**C Language**

**What is C?**

C is a programming language developed at AT & T’s Bell Laboratories of USA in 1972. It was designed and written by a man named “*Dennis Ritchie”.* C seems so popular is because it is reliable, simple and easy to use. Major parts of popular operating systems like Windows, UNIX, and Linux are still written in C. C programming language provides system level access. If anyone extends the operating system to work with new devices one needs to write device driver programs. These programs are exclusively written in C.

[Mobile devices like cellular phones and palmtops have because rage of today. Also, common consumer devices like microwave ovens, washing machines, and digital cameras are getting smarter by the day. This smartness comes from a microprocessor, an operating system and a program embedded in these devices. These programs not only have to run faster but also have to work in limited amount of memory.

At times one is required to very closely interact with the hardware devices. Since C provides several language elements that make this interaction feasible without compromising the performance it is the preferred choice of the programmer]

**History of C**

C is one of the most popular computer languages today because it is a structured, high level, machine independent language. It allows software developers to develop programs without worrying about the hardware platforms where they will be implemented.

The root of all modern languages is ALGOL, introduced in the early 1960s. ALGOL was the first computer language to use a block structure. Although it never became popular in USA, it was widely used in Europe. ALGOL gave the concept of structured programming to the computer science community

In 1967, Martin Richards developed a language called BCPL (Basic combined programming language) primarily for writing system software. In 1970 Ken Thompson designed a language using many features of BCPL and it was called simply B. B was used to design early version of UNIX operating system at Bell Laboratories

**C was evolved from ALGOL, BCPL and B by Dennis Ritchie at Bell Laboratories in 1972. C uses many concepts from these languages and added the concept of data types and other powerful features. Since it was developed along with the UNIX operating system, it was strongly associated with UNIX. This operating system, which was also developed in Bell Laboratories, was coded almost entirely in C.**

**ANSI C**

During 1970s, C had evolved into “***traditional C”*.** The rapid growth of C led to the development of different versions of the language that were similar but often incompatible. This posed a serious problem for system developers.

To assure that the C language remains standard, in 1983, American National Standards Institute (ANSI) appointed a technical committee to define a standard for C. The ANSI committee approved a version of C in December 1989 which is now known as ANSI C. It was then approved by International Standards Organization (ISO). This version of C is also referred to as C89. Although C++ and Java were evolved out of C, the standardization committee of C felt that a few features of C++/Java, if added to C, would enhance the usefulness of C language. The result was the 1999 Standard for C. This version is usually referred to as C99. The history and development of C is illustrated in following Fig 1.1

+

ALGOL

BCPL

B

Traditional C

K&R C

ANSI C

ANSI/ISO C

C99

1960

1967

1970

1972

1978

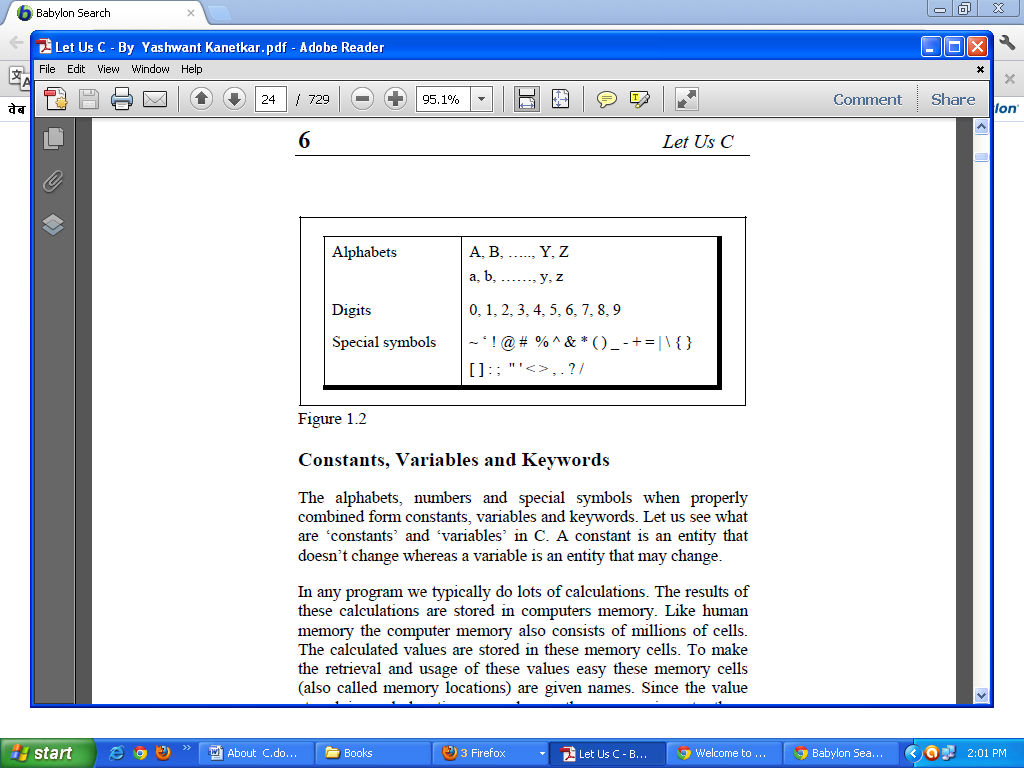
1989

1990

1999

**C CHARACTER SET**

A character denotes any alphabet, digit or special symbol used to represent information. Following table shows the valid alphabets, numbers and special symbols allowed in C



**IDENTIFIERS**

Identifiers are the names that are given to various program elements such as variables, symbolic constants and functions. Variable or function identifier that is called a symbolic constant name here are the rules you need to know:

1. Identifier name must be a sequence of letter and digits, and must begin with a letter.
2. The underscore character (‘\_’) is considered as letter.
3. Names shouldn't be a keyword (such as int , float, if ,break, for etc)
4. Both upper-case letter and lower-case letter characters are allowed. However, they're not interchangeable.
5. No identifier may be keyword.
6. No special characters, such as semicolon,period,blank space, slash or comma are permitted

**Legal identifiers:**

float \_number;

float a;

int this\_is\_a\_very\_detailed\_name\_for\_an\_identifier;

**Illegal** (it's your job to recognize why):

float :e;

float for;

float 9PI;

float .3.14;

float 7g;

**CONSTANTS**

Constants in C refer to fixed values that do not change during the execution of a program. C supports two types of constants.

