**INDEX**

1. **INTRODUCTION TO C LANGUAGE**
   1. PREPROCESSOR DIRECTIVES
   2. HEADAER FILES
   3. DATA TYPES
   4. QUALIFIERS
   5. OPERATORS
   6. EXPRESSIONS
   7. PRECEDENCE AND ASSOSIATIVITY
   8. DATA INPUT AND OUTPUT
   9. CONTROL STATEMENT
      1. DECISION CONTROL STATEMENTS
      2. LOOP CONTROL STATEMENTS
      3. CASE CONTROL STATEMENTS
2. **DERIVED AND USER DEFINED DATATYPES**
   1. ARRAY
   2. STRING
   3. 2-DIMENSIONAL ARRAY AND OPERATIONS
      1. MATRIX ADDITION AND SUBSTRACTION
      2. MATRIX MULTIPLICATION
      3. TRANSPOSE OF MATRIX
   4. STRUCTURE
   5. UNION
   6. ENUMERATED DATATYPE
3. **POINTERS**
   1. ARRAY OF POINTERS
   2. STRUCTURE AND POINTERS
4. **FUNCTIONS**
   1. FUNCTION DEFINITION AND PROTOTYPING
   2. FUNCTION CALL BY VALUE
   3. FUNCTION CALL BY REFERENECE
   4. POINTER TO A FUNCTION
   5. RECURSIVE FUNCTION
5. **SORTING, SEARCHING, AND** 
   1. BUBBLE SORT
   2. SELECTION SORT
   3. LINEAR SEARCH
   4. BINARY SEARCH
   5. SCOPE RULES
   6. STORAGE CLASSES
   7. MEMORY ALLOCATION
   8. BIT-WISE OPERATIONS
6. **DATA FILES**
   1. FORMATTED TEXT FILE OPERATIONS
   2. UNFORMATTED TEXT FILE OPERATIONS
   3. COMMAND LINE ARGUMENTS

**MODULE 1: INTRODUCTION TO C LANGUAGE**

**a . PREPROCESSOR DIRECTIVES :** When a C program is compiled first source code is processed by a program called preprocessor. This process is called preprocessing. Commands used in preprocessor are called preprocessor directives and they begin with “#” symbol.

|  |  |
| --- | --- |
| **Preprocessor** | **Syntax/Description** |
| Header file inclusion | **Syntax:**#include <file\_name> The source code of the file “file\_name” is included in the main program at the specified place. |
| Macro | **Syntax:**#defineThis macro defines constant value and can be any of the basic data types. |
| Conditional compilation | **Syntax:**#ifdef, #endif, #if, #else, #ifndefSet of commands are included or excluded in source program before compilation with respect to the condition. |
| Other directives | **Syntax:**#undef, #pragma #undef is used to undefine a defined macro variable. #Pragma is used to call a function before and after main function in a C program. |

**b . HEADAER FILES**

|  |  |
| --- | --- |
| **HEADER FILES** | **SOME LIBRARY FUNCTIONS** |
| stdio.h | Printf()  Scanf() |
| stdlib.h | System(clear) |
| math.h | Pow()  Sqrt()  Sin()  Cos() |
| string.h | strcat  strcmp  strlen |

C TOKENS: C tokens are the basic buildings blocks in C language which are constructed together to write a C program. Each and every smallest individual units in a C program are known as C tokens. C tokens are of six types. They are,

Keywords (eg: int, while),

Identifiers (eg: main, total),

Constants (eg: 10, 20),

Strings (eg: “total”, “hello”),

Special symbols (eg: (), {}),

Operators (eg: +, /,-,\*)

**c . DATA TYPES**

|  |  |
| --- | --- |
| **Types** | **Data Types (size)** |
| Basic data types | Int(2) , char(1), float(4), double(8) |
| Enumeration data type | enum |
| Derived data type | pointer, array, structure, union |
| Void data type | void |

**d. QUALIFIERS**

**TYPE QUALIFIERS:**

There are two types of qualifiers these are used to modify properties of variable. They are,

