1**.Rules for identifiers**

Identifier refers to name given to entities such as variables, functions, structures etc.Identifier must be unique. They are created to give unique name to a entity to identify it during the execution of the program. For example:int money;double accountBalance;

Rules for an Identifier

An Identifier can only have alphanumeric characters(a-z , A-Z , 0-9) and underscore( \_ ).

The first character of an identifier can only contain alphabet(a-z , A-Z) or underscore ( \_ ).

Identifiers are also case sensitive in C. For example name and Name are two different identifiers in C.

## 2. Associativity of unary operators

If two operators of same precedence (priority) is present in an expression, Associativity of operators indicate the order in which they execute.

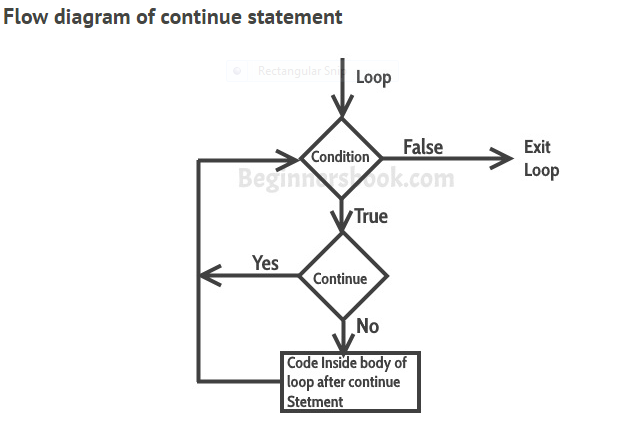
Eg.1==2!=3

Here, operators == and != have same precedence. The associativity of both == and != is left to right, i.e, the expression on the left is executed first and moves towards the right.

The **unary operators** operate on a single operand and following are the examples of **Unary operators** − The increment (++) and decrement (--)**operators**. The **unary** minus (-) **operator**. The logical not (!) **operator**. Associativity of unary from Right to left

3 **.Explain with example continue statement**

.The **continue statement** is used inside [loops](https://beginnersbook.com/2014/01/c-loops-examples/). When a continue statement is encountered inside a loop, control jumps to the beginning of the loop for next iteration, skipping the execution of statements inside the body of loop for the current iteration.



#include<stdio.h>

include<conio.h>

**void** main()

{

**int**i=1;

clrscr();

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

{

**if**(i==3)//if value of i is equal to 3, it will continue the loop

{

**continue**;

}

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

}//end of for loop

getch();

}

o/p: 1 2 4 5 6 7 8 9 10

4 **Discuss on Library Functions**

.Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library.

* Each library function in C performs specific operation.
* We can make use of these library functions to get the pre-defined output instead of writing our own code to get those outputs.
* These library functions are created by the persons who designed and created C compilers.
* All C standard library functions are declared in many header files which are saved as file\_name.h.
* We are including these header files in our C program using “#include<file\_name.h>” command to make use of the functions those are declared in the header files.

5**.Prototype of a function with integer array and a float number as arguments and return a float array.**

Float [] function\_name(float,int []);

A function prototype is a function declaration that specifies the data types of its arguments in the parameter list. The compiler uses the information in a function prototype to ensure that the corresponding function definition and all corresponding function declarations and calls within the scope of the prototype contain the correct number of arguments or parameters, and that each argument or parameter is of the correct data type.

Eg.Intadd(int,int);

6**. Find the length of a string using pointers.**

#include<stdio.h>

#include<conio.h>

intstring\_ln(char\*);

void main() {

   charstr[20];

   int length;

   clrscr();

   printf("\nEnter any string : ");

   gets(str);

   length = string\_ln(str);

   printf("The length of the given string %s is : %d", str, length);

   getch();

}

Int string\_ln(char\*p) /\* p=&str[0] \*/

{

   int count = 0;

   while (\*p != '\0') {

      count++;

      p++;

   }

   return count;

}

7. Different ways to initialize Two dimensional array.

There are two ways to initialize a two Dimensional arrays during declaration.

intdisp[2][4]={

{10,11,12,13},

{14,15,16,17}

};

OR

intdisp[2][4]={10,11,12,13,14,15,16,17};

#include <stdio.h>

8. **Discuss on enumerated Datatypes.**

An enumeration is a user-defined data type that consists of integral constants. To define an enumeration, keyword enum is used.

enum week { sunday, monday, tuesday, wednesday, thursday, friday, saturday };

int main()

{

enum week today;

today = wednesday;

printf("Day %d",today+1);

return 0;

}

Output: Day 4

Another example of enumeration is:

// Another example program to demonstrate working

// of enum in C

#include<stdio.h>

enum year{Jan, Feb, Mar, Apr, May, Jun, Jul,

Aug, Sep, Oct, Nov, Dec};

int main()

{

inti;

for (i=Jan; i<=Dec; i++)

printf("%d ", i);

return 0;

}

Output:

0 1 2 3 4 5 6 7 8 9 10 11

9) Difference between \*ptr++ and (\*ptr)++

\*ptr++: ptr is incremented by one and gets the value inside that memory location.

(\*ptr)++: value inside ptr is incremented by one

10. **STRUCTURE USING POINTER**

C structure can be accessed in 2 ways in a C program. They are,

Using normal structure variable

Using pointer variable

Dot(.) operator is used to access the data using normal structure variable and arrow (->) is used to access the data using pointer variable. You have learnt how to access structure data using normal variable in C – Structure topic. So, we are showing here how to access structure data using pointer variable in below C program.

EXAMPLE PROGRAM FOR C STRUCTURE USING POINTER:

In this program, “record1” is normal structure variable and “ptr” is pointer structure variable. As you know, Dot(.) operator is used to access the data using normal structure variable and arrow(->) is used to access data using pointer variable.

#include <stdio.h>

#include <string.h>

struct student

{

int id;

char name[30];

float percentage;

};

int main() {

inti;

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

struct student \*ptr;

ptr = &record1;

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

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

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

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

return 0;

}

OUTPUT:

Records of STUDENT1:

Id is: 1

Name is: Raju

Percentage is: 90.500000

11).**File Opening Mode Chart :**

|  |  |
| --- | --- |
| **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 |

### 12 .Reading from a binary file

.For reading and writing to a text file, we use the functions fprintf() and fscanf().They are just the file versions of printf() and scanf(). The only difference is that, fprint and fscanf expects a pointer to the structure FILE.

Functions fread() and fwrite() are used for reading from and writing to a file on the disk respectively in case of binary files.