1. Primary constant
2. Secondary constant

These constant are further categorized as shown-

C Constant

Primary Constant

Secondary Constant

Integer Constant

Real Constant

Character Constant

Array

Pointer

Structure

Union

Enum Etc

1. **Rules to construct integer constant**
   1. An integer constant must have at least one digit
   2. It must not have a decimal point
   3. It can be either positive or negative
   4. If no sign precedes an integer constant, it is assumed to be positive
   5. No commas or blanks are allowed within an integer constant
   6. The allowable range for integer is -32768 to 32767
2. **Rules to construct real constants**
   1. A real constant must have at least one digit
   2. It must have a decimal point
   3. It could be either positive or negative
   4. Default sign is positive
   5. No commas or blanks are allowed within a real constant
3. **Rules to construct character constant**
   1. A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas. Both the inverted commas should point to the left. For example ‘A’
   2. The maximum length of a character constant can be 1 character

**VARAIBLES**

A variable is an identifier that denotes a storage location used to store a data value. A variable is a data name that may be used to store a data value. A variable may take different values at different times during execution of program. Or a variable is an entity that may vary during program execution. Variable names are names given to locations in memory.

**RULES FOR CONSTRUCTING VARIABLES NAMES**

1. A variable name is any combination of 1 to 31 alphabets, digits or underscores. (Some compilers allow variables names whose length could be up to 247 character)
2. The first character in the variable name must be an alphabet or underscore
3. No commas or blanks are allowed within a variable name
4. No special symbol other the underscore can be used in a variable name.

**KEY WORDS**

Keywords are the reserved words whose meaning has been already been explained to the C compiler. The keywords cannot be used as variable names because if we do so, we are trying to assign a new meaning to the keyword, which is not allowed by the computer. There are only 32 keywords available in C

**auto break int long**

**Case char register return**

**Const continue short signed**

**Default do sizeof static**

**Double else struct switch**

**Enum extern typedef union**

**float for unsigned void**

**Goto if volatile while**

**DATA TYPES**

Data Types in a programming language describes that what type of data a variable can hold. C supports 5 fundamental data types’ integer (int), character (char), and floating point (float), double precision floating point (double) and void.

|  |  |  |
| --- | --- | --- |
| **Data type** | **Size (in byte)** | **Range** |
| Int | 2 byte | -32768 to 32767 |
| Float | 4 byte | 3.4\*10-38 to 3.4\*10+38 |
| Char | 1 byte | -128 to +127 |
| Long | 4 byte | -2147483648 to +2147483647 |
| Double | 8 byte | 1.7\*10-308 to 1.7\*10+308 |

**BASIC STRUCTURE OF C PROGRAM**

|  |  |
| --- | --- |
| Documentation section  Link Section  Definition Section  Global declaration  main() function section  {  Declaration part  Executable part  } | Sub program section  Function 1  Function 2  Function 3 |

**OPERATORS**

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. They usually form a part of mathematical of logical expressions. They are of following types-

1. **Arithmetic operators:** C# includes the following arithmetic operators for mathematical operations

|  |  |
| --- | --- |
| **Operators** | **Meanings** |
| **+** | Addition |
| **-** | Subtraction |
| **\*** | Multiplication |
| **/** | Division |
| **%** | Modulo division |

1. **Relational Operators:** Relational operators are used to compare two quantities

|  |  |
| --- | --- |
| **Operators** | **Meanings** |
| **<** | Is less than |
| **<=** | Is less than or equl to |
| **>** | Is greator than |
| **>=** | Is greator than or equal to |
| **==** | Is equal to |
| **!=** | Is not equal to |

1. **Logical Operators:** Logical operators are used to test one or more conditions

|  |  |
| --- | --- |
| **Operators** | **Meanings** |
| **&&** | Logical AND |
| **||** | Logical OR |
| **!** | Logical NOT |

1. **Assignment Operator:** Assignment operator is used to assign the value of an expression or variable to another variable.

|  |  |
| --- | --- |
| **Operators** | **Meanings** |
| **=** | Assignment |

1. **Increment / decrement operators:** C provides two very useful operators. These are increment and decrement

|  |  |
| --- | --- |
| **Operators** | **Meanings** |
| **++** | Increment |
| **--** | Decrement |

1. **Conditional operators:** Conditional operator is a ternary operator. Which is a combination of two operators “? :”

|  |  |
| --- | --- |
| **Operators** | **Meanings** |
| **Exp1? exp2 : exp3** | If exp1 is true exp2 will be returned else exp3 will be returned |

1. **Bitwise operators:** Bitwise operators are used for manipulation of data at bit level. These operators are used for testing the bits, or shifting them left or right.

|  |  |  |
| --- | --- | --- |
| **Operators** | **Meanings** |  |
| **&** | **Bitwise Logical AND** | **%d for decimal** |
| **|** | **Bitwise Logical OR** | **%o for octal** |
| **^** | **Bitwise Logical XOR** | **%x for hexadecimal** |
| **<<** | **Shift left** |  |
| **>>** | **Shift right** |  |

1. **Special operators**

|  |  |
| --- | --- |
| **Operators** | **Meanings** |
| **Typeof** | Type of operator tells type |
| **Sizeof** | Sizeof operator count the size |
| **.(dot)** | Member access operator |
| **,(comma)** | To link related expressions together |

**TYPES OF OPERATOR (BASED ON OPERANDS)**

The operators can be categorized into two categories

1. Unary operators
2. Binary operators
3. Ternary operators

**UNARY OPERATORS:** The unary operators require only one operand to perform an operation.

|  |  |
| --- | --- |
| **Operators** | **Meanings** |
| **++** | Increment |
| **--** | Decrement |
| **-** | Negative sign |

**BINARY OPERATORS:** The binary operators require two operands to perform an operation.

|  |  |
| --- | --- |
| **Operators** | **Meanings** |
| **+** | Addition |
| **-** | Subtraction |
| **\*** | Multiplication |
| **/** | Division |
| **%** | Modulo division |

