References

- Beginning C: From Novice to Professional (Beginning: from Novice to Professional)
  By Ivor Horton
- Programming in C (3rd Edition) by Stephen G. Kochan

C-Fundamentals

Like all programming language C has its own vocabulary and grammar. The sequence of instructions that is given to execute a specific task is called program. The instruction are formed of certain words, symbols and numbers which strictly follow the C grammar or syntax.

Character Set

The char are used to form words, numbers and expressions in any programming language. These depend upon individual style of each programming language. In C the characters are grouped as

a- Letters:- Consists of UPPER CASE letters A..Z and LOWER CASE a..z. It shall be noted that upper and lower case letters are not interchangeable as C is strongly case sensitive

b- Digits:- Consists of number of number from 0..9

c- Special character:- are given below

Comma ; Semicolon : Colon ? question mark
( ) Left and right parentheses + Plus - Minus
C-Identifiers

The identifier may be the name of variable or constant or function, usually for all programming languages the first character of an identifier will be an alphabet followed either by numbers or alphabet up to a maximum length which depends upon the compiler limitations. In turbo C the identifier length may be up to 31.

Note:- all C compilers are case sensitive under scores are allowed for better readability and no special characters are allowed while defining and identifier.

e.g

area (accept)
2area (not accept)
_area (not accept)
area_plus (accept)

Constant

The value of quantity does not vary throughout the program is called constant. C support several types of constant and these types refers to the basic data types of C.

String Constant

Enclosed between double quotes string constant should not be confused with character. A single character constant is enclosed between single quotes.
e.g
“welcome to C” String constant
’b’ single character

Facilities several instructions character which are not possible to enter from the keyboards
C language has created the special backslash to enter from the K.B
C has created the special backslash character constant as listed in below

\b backspace \t horizontal tab
\f fromfeed \" double quote
\n new line \’ single quote
\r carriage return \0 null
\\ backslash \w vertical tab
\o octal constant \x hexadecimal constant

Data Types
The basic data type :-
Integer
Float
Character
Void

**Type Modifier**

A modifier is used to alter the meaning of basic type appropriately suit the requirements of vinous situation.

The modifier are

Signed
Unsigned
Long
Short

**Integer (2 byte)**

There are numbers without decimal part that may or may not be prefixed with minus (-) sign.

E.g

124 -133

Integer may be long or short integer depending on the integer value.

Long (4 byte)
Short (1 byte)

**Float**

Float are numbers with decimal part. In C floats are stored with 6 digit of decimal precision.
Float       stores up to 16 decimal places (4 byte)
Double Floats  stores up to 12 decimal places (8 byte)
Long double float stores up to 24 decimal places

**Character**
Any single letter enclosed within single quotes can be defined as character type data
e.g
'a'        '*'
The range of character type is 0..255

**Void**
This is latest addition to C it is value it has two uses
One to explicitly declare a function as returning no value
Second to create generic pointers

**Variables**
A variable is the store house for constant (variable are separated by comma). All C variable must be declared before they are used.
The general form of variables is
int num, total;

**Assignment operators**
These operators are used to assign the result of an expression to a variable. The usual assignment operator ‘=’ the assignment operator again has got three types:

1- Simple Assignment
   \( A = 2; \)

2- Multiple Assignment
   \( c = a = 5 \)

3- Compound Assignment
   \( a += 5; \)

The third example ‘+=5’ means add 5 to a
\( a -= 5 \) means increment a by 5

\( a = a + 5 \)
\( a = a - 5 \)
\( a = a \% 5 \)

**Auto increment / decrement operator**

There are very useful operators generally not found in other languages. They are “++” and “--”.

The operator
“++” adds 1
While the
“--” subtract 1

**Logical Operator**

C has three logical operators as

```
&&  means logical AND
||  means Logical OR
!   means Logical Not
```

There are used when we want to test more than one condition and make decisions

e.g
a > b && x == 10

**Relational Operators**

These operators are used to compare two quantities and depending on their relation to take decisions

e.g
10 < 20  true
20 > 10  false
C language support six relational operators:-

- \(<\) less than
- \(<=\) less than or equal
- \(>\) greater than
- \(>=\) greater than
- \(==\) equal to
- \(!=\) not equal to

It is important to remember that all expression using relational and logical operators will produce a result of either 0 or 1.