To write into a binary file, you need to use the function fwrite(). The functions takes four arguments: Address of data to be written in disk, Size of data to be written in disk, number of such type of data and pointer to the file where you want to write.

fwrite(address\_data,size\_data,numbers\_data,pointer\_to\_file);

### Reading from a binary file

Function fread() also take 4 arguments similar to fwrite() function as above.

fread(address\_data,size\_data,numbers\_data,pointer\_to\_file);

### 13. Explain about Static variables

### Static variables

**Scope:** Local to the block in which the variable is defined

**Default initial value:** 0(Zero).

**Lifetime:** Till the whole program doesn't finish its execution.

A static variable tells the compiler to persist/save the variable until the end of program. Instead of creating and destroying a variable every time when it comes into and goes out of scope, staticvariable is initialized only once and remains into existence till the end of the program. A staticvariable can either be internal or external depending upon the place of declaration. Scope of **internal static** variable remains inside the function in which it is defined. **External static** variables remain restricted to scope of file in which they are declared.

They are assigned **0 (zero)** as default value by the compiler.

#include<stdio.h>

void test(); //Function declaration (discussed in next topic)

int main()

{

test();

test();

test();

}

void test()

{

static int a = 0; //a static variable

a = a + 1;

printf("%d\t",a);

}

1 2 3

14**. Explain the arguments passed to the main function as command line arguments.**

It is possible to pass some values from the command line to your C programs when they are executed. These values are called **command line arguments**and many times they are important for your program especially when you want to control your program from outside instead of hard coding those values inside the code.

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.

**argv[0]** holds the name of the program itself and **argv[1]** is a pointer to the first command line argument supplied, and \*argv[n] is the last argument. If no arguments are supplied, argc will be one, and if you pass one argument then **argc** is set at 2.

#include <stdio.h>

int main( int argc, char \*argv[] ) {

printf("Program name %s\n", argv[0]);

if( argc == 2 ) {

printf("The argument supplied is %s\n", argv[1]);

}

else if( argc > 2 ) {

printf("Too many arguments supplied.\n");

}

else {

printf("One argument expected.\n");

}

}

15. **What is a NULL pointer?**

Some of the most common use cases for NULL are  
a) To initialize a pointer variable when that pointer variable isn’t assigned any valid memory address yet.  
b) To check for null pointer before accessing any pointer variable. By doing so, we can perform error handling in pointer related code e.g. dereference pointer variable only if it’s not NULL.  
c) To pass a null pointer to a function argument when we don’t want to pass any valid memory address.

The example of a) is

|  |
| --- |
| int \* pInt = NULL; |

16. **Explain the use of indirection operator with the help of an example.**

The indirection operator (\*) is also called the dereference operator. When a pointer is dereferenced, the value at the address stored by the pointer is retrieved.

int x;

int \*p; *// \* is used in the declaration:*

*// p is a pointer to an integer, since (after dereferencing),*

*// \*p is an integer*

x = 0;

*// now x == 0*

p = &x; *// & takes the address of x*

*// now p == &x, so \*p == x*

\*p = 1; *// equivalent to x = 1, since \*p == x*

*// now \*p == 1 and \*p == x, so x == 1*

17 a) Write a C program to compute the sum of first n terms of the series: **1+2/3!+3/5!+4/7!+…**

Int main()

{

Int I,j,n;

Float sum=0;

Printf(“enter limit”);

Scanf(“%d”,&n);

For(i=1,j=1;i<n;i++,j=j+2)

{

S=s+i/fact(j);

}

Printf(“sum is %f”,sum);

}

Int fact(int n)

{int I,f=1;

For(i=1;i<n;i++)

{f=f\*I;}

Return f;

}

# 17.b What do you mean by by typecasting?

# Type Conversion in C

A type cast is basically a conversion from one type to another. There are two types of type conversion:

1. **Implicit Type Conversion** Also known as ‘automatic type conversion’.
   * Done by the compiler on its own, without any external trigger from the user.
   * Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
   * All the data types of the variables are upgraded to the data type of the variable with largest data type.
   * **bool -> char -> short int -> int ->**
   * **unsigned int -> long -> unsigned ->**
   * **long long -> float -> double -> long double**
   * It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

**Example of Type Implicit Conversion:**

|  |
| --- |
| // An example of implicit conversion  #include<stdio.h>  int main()  {      int x = 10;    // integer x      char y = 'a';  // character c      // y implicitly converted to int. ASCII      // value of 'a' is 97      x = x + y;      // x is implicitly converted to float      float z = x + 1.0;      printf("x = %d, z = %f", x, z);      return 0;  } |

Output:

x = 107, z = 108.000000

1. **Explicit Type Conversion**– This process is also called type casting and it is user defined. Here the user can type cast the result to make it of a particular data type.

The syntax in C:

(type) expression

Type indicated the data type to which the final result is converted.

|  |
| --- |
| // C program to demonstrate explicit type casting  #include<stdio.h>  int main()  {    double x = 1.2;      // Explicit conversion from double to int      int sum = (int)x + 1;      printf("sum = %d", sum);      return 0;  } |

Output:

sum = 2

Advantages of Type Conversion

* + This is done to take advantage of certain features of type hierarchies or type representations.
  + It helps us to compute expressions containing variables of different data types.

18. Discuss the difference between call by value and call by passing techniques with the help of suitable 'examples.

| **SR. NO.** | **CALL BY REFERENCE** | **CALL BY VALUE** |
| --- | --- | --- |
| **1** | **The address of the variables are passed as arguments to the function in Pass By Reference method.** | **The value of the variables are passed as arguments to the function in Pass By Value method.** |
| **2** | **No copy of the variables are created on the memory stack.** | **A copy of the original variables are created in memory stack.** |
| **3** | **The modifications are made to the original variables which is the old value that was passed to the function.** | **The modifications are made to the new variableswhich are known as formal variables.** |
| **4** | **The changes are visible in every function.** | **The changes are visible only in a particular method where the arguments are passed.** |
| **5** | **The pointers are used in call by reference approach.** | **No pointers are used in call by value approach.** |
| **6** | **The pass by reference method creates only a single copy, thereby, providing memory efficiency.** | **The pass by value approach creates two or more copies of the original variables and, hence it is not memory efficient.** |
| **7** | **The actual and formal parameters are created on the same memory stack.** | **The actual and formal parameters are created on a different memory stack.** |

#include <stdio.h>

void swapByValue(int, int); */\* Prototype \*/*

int main() */\* Main function \*/*

{

int n1 = 10, n2 = 20;

*/\* actual arguments will be as it is \*/*

swapByValue(n1, n2);

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("n1: %d, n2: %d**\n**", n1, n2);

}

void swapByValue(int a, int b)

{

int t;

t = a; a = b; b = t;

}

OUTPUT

======

n1: 10, n2: 20

#include <stdio.h>

void swapByReference(int\*, int\*); */\* Prototype \*/*

int main() */\* Main function \*/*

{

int n1 = 10, n2 = 20;

*/\* actual arguments will be altered \*/*

swapByReference(&n1, &n2);

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("n1: %d, n2: %d**\n**", n1, n2);

}

void swapByReference(int \*a, int \*b)

{

int t;

t = \*a; \*a = \*b; \*b = t;

}

OUTPUT

======

n1: 20, n2: 10

18 b) Explain any two bit level operators with exarnples.