**TERNARY OPERATORS:** The ternary operator requires three operands to perform an operation.

|  |  |
| --- | --- |
| **Operators** | **Meanings** |
| **Exp1? exp2 : exp3** | If exp1 is true exp2 will be returned else exp3 will be returned |

**HIERARCHY OF OPERATORS**

The priority or precedence of operators in which the operations in an arithmetic statement are performed is called the hierarchy of operations.

|  |  |  |
| --- | --- | --- |
| Priority | Operator | Description |
| 1st | \* / % | Multiplication, division, modulus division |
| 2nd | + | Addition, subtraction |
| 3rd | = | Assignment |

**THE DECISION CONTROL STRUCTURE**

C program is a set of statements which are normally executed sequentially in the order in which they appear. This happens when no options or repetitions of certain calculations are necessary. Sometimes we have a number of situations where we may have to change the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met. This involves a kind of decision making to see whether a particular condition has occurred or not and then direct the computer to execute certain statements accordingly.

C language gives some decision making capabilities by supporting the following statements:

1. If statement
2. Switch statement
3. Conditional operator statement
4. goto statement

These statements are known as decision making statements. Since these statements control the flow of execution, they are also known as control statements.

**IF STATEMENT**

The if statement is a powerful decision making statement and is used to control the flow of execution of statements. This statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are:

1. simple **if** statement
2. **if…..else** statement
3. **nested if…..else** statement
4. **else if** clause
5. **Simple If Statement**

It is basically a two way decision making statement. It has the following syntax:

|  |
| --- |
| **if (test expression)**  **{ //true block**  **//logic or code**  **}** |
| **if(test expression)**  **Single line logic/code** |

It allows the computer to evaluate the expression first and then, (depending on whatever the value of the expression is true of false), transfers the control to a particular statement. The logical code may be a single line or a group of statements. If the test expression is true the logical code/statements will be executed otherwise the logical code will be skipped and the control of compiler will be transferred to the next statement after the if block.

Statement 1

Statement 2

**True**

**Test expression**

**ENTRY**

**False**

1. **The i*f…else* Statement**

The if….else statement is the extension of the simple if statement. The general form if….else statement is:-

|  |
| --- |
| **if (test expression)**  **{ //true block**  **//logic or code**  **}**  **else**  **{ //false block**  **//logic or code**  **}** |
| **if(test expression)**  **Single line logic/code**  **else**  **single line logic/code** |

In if…else statement the test expression is true then the true block statements are executed otherwise the false block statements are executed and the control of compiler will be sent to the statement which is immediate after if…..else statement. Flow chart for if…..else is as:

Statement 1

Statement 2

**True**

**Test expression**

**ENTRY**

**False**

Statement 1

1. **Nested if…..else Statement**

When a series of decisions are involved then we have to use more than one if….else statement in nested form.

**False**

Statement 1

Statement 2

**False**

**Condition 1**

**ENTRY**

**True**

Statement 3

**Condition 2**

Statement X

**True**

1. ***else…if clause***

There is another way of putting if statements together when multiple decisions are involved. A multiple decision is a chain of if statements.

|  |
| --- |
| if (condition 1)  statement 1  else if (condition 2)  statement 2  else if (condition 3)  statement 3  else  statement 4 |

1. An electric power distribution company charges its domestic consumers as follows:

|  |  |  |
| --- | --- | --- |
| Consumption units | Rate of charges | Extra charges |
| 000-200 unit | Rs. 0.50 per unit |  |
| 201-400 unit | Rs. 0.65 per unit | Rs. 100 |
| 401-600 unit | Rs. 0.80 per unit | Rs. 230 |
| 601 and above | Rs. 1.00 per unit | Rs. 390 |

**USE OF LOGICAL OPERATORS**

C allows usage of three logical operators which are && (AND), || (OR) and! (NOT). && (AND) operator and || (OR) operator is used to combine more than one condition in if statements

**THE SWITCH CASE CONTROL STRUCTURE**

C has a multi way decision making statement known as switch. The switch statement tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statement associated with that case is executed.

or

The control statement that allows us to make a decision from the number of choices is called a switch, or more correctly a switch–case. They most often appear as follows:

|  |  |
| --- | --- |
|  | switch(integer expression)  {  case value-1:  Block 1  break;  case value-2:  Block 2  break;  ………….  ………….  default:  Default-block;  break;  }  statement-x |

The integer expression following the keyword switch is any C expression that will yield an integer value. It could be an integer constant like 1, 2, or 3, or an expression that evaluates to an integer. The keyword **case** is followed by an integer or a character constant. It is known as case label. Each constant in each case must be different from all the others. The “does this” lines in the above from of switch represent any valid C statement.

When the **switch** is executed, the value of the expression is successfully compared against the values value-1, value-2… If a case is found whose value matches with the value of the expression, then the block of statement that follows the case are executed.

The **break** statement at the end of each block indicates the end of a particular case and causes an exit from the switch statement, transferring the control to the statement –x following the switch.

The **default** is an optional case .When present , it will be executed if the value of the expression does not match with any of the case value .If not present ,no acting takes place if all matches fail and the control goes to the statement-x.

**LOOPS**

***“An operation which involves repeating some portion of the program either a specified number of times, or until a particular condition is being satisfied is known as repetitive operation or looping operation”.*** A loop contains two segments one known as the body of loop and the other one is known as the control statement.

Depending on the position of the control statement in the loop a control structure may be classified either as **entry controlled loop** or as the **exit controlled loop**.

**Entry controlled loop:** - In entry controlled loop the control conditions are tested before the start of loop execution. If the conditions are not satisfied, then the body of loop will not be executed. It is also known as pre-test loop.

**Exit Controlled loop:** - In exit controlled loop the control conditions are tested at the end of the body of loop and therefore the body is executed unconditionally for the first time. It is also known as post-test loop.