**Arithmetic Operators :-**

C language is provide with all basic arithmetic operators. These can operate any built in data allowed in C. The list of arithmetic operators are:

- \(+\) Addition
- \(-\) subtraction
- \(*\) Multiplication
- \(/\) division

**Bitwise Operators**

Unlike some other languages, C language provides and supports a full complement of bitwise operators. These are used for manipulation of data at bit level. As C was designed to take the place of assembly language. It must have the ability to support many operations that are possible in assembly language and hence these operators are supported in C.

The term bitwise operation refers to the testing setting or shifting of the actual bits in a byte or word.

The following are the bitwise operators:-
Input/Output Operations

The (scanf) function is used to input a data line containing mixed mode data. Thus the scanf function is an input statement.

The scanf format is

scanf( format specifier", variable list);

Format Specifier :- specifies the field format in which the data is to be entered

Variable List :- specifies the address of location where the data is stored.

The format specifier and variable list are separated by comma

e.g

scanf(”%d%d”,&a,&b);

and suppose if we want to enter 5,7 for a and b

then, we have to enter data as follows 5 7

Different format specifier are used for different data type

1- For reading an integer.
% xd
 (x) an integer to specify field width
 (d) data type character
 e.g
 scanf("%d%d", &a, &b);
 2-For reading float number
   %xf
 f is the data type character for float.
 e.g
 if the data line is 125.73
 the input statement will be
 scanf("%f%f", &a, &b);
 3-For reading character and string
   % xc    % xs
 Her ‘c’ is for character and ‘s’ is for string
 e.g
 scanf("%c%c", &no, &no1);
 if the data line is
 r s then r assign to no and s is assign to no1
if the data input consists of string “Hello” the input statement will be

scanf(“%s”,name) ;

4- For reading mixed data type

The input statement

scanf(“%d%c%s”,&a,&b,name);

Will read

5 a welcome

*note:-when using scanf we must note

1- All variable list items must be pointer(&), except for string

2- Format specification and variable list must correctly match in the same order

3- The format specifier must be separated by spaces or commas and variable list by comma

For example consider the following scanf:

- scanf(“%d%d”,&a,&b);

When entering values for a and b the value should be separated by commas

- scanf(“%d%d”,&a,&b);

Here when entering a, b the value should be separated by space

Output Operators:-

The output format will be similar to the input one and (printf) function is used for the purpose

E.g
printf("%d\n",a,b);

is used to print two integer values

e.g

printf("%d\n",a,b);

now after displaying the values for a,b the cursor move to the second line.

Some example of printf statement are:-

-printf("%d",a) for integer number

-printf("%f",a) for float number

Also used along with printf statement certain field specification characters.

Let us them in example

1- printf("%4d",1076)

   The output 1076

2-printf("%6d",1076)

   The output 1076

3-printf("%6d",1076);

   1076—

Here the minus(-) sign is used for left justified

4-printf("%06d",1076)

   Use of (0) to pad with zeros the output is

   001076
5. printf("%xys",a)
   Here x is the integer to specify the size of output
   y is the integer to specify the number of decimal
   if a=10.77682
   printf("%7.4f",a) output 10.7768
   printf("%7.2f",a) output 10.78

Example:-

Write a program to find the area of triangle ?
   #include<stdio.h>
   main()
   { float a,b,h;
     printf("Enter b,h
");
     scanf("%f%f",b,&h);
     a=0.5*b*h;
     printf("area %f\n",a);
   }

Example

Write a program to find centigrade for a given Fahrenheit temperature?
   #include<stdio.h>
   main()
   { float c,f
     printf("Enter b,h \n");
     scanf("%f%f",&c,&f);
     c=5.0/9.0* (f-32);
printf("the temperature is \%6.2f",c);

}

Control Structure

C is considered as structured programming and one of the reasons for calling it so is that it has rich and varied program control statement.

The control statement can be branched as conditional and unconditional control statement. Under conditional control statements it has (if) and (switch). The unconditional statement is (go to). The loop construct are (while), (for) and (do-while).

The control statements are widely used to support decision making processes. They largely rely upon a conditional that determines the decision.