**1.const**

Syntax: const data\_type variable\_name; (or) const data\_type \*variable\_name;

**2.volatile**

Syntax: volatile data\_type variable\_name; (or) volatile data\_type \*variable\_name;

**SIZE QUALIFIERS**:

1.short

2.long

3.long long

**SIGN QUALOFIERS**

1.signed

2.unsigned

e . **OPERATORS**

C language offers many types of operators. They are,

Arithmetic operators

Assignment operators

Relational operators

Logical operators

Bit wise operators

Conditional operators (ternary operators)

Increment/decrement operators

Special operators

|  |  |
| --- | --- |
| **OPERATORS** | **EXAMPLES** |
| Arithmetic operators | + , - , \* , / |
| Assignment operators  Relational operators | =  == , != , <= , >= |
| Logical operators | && , || |
| Bit wise operators | & , | , ^ |
| Conditional operators (ternary operators) | ? : ; |
| Increment/decrement operators | ++ , -- |
| Special operators | Sizeof() ,\* , & |

EXPRESSIONS

**PRECEDENCE AND ASSOSIATIVITY**

|  |  |  |  |
| --- | --- | --- | --- |
| CATEGORY | OPERATOR | PRECEDENCE | ASSOCIATIVITY |
| Most precedence group | ()  []  ->  ::  . | 1 | L –> R |
| UNARY | !  -  +  ++  --  &  \*  sizeof | 2 | R -> L |
| MEMBER ACCESS | .\*  ->\* | 3 | L –> R |
| MULTIPLICATION | \*  /  % | 4 | L –> R |
| ADDITIVE | +  - | 5 | L –> R |
| SHIFT | <<  >> | 6 | L –> R |
| RELATIONAL | <  <=  >  >= | 7 | L –> R |
| EQUALITY | ==  != | 8 | L –> R |
| BITWISE | &  ^  |  &&  || | 9 | L –> R |
| CONDITIONAL | ?: | 10 | R -> L |
| ASSIGNMENT | =  \*=  /=  %=  +=  -=  &=  ^=  |=  <<=  >>= | 11 | R -> L |
| COMMA | , | 12 | L –> R |

**DATA INPUT AND OUTPUT**

1. getchar() and putchar()
   1. example

#include <stdio.h>

int main( ) {

char c;

printf( "Enter a value :");

c = getchar( );

printf( "\nYou entered: ");

putchar( c );

return 0;

}

1. gets() and puts()
   1. example

#include <stdio.h>

int main( ) {

char str[100];

printf( "Enter a value :");

gets( str );

printf( "\nYou entered: ");

puts( str );

return 0;

}

1. scanf() and printf()
   1. example

#include <stdio.h>

int main( ) {

char str[100];

int i;

printf( "Enter a value :");

scanf("%s %d", str, &i);

printf( "\nYou entered: %s %d ", str, i);

return 0;

}

**CONTROL STATEMENT**

1. **DECISION CONTROL STATEMENTS**

|  |  |
| --- | --- |
| **Decision control statements** | **Syntax/Description** |
| If | Syntax:  if (<condition>)  { <Statements>; }  Description:  In these type of statements, if condition is true, then respective block of code is executed. |
| if…else | Syntax:  if (<condition>)  { <Statement1>;< Statement2>; }  else  { <Statement3>; <Statement4>; }  Description:  In these type of statements, group of statements are executed when condition is true. If condition is false, then else part statements are executed. |
| nested if | Syntax:  if (<condition1>){ <Statement1>; }  else\_if(<condition2>)  { <Statement2>; }  else <Statement 3>;  Description:  If condition 1 is false, then condition 2 is checked and statements are executed if it is true. If condition 2 also gets failure, then else part is executed. |

1. **LOOP CONTROL STATEMENTS**

|  |  |
| --- | --- |
| Loop Name | Syntax |
| For | for (<exp1>;< exp2>;<expr3>)  { <statements>; } |
| While | while (<condition>)  { <statements>; } |
| do while | do { <statements>; }  while (<condition>); |