***A loop generally includes the following four steps:***

1. Initialization of the condition variable
2. Execution of the statements in the body of loop
3. Test for a specified value of the condition variable for the execution of loop
4. Increment or decrement or updating the condition variable

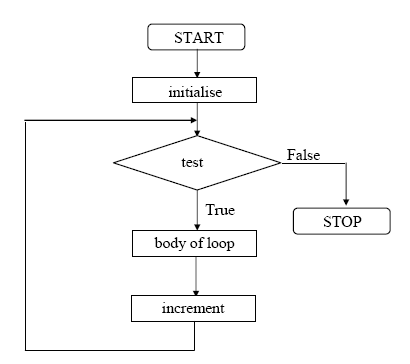
C provides three types of loops they are given as:

* **while loop**
* **for loop**
* **do-while loop**

**The while Loop**

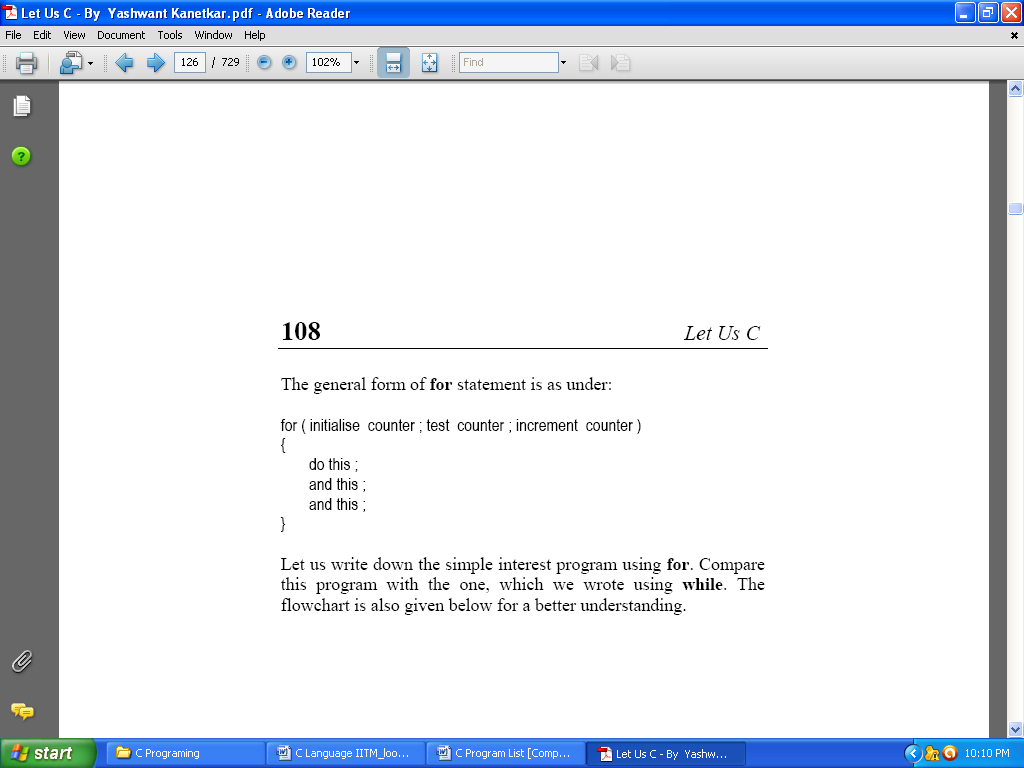
The simplest loop in all the loops in C is the while loop. It is an entry-controlled loop statement. The test condition is evaluated first and if the condition is true then the body of the loop is executed. After this the test condition is once again evaluated if it is true the body is executed again and this process of repeated execution of body of loop continues until the test condition is becomes false and then the control of compiler is transferred out of the loop. Syntax of while loop is given as:

|  |
| --- |
| while(test condition)  {  //Logical statements  //Or body of the loop  } |



**The for Loop**

The for loop is another entry-controlled loop that provides a more concise loop control structure. The general syntax of the “for loop” is given as:

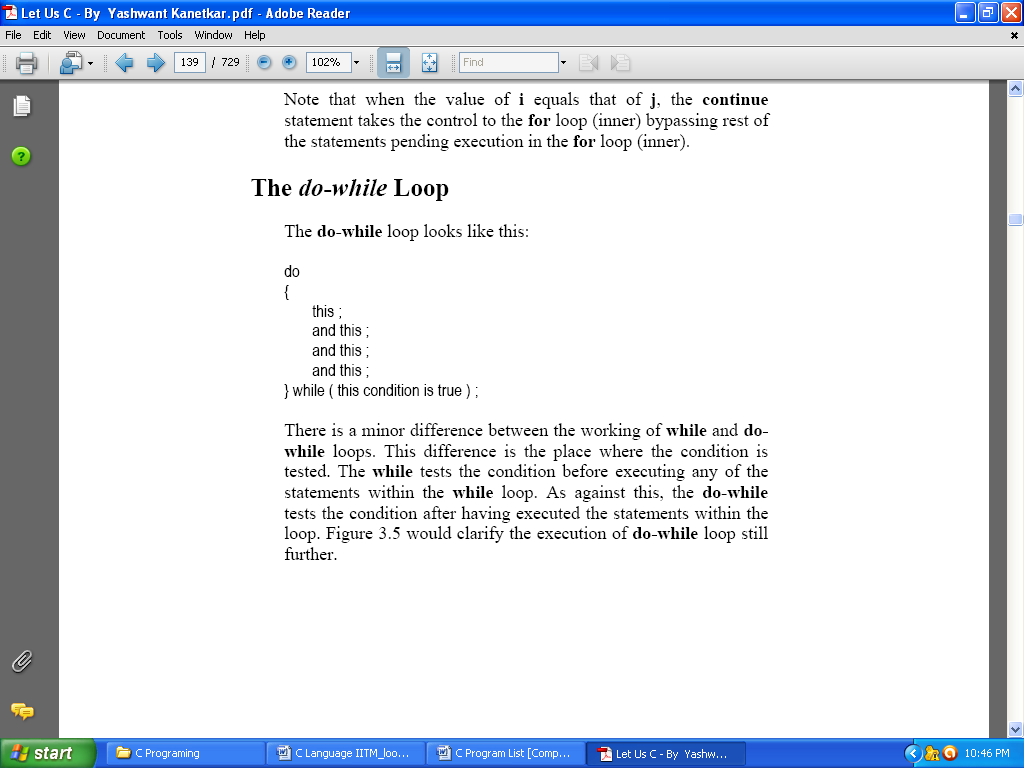


The execution of the “for loop” is as follows:

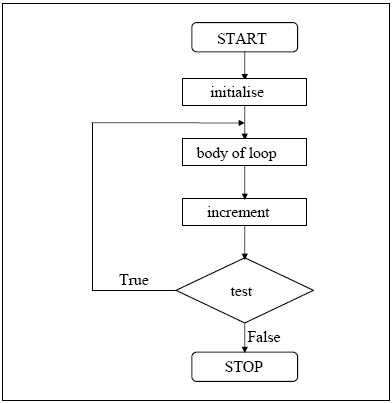
1. Initialization of the control variables is done first
2. The value of the control variable is tested using the test condition. It is a relational expression
3. When the body of the loop is executed, the control is transferred back to the “for statement”.
4. The control variable is updated (i.e. incremented or decrement).
5. New value of the counter variable is tested again by test-condition
6. This process continues till the value of the control variable fails to satisfy the test condition.

**The do-while Loop**

The do-while loop is another form of while loop. In which the body of the loop is executed first and the test condition is tested at the end. If the test condition is true then the program continues the execution of the body of loop once again. This process continues as long as the condition is true. It is an exit-controlled loop structure. The general syntax for the do-while loop is given as:

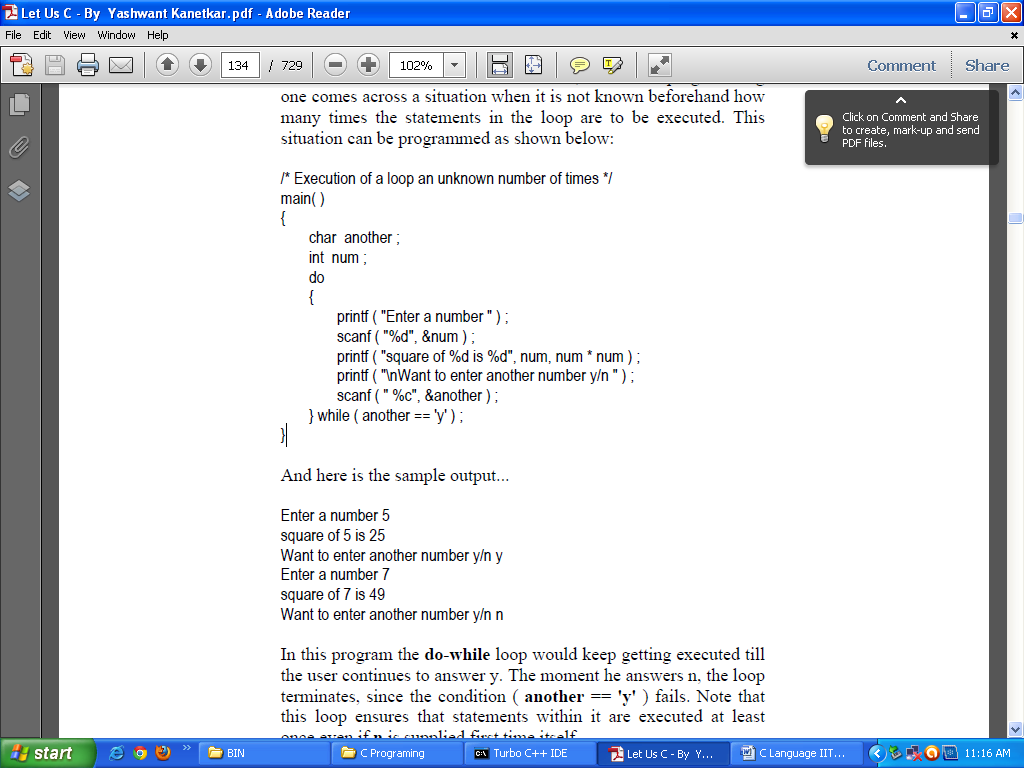


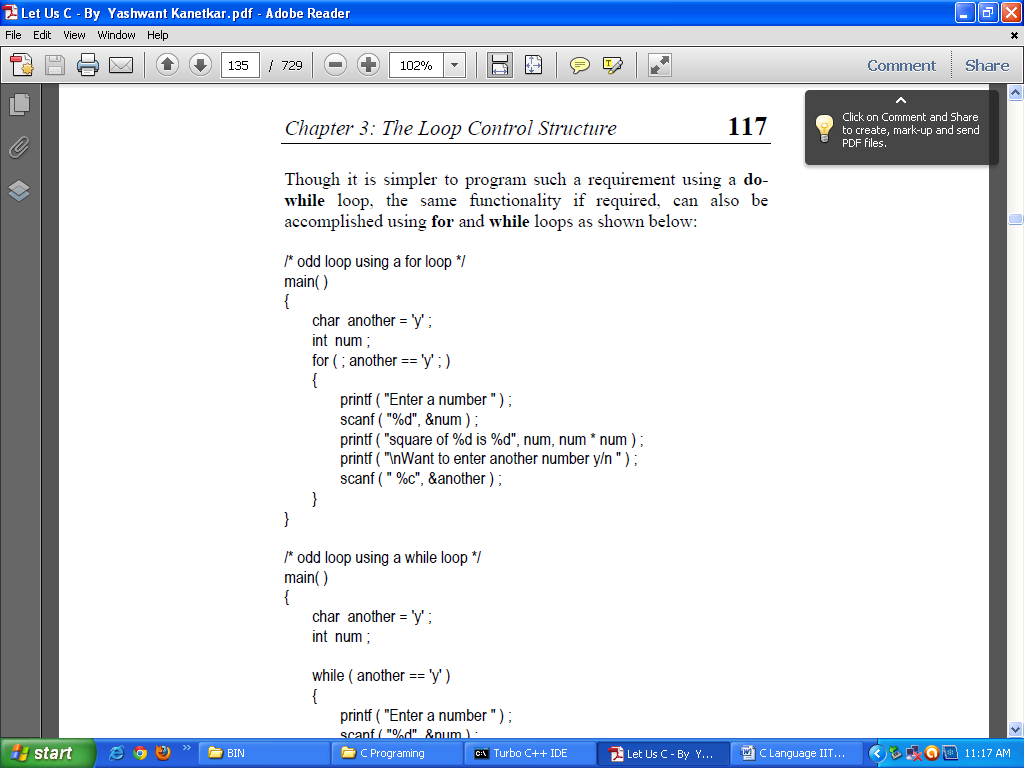
Flowchart for the do-while loop looks like:

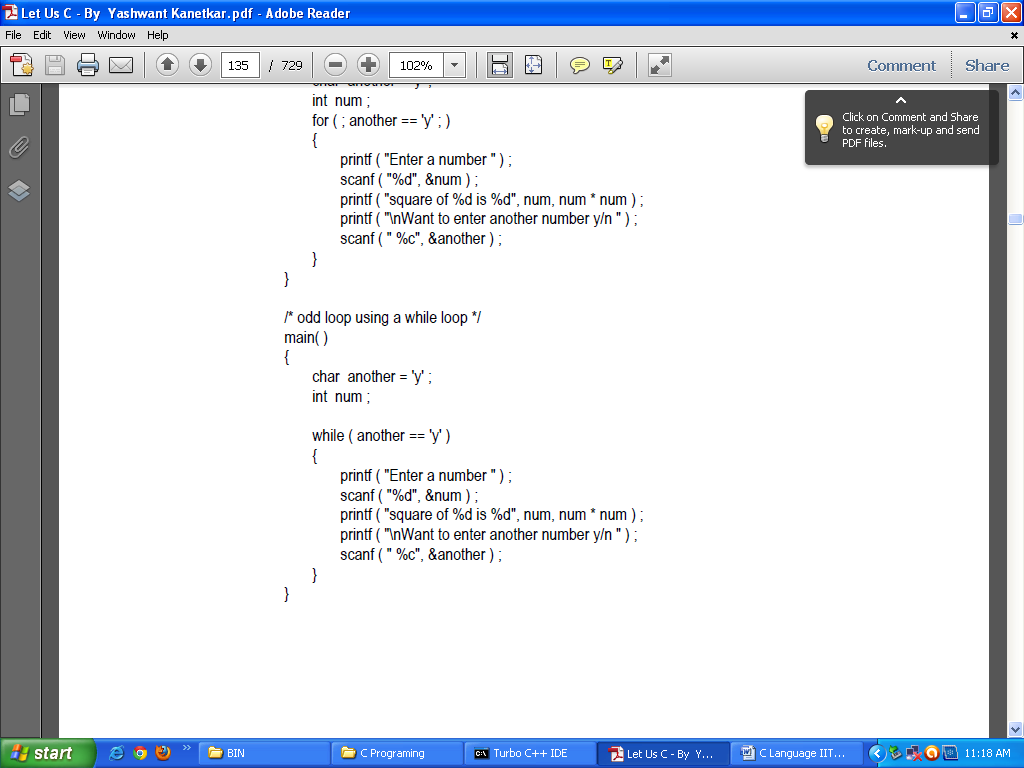


**The Odd Loop**

The loops that we have used so far executed the statements within them a finite number of times. However, in real life programming one comes across a situation when it is not known beforehand how many times the statements in the loop are to be executed.

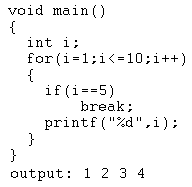






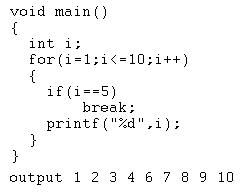
**The break statement**

*C break* statement is used to terminate any type of loop such as [while loop](http://cprogramminglanguage.net/c-while-loop-statement.aspx), [do while loop](http://cprogramminglanguage.net/c-do-while-loop-statement.aspx) and [*for* loop](http://cprogramminglanguage.net/c-for-loop-statement.aspx). C *break* statement terminates the loop body immediately and passes control to the next statement after the loop.



**The Continue Statement**

*C “continue”* statement is used to skip the current iteration in any type of loop as while, do-while and for loop. After *continue* statement, the control returns to the top of the loop.



**What is C function?**

     A large C program is divided into basic building blocks called C function. C function contains set of instructions enclosed by “{  }” which performs specific operation in a C program. Actually, Collection of these functions creates a C program.

**2. Uses of C functions:**

* + C functions are used to avoid rewriting same logic/code again and again in a program.
  + There is no limit in calling C functions to make use of same functionality wherever required.
  + We can call functions any number of times in a program and from any place in a program.
  + A large C program can easily be tracked when it is divided into functions.
  + The core concept of C functions are, re-usability, dividing a big task into small pieces to achieve the functionality and to improve understandability of very large C programs.

**3. C function declaration, function call and function definition:**

There are 3 aspects in each C function. They are,

* + Function declaration or prototype  - This informs compiler about the function name, function parameters and  return value’s data type.
  + Function call – This calls the actual function
  + Function definition – This contains all the statements to be executed.

| **S.no** | **C function aspects** | **syntax** |
| --- | --- | --- |
| 1 | function definition | return\_type function\_name ( arguments list ) { Body of function; } |
| 2 | function call | function\_name ( arguments list ); |
| 3 | function declaration | return\_type function\_name ( argument list ); |

**Simple example program for C function:**

* + As you know, functions should be declared and defined before calling in a C program.
  + In the below program, function “square” is called from main function.
  + The value of “m” is passed as argument to the function “square”. This value is multiplied by itself in this function and multiplied value “p” is returned to main function from function “square”.

|  |
| --- |
| #include<stdio.h>  // function prototype, also called function declaration  float square ( float x )    // main function, program starts from here  int main( )  {            float m, n ;          printf ( "\nEnter some number for finding square \n");          scanf ( "%f", &m ) ;          // function call          n = square ( m ) ;          printf ( "\nSquare of the given number %f is %f",m,n );    }    float square ( float x )   // function definition  {          float p ;          p = x \* x ;          return ( p ) ;  } |

**4. How to call C functions in a program?**

There are two ways that a C function can be called from a program. They are,

* 1. Call by value
  2. Call by reference

**1. Call by value:**

* + In call by value method, the value of the variable is passed to the function as parameter.
  + The value of the actual parameter can not be modified by formal parameter.
  + Different Memory is allocated for both actual and formal parameters. Because, value of actual parameter is copied to formal parameter.