1-If Statement

This is one of the powerful conditional control statements. The if statements can be used in different forms depending upon the nature of the conditional test and the main forms are

1- Simple if
2- Block if
3- Nested if

1-Simple if
The general form of simple if statement is

    if ( test expression )

Ex: if ( x>y )

    printf(” x is greater than y\n”);

    If ( sales >1000 && sales<=2000 )

    Comm =sales*0.03

The test expression is evaluated first. If it returns true the statement followed by if, is executed if it returns false then the control is transferred to the next statement.

Example :-

Calculate the sales commission for the data given

below less than 1000    no commission
above 1000 but below 2000  5% of sales
above 2000 but below 5000  8% of sales
above 5000 10% of sales

#include <stdio.h>

main()
{  
  float sales, com;
  
  printf("enter sales value\n");
  
  scanf(" %f", & sales);
  
  if (sales<=1000)
    com=0;
  
  if (sales>1000 && sales <=2000)
    com=sales*0.05;
  
  if (sales>2000 && sales<=5000)
    com=sales*0.08;
  
  if (sales>5000)
    com=sales*0.10;
  
  printf("\ncommission %.2f", com);
  
}  

2. Block if statement

Here a block or group of statement follow the test expression

The general form
if ( test expression )
{
  statements ;
}

• Simple if-else statement

Here an optional clause ‘else’ is used along with the simple if statement the general form is

if ( test expression )

  state 1;

else
state2;

e. g :-

if ( total>250 )

printf (” result is pass\n”);

else

printf (” result if fail\n”);

• **Block if –else statement**

  This is similar to the form as in (block if statement) except a block or group of statements follows the test expression .

  The general form

  ```
  if (test expression)
  { state1;state2;
  }
  else
  { state3
  }
  ```

  e.g

  ```
  if(sales> 3000)
  {
  com=sales*0.08;
  ```
3-Nested if

This is the most important aspect of (if) statement in the scene. It may lead to confusion if not properly used. The (nested if) is a statement that is the object of either if or else.

The general form

If(test expression)
{
    State1;
}
else
    if (test expression)
    {
        State2
    }
else
    {
        State3
    }
Example

Write a program to determine if any number is even or odd?

```c
#include<stdio.h>

main()
{
    int n;
    scanf("%d",n)
    if n mod 2==0
        printf("The number is even");
    else
        printf("The number is odd");
}
```

**Switch Statement**

This is a multiple branching control statement and this one is very useful for decision making when more than one case is involved.

The general form of switch statement

```c
switch(exp)
{
    case label 1:   {state1; break;}
    case label 2:   {state2; break ;}
    case label 3:  { state3; break;}
```
default : state4;

Switch statement differs from an (if) statement in that ‘switch’ can only test for equality where as ‘if’ can evaluate for entire given range. When the ‘switch’ is executed the value of expression compared against the label1,label2,label3…etc. If case is found whose value matches with the expression ,then the sequence of statements that follows the case are executed.

The ‘break’ statement at the end of the each block indicates the end of the particular case and causes and exit from ’switch’ statement transferring the control outside switch.

The ‘default’ statement is executed if no matches are found and is optional.

**Goto statement**

This is an unconditional control statement. The goto statement may be used to exit from several layers of nesting. The goto requires a label for operation and the label must be in the same function as the goto that uses.

The general form:

goto label

The use goto is highly discouraged since more “goto” means more confusion.

Looping
C has a powerful looping constructs in it, and this is one form of control statement where the number of repetitions is already known to us. In C there are three looping constructs. They are `for`, `while`, and `do while`.

1-for

The general form of a for loop is

```c
for ( exp1 ; exp2; exp3 )
{
    st1;
    st2;
    ...
    ...
    ...
}
```

- `exp1` is the initial value
- `exp2` is the testing value
- `exp3` is the increment step value

For example:
sum=0;

for ( I = 0; i<=100; i++)

sum+=i;

note that there is no semicolon after for loop ; if there is a semicolon after the for loop then the loop is said to be a time delay loop or “software delay”

*By using the comma operator more than one variable can be initialized as seen below

for ( fact=1; i=1; i<10; i++)

fact*=1;

Without the comma operator the same will be seen as

fact=1;

for ( i=1; i<10; ++i)

fact*=1;

* we can write the above example in another way

fact=i=1;

for ( ; i<=10; i++)

fact*=i;
*the last form of for construct in which the initial value, final value and also increment value is not given

for ( ; ; )
{
state
}
The above for loop executes infinitely

2- while

“while” is an entry controlled loop statement i.e. the test condition is evaluated and if it is true the body of the loop is evaluated. This process is repeated until the test condition becomes false; then the control is transferred out of loop.