**DIFFERENCE BETWEEN WHILE & DO WHILE**

|  |  |
| --- | --- |
| While | do while |
| Loop is executed only when condition is true. | Loop is executed for first time irrespective of the condition. After executing while loop for first time, then condition is checked. |

1. **Case control statements**
   1. Switch
      1. SYNTAX:

switch (<expression>)

{

case <label1>: <statements>;

break;

case <label2>: <statements>;

break;

case <label3>: <statements>;

break;

default: <statements>;

break;

}

* 1. **Break**
     1. EXAMPLE

for(i=0;i<10;i++)

{

if(i==5)

{ break; }

printf("%d ",i);

}

o/p: 0 1 2 3 4

* 1. **Continue**
     1. EXAMPLE

for(i=0;i<10;i++)

{

if(i==5)

{ continue; }

printf("%d ",i);

}

o/p: 0 1 2 3 4 6 7 8 9

* 1. **Goto**
     1. SYNTAX

{

…….

go to label;

…….

…….

LABEL:

statements;

}

* + 1. **EXAMPLE**

int main()

{

int i;

for(i=0;i<10;i++)

{

if(i==5)

{

printf("\nWe are using goto statement when i = 5");

goto HAI;

}

printf("%d ",i);

}

HAI : printf("\nNow, we are inside label name \"hai\" \n");

}

o/p :

0 1 2 3 4

We are using goto statement when i = 5

Now, we are inside label name “hai”

**MODULE 2: DERIVED AND USER DEFINED DATATYPES**

**ARRAY**

Arrays is a sequential collection of elements of the same data type, which have a fixed-size. An array is used to store a collection of data. Arrays consist of continuous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Array [0]  1000 | Array [1]  1001 | Array [2]  1002 | Array [3]  1003 | Array [4]  1003 | Array [5]  1004 |

**First element** **Last element**

**DECLARING ARRAY**

For declaration of the program, we have to note the type of element and the number of element s or the of size of the array

<DATATYPE> <ARRAY NAME>[<ARRAY SIZE>];

Example

int a[10];

here int means the datatype ,a is the array name and 10 is the array size.it can hold 10 integer elements indexing from 0 to 9

**NOTE:**

if you try to access the a[10] element ,it will show as a error.

**INITIALIZING AN ARRAY**

Here we have two ways for initialising an array

**TYPE 1 : MULTIPLE INTIALISING IN A SINGLE STATEMENT**

<ARRAY NAME> [ ]= {value1, value2,value3,….}

Here we have to declare the elements in the sequence order for initialising the element in indexed positions.

Example

a[]={1,2,3,4,5,6,7,8,9,10};

**TYPE 2 : SINGLE INTIALISING**

<ARRAY NAME> [ <ELEMENT POSITION> ] = < VALUE >;

Here we have to specify the array index value and the value

Example

a[5]=6;

**STRING**

For string we use character array it is ended with '\0'

**CHARACTER ARRAY INTIALISATION & DECLARATION**

We can intailise in two ways are explained below:

Char <name> [<size>] = “<string>”;

Char <name> [<size>] ={‘<character1>’, ‘<character2>’, ‘<character3>’,…..,’\0’};

Example

char name [4]="life";

char name [4]={ 'l','i','f','e','\0' }

STRING HEADERFILE AND FUNCTIONS

|  |  |
| --- | --- |
| String functions | Description |
| strcat ( ) | Concatenates str2 at the end of str1 |
| strncat ( ) | Appends a portion of string to another |
| strcpy ( ) | Copies str2 into str1 |
| strncpy ( ) | Copies given number of characters of one string to another |
| strlen ( ) | Gives the length of str1 |
| strcmp ( ) | Returns 0 if str1 is same as str2. Returns <0 if strl < str2. Returns >0 if str1 > str2 |
| strstr ( ) | Returns pointer to first occurrence of str2 in str1 |
| strrstr ( ) | Returns pointer to last occurrence of str2 in str1 |
| strrev ( ) | Reverses the given string |

**MULTI DIMENSIONAL ARRAY**

In c we can create array with multiple dimensional means array of arrays.

