- The arithmetic functions calculate common arithmetic functions which are not directly supported by the language.

Header Files

math.h

(abs, labs, fabs, ceil, floor, sqrt)

Prototype

```
int abs (int Value);

float abs (float Value);

double abs (double Value);

long double abs (long double Value);

float ceil (float Value);

double ceil (double Value);

long double ceil (long double Value);

float fabs (float Value);

double fabs (double Value);

long double fabs (long double Value);

float floor (float Value);

double floor (double Value);

long double floor (long double Value);

long labs (long Value);

float sqrt (float Value);

double sqrt (double Value);

long double sqrt (long double Value);
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Arguments

Value

The arithmetic value to which the function is to be applied.

Return Values

abs

Absolute value of an integer

labs

Absolute value of a long integer

fabs

Absolute value of a double

ceil

The ceiling of a double

floor

The floor of a double

sqrt

The square root of a double

Example

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void Display (double);
void DisplayInt (int Value);
void DisplayLong (long Value);

int main () {
  DisplayInt (rand () - rand());
  DisplayLong ((long)(rand () - rand()));
  Display (.16 * (double)(rand() % 100) - 8.0);
  DisplayInt (rand () - rand());
  DisplayLong ((long)(rand () - rand()));
  Display (.16 * (double)(rand() % 100) - 8.0);
  DisplayInt (rand () - rand());
  DisplayLong ((long)(rand () - rand()));
  Display (.16 * (double)(rand() % 100) - 8.0);
  return 0;
}

void DisplayInt (int Value) {
  printf ("The absolute value of int %i is %i\n",
          Value, abs (Value));
}

void DisplayLong (long Value) {
  printf ("The absolute value of long %li is %li\n",
          Value, abs (Value));
}

void Display (double Value) {
  printf ("The absolute value of double %f is %f\n",
          Value, fabs (Value));
  printf ("The ceiling of %f is %f\n",
          Value, ceil (Value));
  printf ("The floor of %f is %f\n",
          Value, floor (Value));
  printf ("The square root of the absolute value of %f is %f\n\n",
          Value, sqrt (fabs (Value)));
}
```

2. String functions

strings are stored as arrays of characters, with the end being marked by a NULL character (binary zero.) The standard C++ libraries include a number of functions to copy, compare, convert, and convert ASCIIZ strings.

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

→ Convert	ASCIIZ Conversion Functions	strupr, strlwr
→ Compare	Compare Strings	strcmp. strncmp, strcmpi, stricmp
→ Copy	Copy Strings	strcpy, strncpy, strdup

CONVERT STRING FUNCTIONS

The strlwr and strupr functions convert ASCIIZ strings to all upper case or all lower case. Non-alphabetic characters in the string are not affected.

Usage

These functions are used to convert ASCIIZ strings to all upper case or all lower case.

Header Files

string.h

strlwr, strupr

string

std::strlwr, std::strupr

Prototype

```
char * strlwr ( char * S );

char * strupr ( char * S );
```

Arguments

A pointer to the string to be comverred.

Return Values

strlwr

A pointer to the argument string after conversion to lower case.

strupr

A pointer to the argument string after conversion to upper case.

Side Effects

strlwr

The argument string is converted to lower case.

strupr

The argument string is converted to upper case.

Example

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

int main () {
  char Hello [] = "Hello";
  char * Lower;

  cout << "The initial string is: " << Hello << "\n";
  strupr (Hello);
  cout << "The upper case string is: " << Hello << "\n";
  Lower = strlwr (Hello);
  cout << "The lower case string is: " << Lower << "\n";

  return 0;
}
```


Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

COMPARE STRING FUNCTIONS

These functions are used to cmpare ASCIIZ strings, usually for sorting or ordering.

Header Files

string.h

strcmp, strcmpi, stricmp, strncmp

Prototype

```
int strcmp ( const char * S1, const char * S2 );

int strcmpi ( const char * S1, const char * S2 );

int stricmp ( const char * S1, const char * S2 );

int strncmp ( const char * S1, const char * S2, size_t Count );
```

Arguments

Count

The maximum number of characters to be compared.

S1

A pointer to the first string to be compared.

S2

A pointer to the second string to be compared.

COPY STRING FUNCTIONS

Header Files

string.h

strcpy, strdup, strncpy

Prototype

```
char * strcpy ( char * Destination, const char * Source );

char * strdup ( const char * Source );

char * strncpy ( char * Destination, const char * Source, size_t count );
```

Arguments

Count

The maximum number of characters to be copied.

Destination

A pointer to the receiving field.

Source

A pointer to the string to be copied.

Return Values

strcpy

A pointer to the receiving field.

strdup

A pointer to the receiving field.

strncpy

A pointer to the receiving field.

TRIGNOMETRIC FUNCTIONS:

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com

Purpose

The trigonometric functions calculate the sine, cosine, or tangent of an angle.

Header File

math.h

(sin, cos, tan)

Prototypes

```
double sin (double Angle);
double cos (double Angle);
double tan (double Angle);
```

Argument

Angle

The angle in radians

Return Values

sin

sine (Angle)

cos

cosine (Angle)

tan

tangent (Angle)

Example

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void Display (double Angle);

int main () {
  Display (.08 * (double)(rand() % 100) - 4.0);
  Display (.08 * (double)(rand() % 100) - 4.0);
  Display (.08 * (double)(rand() % 100) - 4.0);
  return 0;
}

void Display (double Angle) {
  printf ("The sine of %f radians is %f\n",
          Angle, sin (Angle));
  printf ("The cosine of %f radians is %f\n",
          Angle, cos (Angle));
  printf ("The tangent of %f radians is %f\n\n",
          Angle, tan (Angle));
}
```

Pre pared by: Sardar Azeem (MBA (BGF) Computer HW And Network Engineer: Pict Computer Center Link Road Abbottabad.
Email: Azeem_aag2000@yahoo.com/Website:www.pictcomputeratd.com