The general form of “while” is

while ( test expression )
{
sequence of statements
}
The following while loop helps us to sum the numbers between 1 to 100

sum=0;
n=1;

while ( i<=100) 
{
    sum+=i;
    i++
}

*if we want the loop to executed infinitely . The general form this
while (1)
{
    statement;
}

3-looping using do-while

It may be necessary to execute the body of the loop, before the test is performed . Such state can be handled using 'do-while'

The general form is

do {
sequence of statement ;

}

while (test expression )

Example:-

Write a program to find the sum of the odd numbers between 1 and 100?

#include <stdio.h>

main()

{

int i,sum;

sum=0;

i=1;

do {

sum+=i;

i+=2;
}

while (i<=100 );
Break statement

An early exit from the loop is possible by a ‘break’ statement. When break statement is encountered the loop is immediately exited and the program continues with the statement immediately following the loop.

The general form is

     while (test expression )

     {

        if something break :

        }

*Example

Write a program to sum of all positive number and terminate when reading a negative number?

main ()

{  

int n, s=0;

printf ("/n Enter a number and negative number to exit /n");

while (1)
{
    scanf ("%d", n);
    if (n<0)
        break;
    else
        s+=n;
}

printf ("\n the sum is %d ", s);

5-Continue statement

This statement causes the loop to be continued while the next iteration after skipping the statement in between thus by bassing the rest of the loop

The general form is
for ( expression 1; expression 2; expression 3 )
{
    if something continue;
}

Example

Write a program to find the sum of the positive number among ten numbers?

#include <stdio.h>

main()
{
    int i,no,sum=0;
    printf ( "\n enter 10 number \n" );
    for ( i=0; i<10; i++ )
    {
        scanf ( "%d", &no );
        if ( no<0 ) continue;
        else
            sum+=no;
printf ( "/n sum of positive number %d \n " ,sum )

Exit ( ) Function

The exit function causes immediate termination of the entire program. This function stops the program execution and causes a force return the operating system. Because of this property this is used only for specific purposes and not commonly used in (C) program. When this is used a program will look like.

Looping

C has a powerful looping constructs in it and this is one form of control statement where number of repetitions is already known to us. In c there are three loping constructs. They are for, while, do while.

1-for

The general form of a for loop is

for (exp1 ; exp2; exp3 )
{
}
C-LANGUAGE

1st Class

exp1 is the initial value
exp2 is the testing value
exp3 is the increment step value
e. g

sum=0;

for ( I = 0;i<=100; i++)

sum+=i;

note that there is no semicolon after for loop ; if there is a semicolon after the for loop then the loop is said to be a time delay loop or “software delay”

By using the comma operator more than one variable can be initialized as seen below

for ( fact=1; i=1; i<10; i++)
fact*=1;

Without the comma operator the same will be seen as

fact=1;

for ( i=1; i<10; ++i)

fact*=1;

* we can write the above example in another way

fact=i=1;

for ( ; i<=10; i++)

fact*=1;

*the last form of for construct in which the initial-value , final value and also increment value is not given

for ( ; ; )

{
    state
}

The above for loop executes infinitely

2- while
“while” is an entry controlled loop statement i.e. the test condition is evaluated and if it is true the body of the loop is evaluated. This process is repeated until the test condition becomes false; then the control is transferred out of loop.

The general form of ‘while’ is

while ( test expression )
{
  sequence of statements;
}

The following while loop helps us to sum the numbers between 1 to 100

sum=0;
n=1;
while ( i<=100)
{
  sum+=i;
i++
}

*if we want the loop to executed infinitely. The general form this...
while (1) {
  statement;
}

3-looping using do-while

It may be necessary to execute the body of the loop, before the test is performed. Such state can be handled using ‘do-while’

The general form is

```
do {
  sequence of statement;
} while (test expression);
```

Example:

Write a program to find the sum of the odd numbers between 1 and 100?

```c
#include <stdio.h>

main()
```
{  
int i , sum;  
sum = 0;  
i = 1; 
do {  
sum += i;  
i += 2;  
}  
while ( i <= 100 );  

Break statement  

An early exit from the loop is possible by a ‘break’ statement. When break statement is encountered the loop is immediately exited and the program continues with the statement immediately following the loop.