<Datatype> <name>[size1][size2]...[sizeN];

**2-DIMENSIONAL ARRAY AND OPERATIONS**

Two dimensional array are array of another array

DECLARATION OF 2 DIMENSIONAL ARRAY

<Datatype> <name>[size1][size2];

COLUMN 0 COLUMN2 COLUMN3

|  |  |  |
| --- | --- | --- |
| A[0][0] | A[0][1] | A[0][2] |
| A[1][0] | A[1][1] | A[1][2] |

**DECLARING & INITIALISING 2 DIMENSIONAL ARRAY**

<Datatype> <name>[size1][size2] = { A[0][0],A[0][1], A[0][2], A[1][0], A[1][1] ,A[1][2] };

<Datatype> <name>[size1][size2] = {

{ A[0][0],A[0][1], A[0][2]},

{ A[1][0], A[1][1] ,A[1][2] }

};

Example

Int a[3,3]={0,1,2,3,4,5,6,7,8,9}

Int a[3,3]={

{0,1,2},{3,4,5},{6,7,8}

};

**ACESSING ELEMENT OF ARRAY**

For every element can be initialise by considering the rows as “I” and column as “j”. Generally, an element is founded as in format a[i][j] .

**MATRIX ADDITION AND SUBSTRACTION**

Array Addition

#include<stdio.h>

void main()

{

int n,m,i,j;

printf(“enter the number of rows and columns\t”);

scanf(“%d%d”,&n,&m);

int a[n][m],b[n][m],c[n][m];

printf (“enter the elements of first matrix”);

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

{

for(j=0;j<m;j++)

{

scanf(“%d”,&a[i][j]);

}

}

//printing first array

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

{

for(j=0;j<m;j++)

{

printf(“%d\t”,a[i][j]);

}

printf(“\n”);

}

printf(“enter the second array elements”);

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

{

for(j=0;j<m;j++)

{

scanf(“%d”,&b[i][j]);

}

}

//printing second array

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

{

for(j=0;j<m;j++)

{

printf(“%d\t”,b[i][j]);

}

printf(“\n”);

}

//addition

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

{

for(j=0;j<m;j++)

{

c[i][j]=a[i][j]+b[i][j];

}

}

//printing resultant array

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

{

for(j=0;j<m;j++)

{

printf(“%d\t”,c[i][j]);

}

printf(“\n”);

}

}

Array Substaction

#include<stdio.h>

void main()

{

int n,m,i,j;

printf(“enter the number of rows and columns\t”);

scanf(“%d%d”,&n,&m);

int a[n][m],b[n][m],c[n][m];

printf (“enter the elements of first matrix”);

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

{

for(j=0;j<m;j++)

{

scanf(“%d”,&a[i][j]);

}

}

//printing first array

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

{

for(j=0;j<m;j++)

{

printf(“%d\t”,a[i][j]);

}

printf(“\n”);

}

printf(“enter the second array elements”);

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

{

for(j=0;j<m;j++)

{

scanf(“%d”,&b[i][j]);

}

}

//printing second array

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

{

for(j=0;j<m;j++)

{

printf(“%d\t”,b[i][j]);

}

printf(“\n”);

}

//addition

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

{

for(j=0;j<m;j++)

{

c[i][j]=a[i][j]-b[i][j];

}

}

//printing resultant array

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

io{

for(j=0;j<m;j++)

{

printf(“%d\t”,c[i][j]);

}

printf(“\n”);

}

}

**MATRIX MULTIPLICATION**

#include<stdio.h>

void main()

{

int n,m,p,q;

for(“enter the number of rows and columns of 1st array”\n);

scanf(“%d%d”,&n,&m);

for(“enter the number of rows and columns of 2nd array”\n);

scanf(“%d%d”,&n,&m);

if (n!=p)

{

printf(“these matrixes can’t be multiplied”);

exit(0);

}

else

{

int a[n][m],b[p][q],c[m][q]

printf(“enter the elements of the first array”);