C provides six [operators](https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B) for [bit manipulation](https://en.wikipedia.org/wiki/Bit_manipulation).[[1]](https://en.wikipedia.org/wiki/Bitwise_operations_in_C#cite_note-k&r2e-1)

|  |  |
| --- | --- |
| **Symbol** | **Operator** |
| & | bitwise AND |
| | | bitwise inclusive OR |
| ^ | bitwise XOR (eXclusive OR) |
| << | left shift |
| >> | right shift |
| ~ | bitwise NOT (one's complement) (unary) |

### Bitwise AND &

|  |  |  |
| --- | --- | --- |
| **bit a** | **bit b** | a & b**(a AND b)** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

The bitwise AND operator is a single ampersand: &. It is just a representation of AND which does its work on the bits of the operands rather than the truth value of the operands. Bitwise binary AND does the logical **AND** (as shown in the table above) of the bits in each position of a number in its binary form.

For instance, working with a byte (the char type):

11001000

& 10111000

--------

= 10001000

The [most significant bit](https://en.wikipedia.org/wiki/Most_significant_bit) of the first number is 1 and that of the second number is also 1 so the most significant [bit](https://en.wikipedia.org/wiki/Bit) of the result is 1; in the second most significant bit, the bit of second number is zero, so we have the result as 0. [[2]](https://en.wikipedia.org/wiki/Bitwise_operations_in_C#cite_note-cprogramming.com-2)

### Bitwise OR |

|  |  |  |
| --- | --- | --- |
| **bit a** | **bit b** | **a | b (a OR b)** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Similar to bitwise AND, bitwise OR only operates at the bit level. Its result is a 1 if one of the either bits is 1 and zero only when both bits are 0. Its symbol is | which can be called a pipe.

11001000

| 10111000

--------

= 11111000

### Bitwise XOR ^

|  |  |  |
| --- | --- | --- |
| **bit a** | **bit b** | a ^ b**(a XOR b)** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

The bitwise XOR (exclusive or) performs a logical XOR function, which is equivalent to adding two bits and discarding the carry. The result is zero only when we have two zeroes or two ones. XOR can be used to toggle the bits between 1 and 0. Thus i = i ^ 1 when used in a loop toggles its values between 1 and 0.[[4]](https://en.wikipedia.org/wiki/Bitwise_operations_in_C#cite_note-4)

11001000

^ 10111000

--------

= 01110000

### Bitwise NOT ~ / ones' complement (unary)

|  |  |
| --- | --- |
| **bit a** | ~a**(complement of a)** |
| 0 | 1 |
| 1 | 0 |

The ones' complement (~) or the bitwise complement gets us the complement of a given number. Thus we get the bits inverted, for every bit 1 the result is bit 0 and conversely for every bit 0 we have a bit 1. This operation should not be confused with [logical negation](https://en.wikipedia.org/wiki/Negation) [!](https://en.wikipedia.org/wiki/C_operators#Logical_operators).

~ 11001000

--------

= 00110111

## Shift operators

There are two bitwise shift operators. They are

* Right shift (>>)
* Left shift (<<)

### Right shift >>

The symbol of right shift operator is >>. For its operation, it requires two [operands](https://en.wikipedia.org/wiki/Operand). It shifts each bit in its left operand to the right. The number following the operator decides the number of places the bits are shifted (i.e. the right operand). Thus by doing ch >> 3 all the bits will be shifted to the right by three places and so on.

Example:

If the variable ch contains the bit pattern 11100101, then ch >> 1 will produce the result 01110010, and ch >> 2 will produce 00111001.

Here blank spaces are generated simultaneously on the left when the bits are shifted to the right. When performed on an unsigned type, the operation performed is a [logical shift](https://en.wikipedia.org/wiki/Logical_shift), causing the blanks to be filled by 0s (zeros). When performed on a signed type, the result is technically undefined and compiler dependant,[[5]](https://en.wikipedia.org/wiki/Bitwise_operations_in_C#cite_note-5) however most compilers will perform an [arithmetic shift](https://en.wikipedia.org/wiki/Arithmetic_shift), causing the blank to be filled with the sign bit of the left operand.

Right shift can be used to divide a bit pattern by 2 as shown:

i = 14; *// Bit pattern 00001110*

j = i >> 1; *// here we have the bit pattern shifted by 1 thus we get 00000111 = 7 which is 14/2*

### 19 a)Write a C program to perform selection sort on a set of numbers. The set of

### numbers should be accessed using a pointer pointing to the first element. Do not use

### an array to hold the numbers.

int selectionsort(int \*arr,int n)

{

    int i,j,temp;

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

        for(j=i+1;j<n;j++)

        {

            if(\*(arr+i)>\*(arr+j))

            {

                temp=\*(arr+j);

                \*(arr+j)=\*(arr+i);

                \*(arr+i)=temp;

            }

        }

    return 0;

}

### 19 b. What is the use of typedef construct in C?

### typedef in C

typedef is a keyword used in C language to assign alternative names to existing datatypes. Its mostly used with user defined datatypes, when names of the datatypes become slightly complicated to use in programs. Following is the general syntax for using typedef,

typedef <existing\_name> <alias\_name>

Lets take an example and see how typedef actually works.

typedef unsigned long ulong;

The above statement define a term ulong for an unsigned long datatype. Now this ulong identifier can be used to define unsigned long type variables.

ulong i, j;

20. Write a C program to concatenate two strings without using any standard library

functions.

#include<stdio.h>

#include<string.h>

void concat(char[], char[]);

int main() {

   char s1[50], s2[30];

   printf("\nEnter String 1 :");

   gets(s1);

   printf("\nEnter String 2 :");

   gets(s2);

   concat(s1, s2);

   printf("nConcated string is :%s", s1);

   return (0);

}

void concat(char s1[], char s2[]) {

   int i, j;

   i = strlen(s1);

   for (j = 0; s2[j] != '\0'; i++, j++) {

      s1[i] = s2[j];

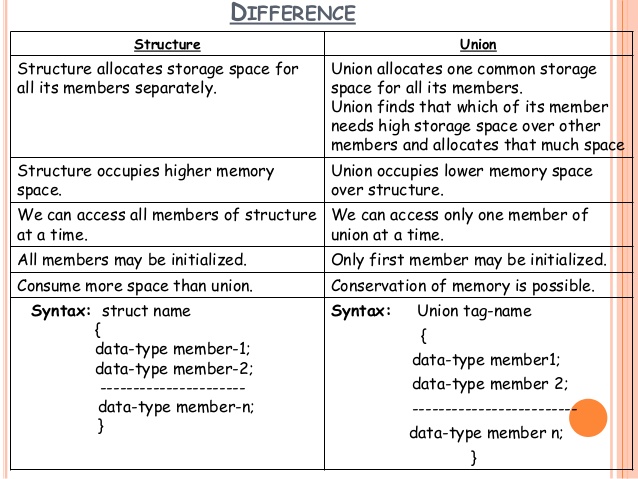
   }

   s1[i] = '\0';

}

20.b. Discuss the differences and sirnilarities between a structure variable and a union

variable.