The general form is:  

while (test expression )  
{  
    if something break ;  
}
Example

Write a program to sum of all positive number and terminate when reading a negative number?

```c
main ()
{
    int n,s=0;
    printf ("/n Enter a number and negative number to exit /n");
    while (1)
    {
        scanf ("%d",n);
        if ( n<0 )
            break ;
        else
            s+=n;
    }
    printf ("/n the sum is %d ",s);
}
```
5-Continue statement

This statement causes the loop to be continued while the next iteration after skipping the statement in between thus by bassing the rest of the loop

The general form is

    for ( expression 1; expression 2; expression 3 )
    {
        if something continue ;
    }

Example

Write a program to find the sum of the positive number among ten numbers?

    #include <stdio.h>

    main()
    {
        int i,no,sum=0;
        printf ( "\n enter 10 number \n" );
        for ( i=0; i<10; i++ )
{  
    scanf ( "%d",&no );  
    if ( no<0 ) continue ;  
    else  
        sum+=no;  
}

printf ( "\n sum of positive number %d \n ",sum )

Exit ( ) Function

The exit function causes immediate termination of the entire program. This function stops the program execution and causes a force return the operating system. Because of this property this is used only for specific purposes and not commonly used in (C) program. When this is used a program will look like

Arrays
Normally variable can store only value at time and it is possible to store a new value only after erasing the previous one. But in real life situations while computing we would like to store more than one value at a time and this could be achieved through Arrays.

An array is a collection of variable of same data type that are referenced by a common name.

Example :

```c
int a[10]
```

On encountering the above statement, the system will allocate 10 memory locations for the identifier (a) and the orientation may be as follows

Here the index starts from (0) and goes up to (9).

1.1 one dimensional array :

This is also called a “list” this means that an item is an array can be accessed by just giving one subscript. The general form of a single-dimension array is “type specifier identifier-name[size]”

Ex:

```c
int a[3]
```
we are declaring an integer array having 5 elements $a[0]...a[4]$ as explained below

Example :- write a program to read five elements and print them using array

Solution

```c
#include<stdio.h>
main()
{
    int no[5],i ;
    printf (“Enter 5 numbers one by one \n”);
    for (i=0; i<5; i++)
        scanf ( ”%d”,& no[i] );
    for ( i=0; i<5; i++)
        printf ( ”%d”, no);
}
```

Example2: Write a program to find the maximum number in an array ?

Solution

```c
#include<stdio.h>
main()
{
    int a[5],i, max;
```
for ( i=0; i<5; i++ )
scanf ( "%d", & a[i] );
max=a[0];
for ( i=1; i<5; i++ )
if ( max<a[i] )
    printf ( "\n the maximum no in the array %d", max );
}

Example3:- Write a program to sort the element of an array.

Solution
#include<stdio.h>
main()
{
    int a[5],i , j , t ;
    printf ( "Enter 5 number one by one
" )
    for ( i=0; i<5; i++ )
scanf ( "%d", & a[i] );
    for ( i=0; i<4; i++ )
        for ( j=i+1; j<5; j++ )
            if ( a[i]>a[j] )
                {
                    t=a[i] ;
                    a[i]=a[j] ;
                    a[j]=t ;
                }
1.2 Two Dimensional Array:

A two-dimensional array in fact is any array of one dimensional arrays. A single-dimensional array can store a list of values, whereas a two-dimensional array can store a table of values.

Example:

```c
int a[3][3] will have element

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

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

a[2][0]  a[2][1]  a[2][2]
```

for reading a two-dimensional array

```c
scanf("%d", &a[i][j]);
```

User Defined Function

A function in C Language is considered as a fundamental building block of the language. In order to avoid the complexity of a program while coding, debugging, and testing,
the program is divided into functional parts or subprograms. Thus each module is considered as a function and functions do everything in C.