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

{

for(j=0;j<m;j++)

{

scanf(“%d”,&a[i][j]);

}

}

//printing the first array

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

{

for(j=0;j<m;j++)

{

printf(“%d\t”,&a[i][j]);

}

printf(“\n”);

}

printf(“enter the elements of the second array”);

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

{

for(j=0;j<q;j++)

{

scanf(“%d”,&b[i][j]);

}

}

//printing the second array

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

{

for(j=0;j<q;j++)

{

printf(“%d\t”,&b[i][j]);

} printf(“\n”);

}

//metrix multiplication

for(i=o;i<m;j++)

{

for(j=0;j<q;j++)

{

c[i][j]=0;

for(k=0;k<m;k++)

{

c[i][j]= c[i][j]+(a[i][k]\*b[k][j]);

}

}

}

//printing the multiplied array

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

{

for(j=0;j<q;j++)

{

printf(“%d\t”,&c[i][j]);

}

printf(“\n”);

}

}

**TRANSPOSE OF MATRIX**

#include<stdio.h>

void main()

{

int n,m;

printf(“enter the number of rows and columns\n”);

scanf(“%d%d”,&n,&m);

printf(“enter the elements of the array\n”);

int a[n][m];

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

{

for(j=0;j<m;j++)

{

scanf(“%d”,&a[i][j]);

}

}

//printing the array

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

{

for(j=0;j<m;j++)

{

printf(“%d\t”,&a[i][j]);

}

printf(“\n”);

}

//transposing array

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

{

for(j=0;j<m;j++)

{

b[i][j]=a[j][i];

}

}

//printing transpose array

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

{

for(j=0;j<m;j++)

{

printf(“%d\t”,&b[i][j]);

}

printf(“\n”);

}

}

**STRUCTURE**

It is a derived data type which is a collection of different data types

**DEFINING A STRUCTURE**  
struct <name>  
{  
<data type> <variable1>;  
<data type >< variable 2>;  
<data type >< variable 3>;  
};

Example:  
struct student  
{  
int  mark;  
char name[10];  
float average;  
}S1;

**INTILISATION OF STRUCTURE**

struct <structure name> <variable name>;

Example : struct student S1;

**ACCESSING STRUCTURE ELEMENTS**

For accessing any member of structure we use dot operator

<structure variable>.<element name>

Example:

A = S1.mark;

**UNION**

**DEFINING A UNION**

It is a derived data type which is a collection of different data types with one storage area

union <name>  
{  
<data type> <variable1>;  
<data type >< variable 2>;  
<data type >< variable 3>;  
};

Example:

union du

{

int x;

float y;

char str[30];

} a;

**INTILISATION OF UNION**

union <union name> <variable name>;

**ACCESSING UNION ELEMENTS**

For accessing any member of union we use dot operator

Union <union variable>.<element name>

Example:

z= a.y;

**ENUMERATED DATATYPE**

The keyword used is enum and it is user define data type

NOTE: If the values are of given the variable the default value of first variable will be zero and other variable having incremented value.

**DEFINING A ENUM**

enum <name>  
{  
 <variable1>=<value1>;  
< variable 2>=<value2>;  
< variable 3>=<value3>;  
};

Example:

enum deck {

club = 0,

diamonds = 10,

hearts = 20,

spades = 3,

}card;

MODULE 3 : POINTER AND ARRAYS

a. ARRAY OF POINTERS

* Consider we have an array named ary and address of ary is 1000.
* the address of next element in the array will be 1002 if it is an integer array (size of int is 2 ).
* hence ary contains the address of ary[0] so ary = &ary[0]
* we can access every elements using p++

example: int i;

int a[3] = {1, 2, 3};

int \*p = a; // == ( int\*p = &a[0] )

for (i=0; i<5; i++)

{

printf("%d", \*p);

p++;

}

b. STRUCTURE AND POINTERS

To access members of structure with structure variable, we used the dot . operator. But when we have a pointer of structure type, we use arrow -> to access structure members.