21 a) Write a recursive function to perform binary search on a set of sorted numbers.

|  |  |  |  |
| --- | --- | --- | --- |
| #include<stdio.h>  #include<stdlib.h>  #define size 10  int binsearch(int[], int, int, int);  int main() {     int num, i, key, position;     int low, high, list[size];     printf("\nEnter the total number of elements");     scanf("%d", &num);     printf("\nEnter the elements of list :");     for (i = 0; i < num; i++) {        scanf("%d", &list[i]);     }     low = 0;     high = num - 1;      printf("\nEnter element to be searched : ");     scanf("%d", &key);     position = binsearch(list, key, low, high);      if (position != -1) {        printf("\nNumber present at %d", (position + 1));     } else        printf("\n The number is not present in the list");     return (0);  }  // Binary Search function  int binsearch(int a[], int x, int low, int high) {     int mid;     if (low > high)        return -1;     mid = (low + high) / 2;     if (x == a[mid]) {        return (mid);     } else if (x < a[mid]) {        binsearch(a, x, low, mid - 1);     } else {        binsearch(a, x, mid + 1, high);     }  } | | | |
| 1  2  3  4 | Enter the total number of elements : 5  Enter the elements of list : 11 22 33 44 55  Enter element to be searched : 33  Number present at 3 |

# 21 b) How do you declare constants in C?

# Different ways to declare variable as constant in C and C++

There are many different ways to make the variable as constant

1. **Using**[**const keyword**](https://www.geeksforgeeks.org/const-qualifier-in-c/)**:** The const keyword specifies that a variable or object value is constant and can’t be modified at the compilation time.

|  |
| --- |
| // C program to demonstrate const specifier  #include <stdio.h>  int main()  {    const int num = 1;      num = 5; // Modifying the value      return 0;  } |

It will throw as error like:

**error: assignment of read-only variable ‘num’**

1. **Using**[**enum keyword**](https://www.geeksforgeeks.org/enumeration-enum-c/)**:** Enumeration (or enum) is a user defined data type in C and C++. It is mainly used to assign names to integral constants, that make a program easy to read and maintain.

|  |
| --- |
| // In C and C++ internally the default  // type of 'var' is int  enum VARS { var = 42 };  // In C++ 11 (can have any integral type):  enum : type { var = 42; }    // where mytype = int, char, long etc.  // but it can't be float, double or  // user defined data type. |

**Note:** The data types of **enum** are of course limited as we can see in above example.

1. **Using**[**constexpr**](https://www.geeksforgeeks.org/understanding-constexper-specifier-in-c/)**keyword:** Using constexpr in C++(not in C) can be used to declare variable as a guaranteed constant. But it would fail to compile if its initializer isn’t a constant expression.

|  |
| --- |
| #include <iostream>  int main()  {      int var = 5;      constexpr int k = var;      std::cout << k;      return 0;  } |

Above program will throw an error i.e.,

error: the value of ‘var’ is not usable in a constant expression

because the variable ‘var’ in not constant expression. Hence in order to make it as constant, we have to declare the variable ‘var’ with **const** keyword.

1. **Using**[**Macros**](https://www.geeksforgeeks.org/interesting-facts-preprocessors-c/)**:** We can also use Macros to define constant, but there is a catch,

|  |
| --- |
| #define var 5 |

Since Macros are handled by the pre-processor(the pre-processor does text replacement in our source file, replacing all occurrences of ‘var’ with the literal 5) not by the compiler.  
Hence it wouldn’t be recommended because Macros doesn’t carry type checking information and also prone to error. In fact not quite constant as ‘var’ can be redefined like this,

|  |
| --- |
| // C program to demonstrate the problems  // in 'Macros'  #include <stdio.h>  #define var 5  int main()  {      printf("%d ", var);  #ifdef var  #undef var  // redefine var as 10  #define var 10  #endif      printf("%d", var);      return 0;  } |

**Output:**

5 10

22 a) Wite a program to compute the product of two matrices. Use array of pointers to access the matrices.

Hint:

Using **Pointers**, the code is like this:

for(i=0;i<r1;i++)

for(j=0;j<c2;j++)

for(k=0;k<c1;k++)

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

Another method:

// Print product of both matrix

printf("Product of both matrices is : \n");

matrixPrint(product);

**return** 0;

}

/\*\*

\* Function to input elements in matrix from user.

\*

\* @mat Two-dimensional array to store user input.

\*/

**void** matrixInput(**int** mat[][COL])

{

**int** row, col;

**for** (row = 0; row < ROW; row++)

{

**for** (col = 0; col < COL; col++)

{

scanf("%d", (\*(mat + row) + col));

}

}

}

/\*\*

\* Function to print elements in a two-dimensional array.

\*

\* @mat Two-dimensional array to print.

\*/

**void** matrixPrint(**int** mat[][COL])

{

**int** row, col;

**for** (row = 0; row < ROW; row++)

{

**for** (col = 0; col < COL; col++)

{

printf("%d ", \*(\*(mat + row) + col));

}

printf("\n");

}

}

/\*\*

\* Function to multiply two matrices.

\*

\* @mat1 First matrix

\* @mat2 Second matrix

\* @res Resultant matrix to store product of both matrices.

\*/

**void** matrixMultiply(**int** mat1[][COL], **int** mat2[][COL], **int** res[][COL])

{

**int** row, col, i;

**int** sum;

**for** (row = 0; row < ROW; row++)

{

**for** (col = 0; col < COL; col++)

{

sum = 0;

/\*

\* Find sum of product of each elements of

\* rows of first matrix and columns of second

\* matrix.

\*/

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

{

sum += (\*(\*(mat1 + row) + i)) \* (\*(\*(mat2 + i) + col));

}

/\*

\* Store sum of product of row of first matrix

\* and column of second matrix to resultant matrix.

\*/

\*(\*(res + row) + col) = sum;

}

}

}

22 b) Explain dangling else problem.

When there is multiple IF and a single ELSE then the ELSE part doesn't get a clear view to go with which IF, this problem is called dangling else problem.

For Eg:

if(....)

{

}

if(.....)

{

}

else

{

}

23.a) Assume there are fwo files first.Dd and second.trt. Write a C program to merge the contents of two files into a new file third.ort.

Let the given two files be file1.txt and file2.txt. The following are steps to merge.   
1) Open file1.txt and file2.txt in read mode.  
2) Open file3.txt in write mode.  
3) Run a loop to one by one copy characters of file1.txt to file3.txt.  
4) Run a loop to one by one copy characters of file2.txt to file3.txt.  
5) Close all files.