The general form of C functions is

Argument declaration

{  
    Body of the function;
    Return(expression);
}

Example

Program to illustrate the uses of returns()

#include <stdio.h>

main()
{
    int a,b;
    printf(" Enter 2 number \n");
    scanf("%d%d",&a,&b);
    small(a,b);
}
small(a,b)

int x,y;
{
    if (x < y)
        printf("a is less than b");
    Else
        printf("b is less than a");
    return;
}

**Calling Function**

Normally we can call function in two ways:

a- Call by value
b- Call by reference

The former and later are also called value parameter and variable respectively.

We can invoke the function in two ways. First we can place it as the part of an expression.

i)  area=aof1(b,h)

ii) if (strequal(st1,st2) =0)
    printf("\n given string are equal");
    else
printf(“\n given strings are unequal”);

*call by value

Example

#include <stdio.h>

main()
{
    int   a,b;
    printf(” Enter value for a & b\n”);
    scanf(”%d%d”,&a,&b);
    fun(a,b);
    printf(“”the value of a,b after execution”);  
    printf(“”the function is %d%d\n”,a,b);
    fun(d,e)
    int d,e;
    {        
        d=d*5;
        e=e*7;
Structure

A “structure” is a collection of data item (fields) or variable of different data type that is referenced under the same name. It provides convenient of means of keeping related information together.

The declaration of structure as follows

```c
Struct tag-name {
    Data type members
};
```

Example: struct of student contain name, examname, and mark on 3 subjects (math, physics, chemistry)
Struct student
{
    char name[25];
    char examname[10];
    int math, phys, chem;
};

It is permitted in (C) to combine both template declaration and variable declaration in a single statement as seen below

Struct student
{
    char name[25];
    char examname[10];
    int math, physic, chem;
}
St1, st2, st3 or can be written
Struct student st1, st2, st3;

*The use of (tag-name) is optional

Struct
{
    Data type member;
}

st1, st2, st3;

To give a value to structure field we use the following statement

st1.name="Ali";

st1.examname="English";

st1.math=75;

File

To deal with file we must do

1. Opening file
2. Reading data from file
3. Writing data to file
4. Closing file
*fopen( ) { open file)

    fp=fopen ("filename"."mode");

A mode can be of any one of the following:

    r      open the file for reading
    w      open the file for writing
    a      open the file for appending data to it

fclose     ( close the file )

example

file      *p1,*p2;

p1=fopen ( "abc.data","w");

p2=fopen ( "def.data","r");

fclose (p1);

fclose (p2);
Input/Output operation on file

A-putc () & gets ()

Putc : used to write character to a disk file that was previously opened for writing.

Putc ( ch,fp ) ;

getc ( ) : is use to read character from a file

char ch;

ch = getc ( fp )

Ex1 :-

main ( )
{

File *fp1;

Char c ;

fp1 = fopen ( “text.dat” , “w” ) ;

while ( c=getchar ( )!=’\n’ )

    putc ( c,fp1 );

fclose (fp1);
}


Ex 2 :

```c
main ()
{
    file *fp1;
    char c
    fp1=fopen ("text.dat", "r");
    while ( c=getc(fp1)!='\n' )
        putchar (c);
    fclose (fp1);
}
```

B- getw and putw

They are similar to getc( ) and putc( ) and they are used to read and write integer values.

```c
putw (integer , fp );
getw (fp );
```

Ex3 :

```c
main ()
```
{ 
    file *fp1; 

    fp1=fopen("number.dat", "w"); 

    scanf("%d", &no); 

    while(no!=0) 
    { 
        putw(no,fp1); 

        scanf("%d", &no); 
    } 

    no=0; 

    putw(no,fp1); 

    fclose(fp1); 

Ex 4: - 

    main() 
    
    int no;
### C-language

```c
file *fp1 ;

fp1=fopen ("number.dat", "r") ;

while ( no=getw(fp1)!=0 )
    printf ( "%d\n" , no ) ;

fclose ( fp1 ) ;

}
```

#### C- fprint ( ) & fscan ( )

The function is similar to printf , scanf except they operate with file .

```c
fprint ( fp, "string", argument list )

fscan ( fp, "string", argument list )
```