Example :

struct student

{

int id;

char name[30];

float percentage;

};

Void main() {

int i;

struct student record1 = {1, "Raju", 90.5};

struct student \*ptr;

ptr = &record1;

printf("Records of STUDENT1: \n");

printf(" Id is: %d \n", ptr->id); //prints 1

printf(" Name is: %s \n", ptr->name); //prints ”Raju”

printf(" Percentage is: %f \n\n", ptr->percentage); //prints 90.5

}

**MODULE 4 : FUNCTIONS**

* FUNCTION DEFINITION AND PROTOTYPING
* FUNCTION CALL BY VALUE
* FUNCTION CALL BY REFERENECE
* POINTER TO A FUNCTION

It is used to invoke a function using pointer .Similar to a variable is pointed by a pointer, a function also can be pointed by a pointer .A pointer to a function is declared as follows:

<datatype> (<\*function\_pointer>) ();

This function pointer can point to any function

Eg: int sum(int,int);

int (\*p1) ();

p1=sum;

Here **p1** is the function pointer and **sum** is the function . We point **p1** to the function **sum .**To call the function **sum** ,we can use the pointer **p1** with parameters.

(\*p1)(x,y);

* RECURSIVE FUNCTION

A function that calls itself is known as a recursive function. And, this is known as recursion .Recursion makes program elegant and cleaner. All algorithms can be defined recursively which makes it easier to visualize and prove.

Eg: **Factorial using recursion**

factorial(int n)

{

int fact;

if(n=1)

return(1);

else

fact=n\*factorial(n-1);

return (fact);

}

**Fibanocci using recursion**

#include<stdio.h>

void printFibonacci(int);

int main()

{

     int k,n;

     long int i=0,j=1,f;

printf("Enter the range of the Fibonacci series: ");

    scanf("%d",&n);

    printf("Fibonacci Series: ");

    printf("%d %d ",0,1);

    printFibonacci(n);

     return 0;

}

void printFibonacci(int n)

{

      long int first=0,second=1,sum;

{

     if(n>0)

         sum = first + second;

         first = second;

         second = sum;

         printf("%ld ",sum);

         printFibonacci(n-1);

    }

}

**MODULE 5: SORTING, SEARCHING**

1. **BUBBLE SORT**
   1. Summary : In bubble sort sorting is done by repeatedly swapping adjacent elements if they are in wrong order
2. **SELECTION SORT**
   1. The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.
3. **LINEAR SEARCH**
   1. If we wan to search an element in an array linear search is the simple approach.it compares the given element with all elements in array and finds the element.
4. **BINARY SEARCH**
   1. Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.
5. **SCOPE RULES**
   1. In C programming variable can be declared in two places
      1. Inside a function or block : LOCAL
      2. Outside all function and block : GLOBAL

**Local variables:**

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own

**Global variables:**

Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.A global variable can be accessed by any function.+

Example:

#include <stdio.h>

int a = 20; // global variable declaration

int main () {

int sum,b = 20; // local variable declaration in main function

sum=a+b;

}

1. **STORAGE CLASSES**

A storage class defines the scope and lifetime of a variable or function.There are four different storage classes :

* + - 1. **Auto**

The auto storage class is the default storage class for all local variables.

Example: auto int length;

* + - 1. **Register**

The register storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

Example : register int length;

* + - 1. **Static**

The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

Example : static int length = 5;

* + - 1. **extern**

The scope of this extern variable is throughout the main program. It is equivalent to global variable. Definition for extern variable might be anywhere in the C program.