Note:

* + Actual parameter – This is the argument which is used in function call.
  + Formal parameter – This is the argument which is used in function definition

**Example program for C function (using call by value):**

* + In this program, the values of the variables “m” and “n” are passed to the function “swap”.
  + These values are copied to formal parameters “a” and “b” in swap function and used.

|  |
| --- |
| #include<stdio.h>  // function prototype, also called function declaration  void swap(int a, int b);    int main()  {      int m = 22, n = 44;      // calling swap function by value      printf(" values before swap  m = %d \nand n = %d", m, n);      swap(m, n);  }    void swap(int a, int b)  {      int tmp;      tmp = a;      a = b;      b = tmp;      printf(" \nvalues after swap m = %d\n and n = %d", a, b);  } |

**2. Call by reference:**

* + In call by reference method, the address of the variable is passed to the function as parameter.
  + The value of the actual parameter can be modified by formal parameter.
  + Same memory is used for both actual and formal parameters since only address is used by both parameters.

**Example program for C function (using call by reference):**

* + In this program, the address of the variables “m” and “n” are passed to the function “swap”.
  + These values are not copied to formal parameters “a” and “b” in swap function.
  + Because, they are just holding the address of those variables.
  + This address is used to access and change the values of the variables.

|  |
| --- |
| #include<stdio.h>  // function prototype, also called function declaration  void swap(int \*a, int \*b);    int main()  {        int m = 22, n = 44;      //  calling swap function by reference      printf("values before swap m = %d \n and n = %d",m,n);      swap(&m, &n);    }    void swap(int \*a, int \*b)  {      int tmp;      tmp = \*a;      \*a = \*b;      \*b = tmp;      printf("\n values after swap a = %d \nand b = %d", \*a, \*b);  } |

**TYPE OF C FUNCTION**

* 1. C function with arguments (parameters) and with return value
  2. C function with arguments (parameters) and without return value
  3. C function without arguments (parameters) and without return value
  4. C function without arguments (parameters) and with return value

| **S.no** | **C function** | **syntax** |
| --- | --- | --- |
| 1 | with arguments and with return values | int function ( int );         // function declaration function ( a );                // function call int function( int a )       // function definition {statements;  return a;} |
| 2 | with arguments and without return values | void function ( int );     // function declaration function( a );                // function call void function( int a )   // function definition {statements;} |
| 3 | without arguments and without return values | void function();             // function declaration function();                     // function call void function()              // function definition {statements;} |
| 4 | without arguments and with return values | int function ( );             // function declaration function ( );                  // function call int function( )               // function definition {statements;  return a;} |

**Note:**

* + If the return data type of a function is “void”, then, it can’t return any values to the calling function.
  + If the return data type of the function is other than void such as “int, float, double etc”, then, it can return values to the calling function.

**1. Example program for with arguments & with return value:**

      In this program, integer, array and string are passed as arguments to the function. The return type of this function is “int” and value of the variable “a” is returned from the function. The values for array and string are modified inside the function itself.

|  |
| --- |
| **#include<stdio.h>**  **#include<conio.h>**  **int add(int x, int y)**  **{**  **int result;**  **result = x+y;**  **return(result);**  **}**  **void main()**  **{**  **int z;**  **clrscr();**  **z = add(952,321);**  **printf("Result %d.\n\n",add(30,55));**  **printf("Result %d.\n\n",z);**  **getch();**  **}** |

**2. Example program for with arguments & without return value:**

      In this program, integer, array and string are passed as arguments to the function. The return type of this function is “void” and no values can be returned from the function. All the values of integer, array and string are manipulated and displayed inside the function itself.

|  |
| --- |
| #include<stdio.h>  #include<conio.h>  void add(int x, int y)  {  int result;  result = x+y;  printf("Sum of %d and %d is  %d.\n\n",x,y,result);  }  void main()  {  clrscr();  add(30,15);  add(63,49);  add(952,321);  getch();  } |

**3. Example program for without arguments & without return value:**

      In this program, no values are passed to the function “test” and no values are returned from this function to main function.

|  |
| --- |
| #include<stdio.h>    void test();    int main()  {      test();      return 0;  }    void test()  {         int a = 50, b = 80;         printf("\nvalues : a = %d and b = %d", a, b);  } |

**4. Example program for without arguments & with return value:**

      In this program, no arguments are passed to the function “sum”. But, values are returned from this function to main function. Values of the variable a and b are summed up in the function “sum” and the sum of these value is returned to the main function.

|  |
| --- |
| #include<stdio.h>    int sum();    int main()  {      int addition;      addition = sum();      printf("\nSum of two given values = %d", addition);      return 0;  }    int sum()  {         int a = 50, b = 80, sum;           sum = a + b;         return sum;  } |

**Do you know how many values can be return from C functions?**

* Always, Only one value can be returned from a function.
* If you try to return more than one values from a function, only one value will be returned that appears at the right most place of the return statement.
* For example, if you use “return a,b,c” in your function, value for c only will be returned and values a, b won’t be returned to the program.
* In case, if you want to return more than one values, pointers can be used to directly change the values in address instead of returning those values to the function.

**ARRAY**

Array is a collection of variables belongings to the same data type. You can store group of data of same data type in an array.

* + Array might be belonging to any of the data types
  + Array size must be a constant value.
  + Always, Contiguous (adjacent) memory locations are used to store array elements in memory.
  + It is a best practice to initialize an array to zero or null while declaring, if we don’t assign any values to array.