To successfully run the below program file1.txt and fil2.txt must exits in same folder.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main()  {     // Open two files to be merged     FILE \*fp1 = fopen("file1.txt", "r");     FILE \*fp2 = fopen("file2.txt", "r");       // Open file to store the result     FILE \*fp3 = fopen("file3.txt", "w");     char c;     if (fp1 == NULL || fp2 == NULL || fp3 == NULL)     {           puts("Could not open files");           exit(0);     }       // Copy contents of first file to file3.txt     while ((c = fgetc(fp1)) != EOF)        fputc(c, fp3);     // Copy contents of second file to file3.txt     while ((c = fgetc(fp2)) != EOF)        fputc(c, fp3);     printf("Merged file1.txt and file2.txt into file3.txt");     fclose(fp1);     fclose(fp2);     fclose(fp3);     return 0;  } |

23. b) What is dynamic memory allocation?

**Dynamic Memory Allocation in C**

The process of allocating memory at runtime is known as **dynamic memory allocation**. Library routines known as **memory management functions** are used for allocating and freeing memory during execution of a program. These functions are defined in **stdlib.h** header file.

|  |  |
| --- | --- |
| **Function** | **Description** |
| malloc() | allocates requested size of bytes and returns a void pointer pointing to the first byte of the allocated space |
| calloc() | allocates space for an array of elements, initialize them to zero and then returns a void pointer to the memory |
| Free | releases previously allocated memory |
| realloc | modify the size of previously allocated space |

24)a A student database stores following information about students in a class: Rollno,

name, gender, CGPA .Write a program to prepare a rank list based on CGPA . Also prepare a list of students having CGPA iess than 7.

#include<stdio.h>  
struct student  
{  
          int rollno;  
          char name[20];  
          char college[40];  
          int score;  
};  
void main()  
    {  
          struct student s[20],temp;  
          int i,j,n;  
          clrscr();  
          printf("\nEnter no. of Students : ");  
          scanf("%d",&n);  
          printf("\nEnter the rollno,name,college name,score ");  
          for(i=0;i<n;i++)  
          scanf("%d%s%s%d",&s[i].rollno,s[i].name,s[i].college,&s[i].score);  
          for(i=0;i<=n-1;i++)  
          {  
            for(j=0;j<=n-1;j++)  
            {  
                if(s[j].score<s[j+1].score)  
                {  
                  temp=s[j];  
                  s[j]=s[j+1];  
                  s[j+1]=temp;  
                }  
            }  
          }  
          printf("\nThe Rank List is :\n");  
          for(j=0;j<n;j++)  
          printf("%d\t%s\t%s\t%d\n",s[j].rollno,s[j].name,s[j].college,s[j].score);  
          getch();  
        }

24 b**) Explain scope of a variable with suitable examples.**

Every variable in C programming has two properties: type and storage class.Type refers to the data type of a variable. And, storage class determines the scope and lifetime of a variable.There are 4 types of storage class:

1. Automatic
2. external
3. static
4. register

## Local Variable

The variables declared inside the function are automatic or local variables.The local variables exist only inside the function in which it is declared. When the function exits, the local variables are destroyed.

int main() {

int n; // n is a local variable to main() function ... .. ...

}

void func() {

int n1; // n1 is local to func() function

}

In the above code, n1 is destroyed when func() exits. Likewise, n gets destroyed when main() exits.

## Global Variable

Variables that are declared outside of all functions are known as external variables. External or global variables are accessible to any function.

### Example #1: External Variable

#include <stdio.h>

void display();

int n = 5; // global variable

int main()

{

++n; // variable n is not declared in the main() function

display();

return 0;

}

void display()

{

++n; // variable n is not declared in the display() function

printf("n = %d", n);

}

**Output**

n = 7

Suppose, a global variable is declared in file1. If you try to use that variable in a different file file2, the compiler will complain. To solve this problem, keyword extern is used in file2 to indicate that the external variable is declared in another file.

## Register Variable

The register keyword is used to declare register variables. Register variables were supposed to be faster than local variables.However, modern compilers are very good at code optimization and there is a rare chance that using register variables will make your program faster. Unless you are working on embedded system where you know how to optimize code for the given application, there is no use of register variables.

## Static Variable

A static variable is declared by using keyword static. For example; The value of a static variable persists until the end of the program

static int i;

.Example #2: Static Variable

#include <stdio.h>

void display();

int main()

{

display();

display();

}

void display()

{

static int c = 0;

printf("%d ",c);

c += 5;

}

**Output**

0 5

During the first function call, the value of c is equal to 0. Then, it's value is increased by 5.During the second function call, variable c is not initialized to 0 again. It's because c is a static variable. So, 5 is displayed on the screen.

# 25, STRUCTURE OF A C- PROGRAM

Basic Structure of a C Program:

* Documentation section : The documentation section consists of a set of comment lines giving the name of the program, the author and other details, which the programmer would like to use later.
* Link section : The link section provides instructions to the compiler to link functions from the system library.
* Definition section : The definition section defines all symbolic constants.
* Global declaration section : There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions. This section also declares all the user-defined functions.
* main () function section : Every C program must have one main function section. This section contains two parts; declaration part and executable partDeclaration part : The declaration part declares all the variables used in the executable part.Executable part : There is at least one statement in the executable part. These two parts must appear between the opening and closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program. All statements in the declaration and executable part end with a semicolon.
* Subprogram section : The subprogram section contains all the user-defined functions that are called in the main () function. User-defined functions are generally placed immediately after the main () function, although they may appear in any order. Note:All section, except the main () function section may be absent when they are not required.

Sample C Program:

#include<stdio.h> <———————-Preprocessing Directive

void main()

{ <——————–Start of a Program

/\*………….Printing Starts………….\*/

Printf(“Learn at every moment”);

/\*………….Printing starts……..\*/

} <———————-End of a Program

In C many library functions are grouped category-wise and stored in different files known as header files. Ex. stdio.h–>standard input output header file

To use the functions defined in the header file that need to be included in the program

This can be achieved by the preprocessing directive “#include”

“#include” includes the content of header file(stdio.h) at the beginning of program.

**26.FEATURES OF C –LANGUAGE**

* It is a robust language with rich set of built-in functions and operators that can be used to write any complex program.
* The C compiler combines the capabilities of an assembly language with features of a high-level language.
* Programs Written in C are efficient and fast. This is due to its variety of data type and powerful operators.
* It is many time faster than BASIC.
* C is highly portable this means that programs once written can be run on another machines with little or no modification.
* Another important feature of C program, is its ability to extend itself.
* A C program is basically a collection of functions that are supported by C library. We can also create our own function and add it to C library.
* C language is the most widely used language in operating systems and embedded system development today.