Example :

#include<stdio.h>

int main( )

{

extern int y;

printf("The value of y is %d",y);

}

int y=50;

1. **MEMORY ALLOCATION**
2. **BIT-WISE OPERATIONS**

The following table lists the Bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then –

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) = 12 i.e., 0000 1100 |
| ∣ | Binary OR Operator copies a bit if it exists in either operand. | (A ∣ B) = 61 i.e., 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) = 49 i.e., 0011 0001 |
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) = -61 i.e., 1100 0011 in 2's complement form. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 = 240 i.e., 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 = 15 i.e., 0000 1111 |

**MODULE 6: FILE OPERATIONS IN C**

**FILE**

It is a collection of data bytes that stored on secondary storage devices

EXAMPLE

text files, binary files

**BASIC FILE OPERATIONS IN C PROGRAMMING**

1. Opening/Creating a file
2. Closing a file
3. Reading a file
4. Writing in a file

|  |  |
| --- | --- |
| **File operation** | **Declaration & Description** |
| **fopen()**– To open a file | Declaration: FILE \***fopen**(const char \*filename, const char \*mode)  fopen() function is used to open a file to perform operations such as reading, writing etc. In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist.  FILE \*fp; fp=**fopen**(“filename”, ”‘mode”);  Where, fp – file pointer to the data type “FILE”. filename – the actual file name with full path of the file. mode – refers to the operation that will be performed on the file. Example: r, w, a, r+, w+ and a+. Please refer below the description for these mode of operations. |
| **fclose()**– To close a file | Declaration: int **fclose**(FILE \*fp);  fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as below. **fclose**(fp); |
| **fgets()** – To read a file | Declaration: char \***fgets**(char \*string, int n, FILE \*fp)  fgets function is used to read a file line by line. In a C program, we use fgets function as below. **fgets** (buffer, size, fp);  where, buffer – buffer to  put the data in. size – size of the buffer fp – file pointer |
| **fprintf()**– To write into a file | Declaration: int **fprintf**(FILE \*fp, const char \*format, …);fprintf() function writes string into a file pointed by fp. In a C program, we write string into a file as below.fprintf (fp, “some data”); or fprintf (fp, “text %d”, variable\_name); |

**MODE OF OPERATIONS PERFORMED ON A FILE IN C LANGUAGE:**

Mode Description

r opens a text file in reading mode

w opens or create a text file in writing mode.