**Example for C Arrays:**

* + int a[10];       // integer array
  + char b[10];   // character array   i.e. string

**Types of C arrays:**

There are 2 types of C arrays. They are,

* 1. One dimensional array
  2. Multi dimensional array
     + Two dimensional array
     + Three dimensional array, four dimensional array etc…

**1. One dimensional array in C:**

* + Syntax : data-type arr\_name[array\_size];

|  |  |  |
| --- | --- | --- |
| **Array declaration** | **Array initialization** | **Accessing array** |
| Syntax:  data\_type arr\_name [arr\_size]; | data\_type arr\_name [arr\_size]= (value1, value2, value3,….); | arr\_name[index]; |
| int age [5]; | int age[5]={0, 1, 2, 3, 4, 5}; | age[0];\_/\*0\_is\_accessed\*/ age[1];\_/\*1\_is\_accessed\*/ age[2];\_/\*2\_is\_accessed\*/ |
| char str[10]; | char str[10]={‘H’,‘a’,‘i’}; (or) char str[0] = ‘H’; char str[1] = ‘a’; char str[2] = ‘i; | str[0];\_/\*H is accessed\*/ str[1];  /\*a is accessed\*/ str[2];  /\* i is accessed\*/ |

**Example program for one dimensional array in C:**

|  |
| --- |
| #include<stdio.h>    int main()  {      int i;      int arr[5] = {10,20,30,40,50};      // declaring and Initializing array in C      //To initialize all array elements to 0, use int arr[5]={0};      /\* Above array can be initialized as below also         arr[0] = 10;         arr[1] = 20;         arr[2] = 30;         arr[3] = 40;         arr[4] = 50;      \*/      for (i=0;i<5;i++)      {          // Accessing each variable          printf("value of arr[%d] is %d \n", i, arr[i]);      }  } |

**Two dimensional array in C:**

* + Two dimensional array is nothing but array of array.
  + syntax : data\_type array\_name[num\_of\_rows][num\_of\_column]

|  |  |  |  |
| --- | --- | --- | --- |
| **S.no** | **Array declaration** | **Array initialization** | **Accessing array** |
| 1 | Syntax:  data\_type arr\_name [num\_of\_rows][num\_of\_column]; | data\_type arr\_name[2][2] = {{0,0},{0,1},{1,0},{1,1}}; | arr\_name[index]; |
| 2 | Example: int arr[2][2]; | int arr[2][2] = {1,2, 3, 4}; | arr [0] [0] = 1;  arr [0] ]1] = 2; arr [1][0]  = 3; arr [1] [1] = 4; |

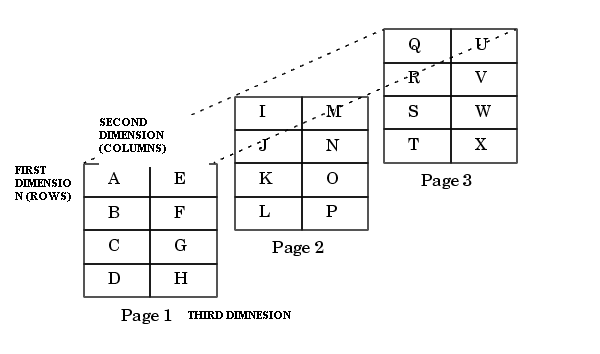
**Example program for two dimensional array in C:**

|  |
| --- |
| #include<stdio.h>  int main()  {      int i,j;      // declaring and Initializing array      int arr[2][2] = {10,20,30,40};      /\* Above array can be initialized as below also         arr[0][0] = 10;   // Initializing array         arr[0][1] = 20;         arr[1][0] = 30;         arr[1][1] = 40;      \*/      for (i=0;i<2;i++)      {         for (j=0;j<2;j++)         {            // Accessing variables            printf("value of arr[%d] [%d] : %d\n",i,j,arr[i][j]);         }      }  } |

**MULTI DIMENSIONAL ARRAYS**

* A multi dimensional array is an array of arrays.
* Like we have one index in a single dimensional array, two indices in a two dimensional array, in the same way we have n indices in a n-dimensional array or multi dimensional array.
* Conversely, an n dimensional array is specified using n indices.
* An n dimensional m1 x m2 x m3 x ….. mn array is a collection m1\*m2\*m3\* ….. \*mn elements.
* In a multi dimensional array, a particular element is specified by using n subscripts as **A[I1][I2][I3]…[In], where,**

**I1<=M1 I2<=M2 I3 <= M3 ……… In <= Mn**



**PROGRAM TO READ AND DISPLAY A 2X2X2 ARRAY**

**#include<stdio.h>**

**int main()**

**{ int array1[2][2][2], i, j, k;**

**printf(“\n Enter the elements of the matrix”);**

**for(i=0;i<2;i++)**

**{ for(j=0;j<2;j++)**

**{**

**for(k=0;k<2;k++)**

**{**

**printf(“\n array[%d][ %d][ %d] = ”, i, j, k);**

**scanf(“%d”, &array1[i][j][k]);**

**}**

**}**

**}**

**printf(“\n The matrix is : “);**

**for(i=0;i<2;i++)**

**{ printf(“\n\n”);**

**for(j=0;j<2;j++)**

**{**

**printf(“\n”);**

**for(k=0;k<2;k++)**

**printf(“\t array[%d][ %d][ %d] = %d”, i, j, k, array1[i][j][k]);**

**}**

**}**

**}**

**Passing Arrays as Function parameters**

If you want to pass a single-dimension array as an argument in a function, you would have to declare function formal parameter in one of following three ways and all three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received. Similar way you can pass multi-dimensional array as formal parameters.

Way-1

Formal parameters as a pointer as follows. You will study what is pointer in next chapter.

void myFunction(int \*param)

{

.

.

.

}

Way-2

Formal parameters as a sized array as follows:

void myFunction(int param[10])

{

.

.

.

}

Way-3

Formal parameters as an unsized array as follows:

void myFunction(int param[])

{

.

.

.

}

Example

Now, consider the following function, which will take an array as an argument along with another argument and based on the passed arguments, it will return average of the numbers passed through the array as follows:

double getAverage(int arr[], int size)

{

int i;

double avg;

double sum;

for (i = 0; i < size; ++i)

{

sum += arr[i];

}

avg = sum / size;

return avg;

}

Now, let us call the above function as follows:

#include <stdio.h>

/\* function declaration \*/

double getAverage(int arr[], int size);

int main ()

{

/\* an int array with 5 elements \*/

int balance[5] = {1000, 2, 3, 17, 50};

double avg;

/\* pass pointer to the array as an argument \*/

avg = getAverage( balance, 5 ) ;

/\* output the returned value \*/

printf( "Average value is: %f ", avg );

return 0;

}

When the above code is compiled together and executed, it produces the following result:

Average value is: 214.400000