## DATA TYPES IN C LANGUAGE

* FUNDAMENTAL DATA TYPES

It is also called as Primitive data type.   
[1.](http://cquestionbank.blogspot.com/2011/02/char-in-c.html)Char

[2.](http://cquestionbank.blogspot.com/2011/02/int-in-c.html)int  
[3.](http://cquestionbank.blogspot.com/2011/02/float-in-c.html)float  
4.double[5.](http://cquestionbank.blogspot.com/2011/02/void-in-c.html)void

DERIVED DATA TYPES

* Data types that are derived from fundamental data types are called derived data types. Derived data types don't create a new data type but,instead they add some functionality to the basic data types.
* In C, two derived data type are : Array & Pointer.
* Array : An array is a collection of variables of same type. They are stored in contagious memory allocation.
* e.g
* int a[10];
* char chi [20];

POINTER

A pointer is a special variable that holds a memory address (location in memory) of another variable.

int i=10;

int \*j;

j=&i;

Here, j is a integer pointer as it holds an address of an integer variable i.

Derived data types are derived from fundamental data types(ie: int, float, char, double,void). They don't create a new data type but use fundamental data type to add extra feature. Ex: Array: An Array is collection of variables of same type. Hence array is an derived data type.

USER DEFINED DATA TYPES

The user defined data types enable a program to invent his own data types and define what values it can taken on. Thus these data types can help a programmer to reducing programming errors.

C supports 2 types of user defined data types.

* typedef (type definition)
* enum (enumerated data type)

## The typedef keyword.

The typedef keyword allows the programmer to create new data types :- in a way.   
You typically use the typedef keyword to improve the readability of your code and as a means   
ofabbreviatingit.   
For example,

the statement

    typedef unsigned char byte;

creates a new datatype called byte which is short-hand for unsigned char.  You can the use   
this new type to declare variables as follows:

    byte a;

ENUMERATED DATA TYPE

An enumeration is a user-defined data type that consists of integral constants. To define an enumeration, keyword enum is used.

enum flag { const1, const2, ..., constN };

Here, name of the enumeration is flag.

## And, const1, const2,...., constN are values of type flag.

## Enumerated Type Declaration

When you create an enumerated type, only blueprint for the variable is created. Here's how you can create variables of enum type.

enum boolean { false, true };

enum boolean check;

## OPERATORS IN C

An operator is a symbol which operates on a value or a variable. For example: + is an operator to perform addition

|  |
| --- |
| Operators in C programming |
| [Arithmetic Operators](https://www.programiz.com/c-programming/c-operators#arithmetic) |
| [Increment and Decrement Operators](https://www.programiz.com/c-programming/c-operators#increment) |
| [Assignment Operators](https://www.programiz.com/c-programming/c-operators#assignment) |
| [Relational Operators](https://www.programiz.com/c-programming/c-operators#relational) |
| [Logical Operators](https://www.programiz.com/c-programming/c-operators#logical) |
| [Conditional Operators](https://www.programiz.com/c-programming/c-operators#conditional) |
| [Bitwise Operators](https://www.programiz.com/c-programming/c-operators#bitwise) |
| [Special Operators](https://www.programiz.com/c-programming/c-operators#other) |

## Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).

| Operator | Meaning of Operator |
| --- | --- |
| + | addition or unary plus |
| - | subtraction or unary minus |
| \* | multiplication |
| / | division |
| % | remainder after division( modulo division) |

## Increment and decrement operators

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

## Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

| Operator | Example | Same as |
| --- | --- | --- |
| = | a = b | a = b |
| += | a += b | a = a+b |
| -= | a -= b | a = a-b |
| \*= | a \*= b | a = a\*b |
| /= | a /= b | a = a/b |
| %= | a %= b | a = a%b |

### Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in [decision making](https://www.programiz.com/c-programming/c-if-else-statement) and [loops](https://www.programiz.com/c-programming/c-for-loop).

| Operator | Meaning of Operator | Example |
| --- | --- | --- |
| == | Equal to | 5 == 3 returns 0 |
| > | Greater than | 5 > 3 returns 1 |
| < | Less than | 5 < 3 returns 0 |
| != | Not equal to | 5 != 3 returns 1 |
| >= | Greater than or equal to | 5 >= 3 returns 1 |
| <= | Less than or equal to | 5 <= 3 return 0 |

### Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in [decision making in C programming](https://www.programiz.com/c-programming/c-if-else-statement).

| Operator | Meaning of Operator | Example |
| --- | --- | --- |
| && | Logial AND. True only if all operands are true | If c = 5 and d = 2 then, expression ((c == 5) && (d > 5)) equals to 0. |
| || | Logical OR. True only if either one operand is true | If c = 5 and d = 2 then, expression ((c == 5) || (d > 5)) equals to 1. |
| ! | Logical NOT. True only if the operand is 0 | If c = 5 then, expression ! (c == 5) equals to 0. |

### Bitwise Operators

During computation, mathematical operations like: addition, subtraction, addition and division are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

| Operators | Meaning of operators |
| --- | --- |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |
| ~ | Bitwise complement |
| << | Shift left |
| >> | Shift right |

## SELECTION STATEMENTS

A selection statement selects among a set of statements depending on the value of a controlling expression.

## The if Statement

The if statement has the following syntax:

if ( expression )

statement

else(opt)

else-statement(opt)

The statement following the control expression is executed if the value of the control expression is true (nonzero). An if statement can be written with an optional else clause that is executed if the control expression is false (0).

Consider the following example:

if (i < 1)

funct(i);

else

{

i = x++;

funct(i);

}

In this example, if the value of i is less than 1, then the statement funct(i) is executed and the compound statement following the keyword else is not executed. If the value of i is not less than 1, then only the compound statement following the keyword else is executed.

The control expression in a selection statement is usually a logical expression, but it can be any expression of scalar type.

## The switch Statement

The switch statement executes one or more of a series of cases, based on the value of a controlling expression. The switch statement has the following syntax:

switch ( expression )

statement

The switch statement is typically a compound statement, within which are one or more case statements executed if the control expression matches the case . The syntax for a case label and expression follows:

case constant-expression : statement

## .ITERATION STATEMENTS

**Iteration** is the process where a set of instructions or statements is executed repeatedly for a specified number of time or until a condition is met. These statements also alter the control flow of the program and thus can also be classified as **control statements** in C Programming Language.

### FOR LOOP

**For loop** is the most commonly used looping technique. The reason why it is so popular is because it has all the three parts of the loop: initialization, test expression, update expression in the same line. The syntax of a for loop is:

|  |  |
| --- | --- |
| 1  2  3  4 | for(initialization; test expression; update expression)  {    //body of loop  } |

Have a look at this simple **for loop** program for a better understanding:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | #include <stdio.h>    int main()  {      int i;      for(i=1; i<=5; i++)          printf("CodinGeek\n");      return 0;  } |

**Output:-**

CodinGeek

CodinGeek

CodinGeek

CodinGeek

CodinGeek

Here we have given the initial value of **i** as **1**(initialization). The test condition is **i<=5** and the update expression is **i++**.