a opens a text file in append mode

r+ opens a text file in both reading and writing mode

w+ opens a text file in both reading and writing mode

a+ opens a text file in both reading and writing mode

rb opens a binary file in reading mode

wb opens or create a binary file in writing mode

ab opens a binary file in append mode

rb+ opens a binary file in both reading and writing mode

wb+ opens a binary file in both reading and writing mode

ab+ opens a binary file in both reading and writing mode

**INBUILT FUNCTIONS FOR FILE HANDLING IN C LANGUAGE:**

|  |  |
| --- | --- |
| **File handling functions** | **Description** |
| [fopen ()](http://fresh2refresh.com/c-programming/c-file-handling/fopen-fclose-gets-fputs-functions-c/) | fopen () function creates a new file or opens an existing file. |
| [fclose ()](http://fresh2refresh.com/c-programming/c-file-handling/fopen-fclose-gets-fputs-functions-c/) | fclose () function closes an opened file. |
| [getw ()](http://fresh2refresh.com/c-programming/c-file-handling/getw-putw-functions-c/) | getw () function reads an integer from file. |
| [putw ()](http://fresh2refresh.com/c-programming/c-file-handling/getw-putw-functions-c/) | putw () functions writes an integer to file. |
| [fgetc ()](http://fresh2refresh.com/c-programming/c-file-handling/fgetc-function-c/) | fgetc () function reads a character from file. |
| [fputc ()](http://fresh2refresh.com/c-programming/c-file-handling/fputc-function-c/) | fputc () functions write a character to file. |
| [fgets ()](http://fresh2refresh.com/c-programming/c-file-handling/fgets-function-c/) | fgets () function reads string from a file, one line at a time. |
| [fputs ()](http://fresh2refresh.com/c-programming/c-file-handling/fopen-fclose-gets-fputs-functions-c/) | fputs () function writes string to a file. |
| [feof ()](http://fresh2refresh.com/c-programming/c-file-handling/feof-function-c/) | feof () function finds end of file. |
| [fgetchar ()](http://fresh2refresh.com/c-programming/c-file-handling/fgetchar-function-c/) | fgetchar () function reads a character from keyboard. |
| [fprintf ()](http://fresh2refresh.com/c-programming/c-file-handling/fscanf-fprintf-ftell-rewind-functions-c/) | fprintf () function writes formatted data to a file. |
| [fscanf ()](http://fresh2refresh.com/c-programming/c-file-handling/fscanf-fprintf-ftell-rewind-functions-c/) | fscanf () function reads formatted data from a file. |
| [fputchar ()](http://fresh2refresh.com/c-programming/c-file-handling/fputchar-function-c/) | fputchar () function writes a character onto the output screen from keyboard input. |
| [fseek ()](http://fresh2refresh.com/c-programming/c-file-handling/fseek-seek_set-seek_cur-seek_end-functions-c/) | fseek () function moves file pointer position to given location.  Syntax : int fseek ( FILE \* STREAM , long offset , int whence)  Where  FILE \*STREAM : file pointer  Offset : pointer position after the whence position  Whence : haves three file pointer locations denoted by  SEEK\_SET, SEEK\_CUR, SEEK\_END |
| [SEEK\_SET](http://fresh2refresh.com/c-programming/c-file-handling/fseek-seek_set-seek_cur-seek_end-functions-c/) | SEEK\_SET moves file pointer position to the beginning of the file  Value : 0 |
| [SEEK\_CUR](http://fresh2refresh.com/c-programming/c-file-handling/fseek-seek_set-seek_cur-seek_end-functions-c/) | SEEK\_CUR moves file pointer position to given location.  Value : 1 |
| [SEEK\_END](http://fresh2refresh.com/c-programming/c-file-handling/fseek-seek_set-seek_cur-seek_end-functions-c/) | SEEK\_END moves file pointer position to the end of file.  Value: 2 |
| [ftell ()](http://fresh2refresh.com/c-programming/c-file-handling/fscanf-fprintf-ftell-rewind-functions-c/) | ftell () function gives current position of file pointer. (ON ERROR IT RETURNS -1 VALUE) |
| [rewind ()](http://fresh2refresh.com/c-programming/c-file-handling/fscanf-fprintf-ftell-rewind-functions-c/) | rewind () function moves file pointer position to the beginning of the file. |
| [getc ()](http://fresh2refresh.com/c-programming/c-file-handling/getc-putc-functions-c/) | getc () function reads character from file. |
| [putc ()](http://fresh2refresh.com/c-programming/c-file-handling/getc-putc-functions-c/) | putc () function writes a character to file. |
| [remove ()](http://fresh2refresh.com/c-programming/c-file-handling/remove-function-c/) | remove () function deletes a file. |
| [fflush ()](http://fresh2refresh.com/c-programming/c-file-handling/fflush-function-c/) | fflush () function flushes a file. |

**COMMAND LINE ARGUMENTS**

THESE ARE THE VALUE PASSED TO THE PROGRAM WHEN THEY EXECUTE IN THE COMMAND LINE ON RUNIG THE PROGRAM. THE COMMAND LINE ARGUMENTS ARE HANDLED USING MAIN() FUNCTION ARGUMENTS WHERE **ARGC** REFERS TO THE NUMBER OF ARGUMENTS PASSED, AND **ARGV[]** IS A POINTER ARRAY WHICH POINTS TO EACH ARGUMENT PASSED TO THE PROGRAM.

int main(int argc, char \*argv[]) // command line arguments

./a.out this is a program

Where,

argc = 5

argv[0] = “./a.out”

argv[1] = “this”

argv[2] = “is”

argv[3] = “a”

argv[4] = “program”

argv[5] = NULL

**PERROR()**

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

int main()

{

FILE \*fp;

char filename[40] = "test.txt";

/\* Let us consider test.txt not available \*/

fp = f open(filename, "r");

if(fp == NULL)

{

perror("File not found");

printf("errno : %d.\n", errno);

return 1;

}

printf("File is found and opened for reading");

fclose(fp);

}

Ferror;

Syntax : int ferror (FILE \*STREAM)

Ferror returns a non zero if an errorf is detected on the named stream