**While loop**

A while loop in C programming repeatedly executes a target statement as long as a given condition is true.

## Syntax

The syntax of a while loop in C programming language is −

while(condition) {

statement(s);

}

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any nonzero value. The loop iterates while the condition is true.

When the condition becomes false, the program control passes to the line immediately following the loop.

## Example

#include <stdio.h>

int main () {

/\* local variable definition \*/

int a = 10;

/\* while loop execution \*/

while( a < 20 ) {

printf("value of a: %d\n", a);

a++;

}

return 0;

}

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

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

Do while loop

A do...while loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

## Syntax

The syntax of a do...while loop in C programming language is −

do {

statement(s);

} while( condition );

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

## Example

#include <stdio.h>

int main () {

/\* local variable definition \*/

int a = 10;

/\* do loop execution \*/

do {

printf("value of a: %d\n", a);

a = a + 1;

}while( a < 20 );

return 0;

}

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

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

1. **Write a C statement to declare an array to store marks of 50 students in English**

Ans : float marks[50];

1. **Write a C statement to declare a structure to employee id, name and base pay**

Ans :

Struct emp

{

char name[50];

int id;

float pay;

};

1. **Rewrite the following statement using conditional operator**

If (a>b) c=a;

Else

c=b;

Ans : c=(a>b)?a:b;

1. **Write a C statement block to copy a two dimensional array into another**

Ans :

int b[20][20];

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

{

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

{

b[i][j]=a[i][j]; /\*assume that a[][] is the entered array and ‘r’ and ‘c’ are the row and column size\*/

}

}

1. Write a function to store N names into an array of pointers

Ans :

1. **Write a macro to find cube of a given number**

Ans : #define cube(x) (x\*x\*x)

1. Write a recursive function to find value of X^n

Ans :

int power(int base, int power)

{

if (power== 1)

return base;

else

return (base\*power(base,power-1));

}

1. Distinguish between macro and a function

Ans:

|  |  |
| --- | --- |
| MACRO | FUNCTION |
| 1. Macro is preprocessed | 1. Function is compiled |
| 1. Speed of execution is faster | 2. Speed of execution is slower |
| 1. No type checking required | 3. Type checking required |
| 1. Macros are of one line code | 4. Functions can be of any number  of lines |
| 1. Before compilation macro name   is replaced by macro value | 5. During function call, transfer of  control takes place |
| Eg: #define MAX(X,Y) (X>Y ? X:Y) | Eg: int MAX(X,Y)  {  if(X<Y)  return Y;  else  return X;  } |
| 1. Macro does not check compile   errors | 6. Functions checks compile errors |
|  |  |

1. Illustrate accessing two dimensional array using pointers

Ans :

A 2D array is viewed as an array of 1D arrays. That is, each row in a 2D array is a 1D array. Therefore given a 2D array A,

int A[m][n]

we can think of A[0] as the address of row 0, A[1] as address of row 1 etc..

Hence to find, say A[0][2] we do the following

A[0][2] = \*(A[0] + 2)

In general, A[i][j] = \*(A[i] + j)

We also note that A[0] = \*A

Therefore A[i][j] = \*(A[i] + j) = \*(\*(A+i) + j)

1. Write a user defined function to compare two strings

Ans :

int str(char \*str1, char \*str2)

{

while (\*str1 != '\0') {

if (\*str1 != \*str2)

return 0;

str1++;

str2++;

}

return 1;

}

1.How maney times “Good morning” is get printed?

For(i=2;i<10;i++)

{Printf(“good morning”);}

Ans: 8 times

2.Write a seudocode to read a number and print whether it is divisible by 10 or not?

Ans:

#include<stdio.h>

int main()

{

int n;

printf("Enter the number;\t");

scanf("%d",&n);

if(n%10==0)

{

printf("The number %d is divisible by 10\n",n);

}

else

{

printf("The number %d is not divisible by 10\n",n);

}

return 0;

}

3.Explane case structure?

Ans;

In computer programming languages, a switch statement is a type of selection control mechanism used to allow the value of a variable or expression to change the control flow of program execution via a multiway branch.

4.Write a seudocode to print multiplication table of a given number?

Ans:

#include<stdio.h>

int main()

{

int n,i;

printf("Enter the number:");

scanf("%d",&n);

printf("The multiplication table od %d is:\n",n);

for(i=1;i<11;i++)

{

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

}

return 0;

}

5.write a program to declare a matrix having 5 rows and 6 columns and read the elements to the matrix?

Ans:

#include<stdio.h>

int main()

{

int m1[100][100],i,j;

printf("enter the elements of the matrice:\n");

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

{

for(j=0;j<6;j++)

{

scanf("%d",&m1[i][j]);

}

}

printf("\nThe entered matrix is:\n");

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

{

for(j=0;j<6;j++)

{

printf("%d\t",m1[i][j]);

}

printf("\n");

}

return 0;

}

6.Write the code to read 10 elements to an array and display it?

Ans:

#include<stdio.h>

int main()

{

int a[10],i;

printf("enter the elements of the array\n");

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

{

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

}

printf("The entered array is:\n");

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

{

printf("%d\t",a[i]);

}

printf("\n");

return 0;

}

7.With an example explain how values can be passed from a main program to a sub program?

Ans:

#include<stdio.h>

int main()

{

int n;

printf("enter the value\n");

scanf("%d",&n);

def(n);

return 0;

}

def(int n)

{

if(n%2==0)

{

printf("even number");

}

else

{

printf("odd number");

}

}

8.Write a program to read a month number(from 1 to 12) and print the corresponding month(1-january ,6-june)?

Ans:

#include<stdio.h>

int main()

{

int a;

printf("Enter the month number:");

scanf("%d",&a);

printf("The entered month is: ");

switch(a)

{

case 1:

printf("January\n");

break;

case 2:

printf("February\n");

break;

case 3:

printf("March\n");

break;

case 4:

printf("April\n");

break;

case 5:

printf("May\n");

break;

case 6:

printf("June\n");

break;

case 7:

printf("July\n");

break;

case 8:

printf("August\n");

break;

case 9:

printf("September\n");

break;

case 10:

printf("October\n");

break;

case 11:

printf("November\n");

break;

case 12:

printf("December\n");

break;

default:

printf("Not found\n");

break;

}

return 0;

}

9.Write a program to read two numbers and find sum of all the number divisible by 5 between them(excluding the given numbers)?

Ans:

#include<stdio.h>

int main()

{

int m,n,a,sum=0,i;

printf("Enter the 1st numbers:");

scanf("%d",&n);

printf("Enter tne 2nd number:");

scanf("%d",&m);

if(n>m)

{

a=n;

n=m;

m=a;

}

for(i=n+1;i<m;i++)

{

if(i%5==0)

{

sum=sum+i;

}

}

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

return 0;

}

1. Preprocessor directives:

The [C preprocessor](https://www.cprogramming.com/tutorial/cpreprocessor.html) modifies a source file before handing it over to the compiler, allowing conditional compilation with [#ifdef](https://www.cprogramming.com/reference/preprocessor/ifdef.html), defining constants with [#define](https://www.cprogramming.com/reference/preprocessor/define.html), including header files with [#include](https://www.cprogramming.com/reference/preprocessor/include.html), and using built in macros such as [\_\_FILE\_\_](https://www.cprogramming.com/reference/preprocessor/__FILE__.html). This is the list of preprocessor directives, or commands to the preprocessor, that are available:

[#include](https://www.cprogramming.com/reference/preprocessor/include.html)

[#define](https://www.cprogramming.com/reference/preprocessor/define.html)

[#undef](https://www.cprogramming.com/reference/preprocessor/undef.html)

[#if](https://www.cprogramming.com/reference/preprocessor/if.html)

[#ifdef](https://www.cprogramming.com/reference/preprocessor/ifdef.html)

[#ifndef](https://www.cprogramming.com/reference/preprocessor/ifndef.html)

[#error](https://www.cprogramming.com/reference/preprocessor/error.html)1

[\_\_FILE\_\_](https://www.cprogramming.com/reference/preprocessor/__FILE__.html)

[\_\_LINE\_\_](https://www.cprogramming.com/reference/preprocessor/__LINE__.html)

[\_\_DATE\_\_](https://www.cprogramming.com/reference/preprocessor/__DATE__.html)

[\_\_TIME\_\_](https://www.cprogramming.com/reference/preprocessor/__TIME__.html)

[\_\_TIMESTAMP\_\_](https://www.cprogramming.com/reference/preprocessor/__TIMESTAMP__.html)

[pragma](https://www.cprogramming.com/reference/preprocessor/pragma.html)

[# macro operator](https://www.cprogramming.com/reference/preprocessor/stringizing-operator.html)

[## macro operator](https://www.cprogramming.com/reference/preprocessor/token-pasting-operator.html)

2. Comments in c:

In the C Programming Language, you can place comments in your source code that are not executed as part of the program.

Comments provide clarity to the C source code allowing others to better understand what the code was intended to accomplish and greatly helping in debugging the code.

A comment starts with a slash asterisk /\* and ends with a asterisk slash \*/ and can be anywhere in your program.

Comment in Single Line

One can create a comment on a single line.

For example:

/\*to print hai\*/

Comment Spans Multiple Lines

You can create a comment that spans multiple lines. For example:

/\*

\* Type: multiple line comment

\* Purpose: To show a comment that spans multiple lines.

\* Language: C

\*/

The compiler will assume that everything after the /\* symbol is a comment until it reaches the \*/ symbol, even if it spans multiple lines within the C program.

3. Features of c:

* It is a robust language with rich set of built-in functions and operators that can be used to write any complex program.
* The C compiler combines the capabilities of an assembly language with features of a high-level language.
* Programs Written in C are efficient and fast. This is due to its variety of data type and powerful operators.
* It is many time faster than BASIC.
* C is highly portable this means that programs once written can be run on another machines with little or no modification.
* Another important feature of C program, is its ability to extend itself.
* A C program is basically a collection of functions that are supported by C library. We can also create our own function and add it to C library.
* C language is the most widely used language in operating systems and embedded system development today.

4. Linker and Loader:

Linker: In high level languages, some built in header files or libraries are stored. These libraries are predefined and these contain basic functions which are essential for executing the program. These functions are linked to the libraries by a program called Linker. If linker does not find a library of a function then it informs to compiler and then compiler generates an error. The compiler automatically invokes the linker as the last step in compiling a program. Not built in libraries, it also links the user defined functions to the user defined libraries. Usually a longer program is divided into smaller subprograms called modules. And these modules must be combined to execute the program. The process of combining the modules is done by the linker.

Loader: Loader is a program that loads machine codes of a program into the system memory. In Computing, a loader is the part of an Operating System that is responsible for loading programs. It is one of the essential stages in the process of starting a program. Because it places programs into memory and prepares them for execution. Loading a program involves reading the contents of executable file into memory. Once loading is complete, the operating system starts the program by passing control to the loaded program code. All operating systems that support program loading have loaders. In many operating systems the loader is permanently resident in memory.

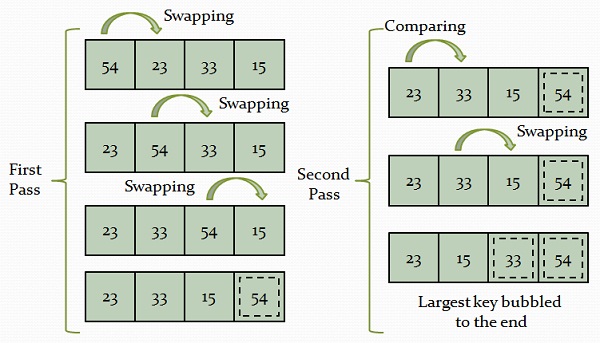
5.Header files in c:

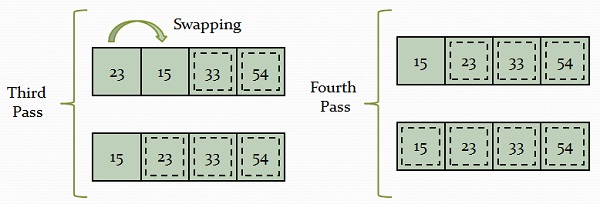
A header file is generally used to define all of the functions, variables and constants contained in any function library that you might want to use. The header file stdio.h should be used if you want to use the two standard I/O functions printf and scanf.

A header file is a file with extension .h which contains C function declarations and macro definitions to be shared between several source files. There are two types of header files: the files that the programmer writes and the files that comes with your compiler.

6. Sorting: Bubble sort & selection sort:

Bubble sort:

**Bubble sort** is the simplest iterative algorithm operates by comparing each item or element with the item next to it and swapping them if needed. In simple words, it compares the first and second element of the list and swaps it unless they are out of specific order. Similarly, Second and third element are compared and swapped, and this comparing and swapping go on to the end of the list. The number of comparisons in the first iteration are n-1 where n is the number elements in an array. The largest element would be at nth position after the first iteration. And after each iteration, the number of comparisons decreases and at last iteration only one comparison takes place. 



#include <stdio.h>

int main()

{

int a[100],n,i,d,temp;

  printf("Enter number of elements\n");

scanf("%d",&n);

  printf("Enter %d integers\n", n);

  for(i=0;i<n;i++)

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

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

{

for(d=0;d<n-i-1;d++)

{

if(a[d]>a[d+1])

{

temp=a[d];

a[d]=a[d+1];

a[d+1]=temp;

}

}

}

printf("Sorted list in ascending order:\n");

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

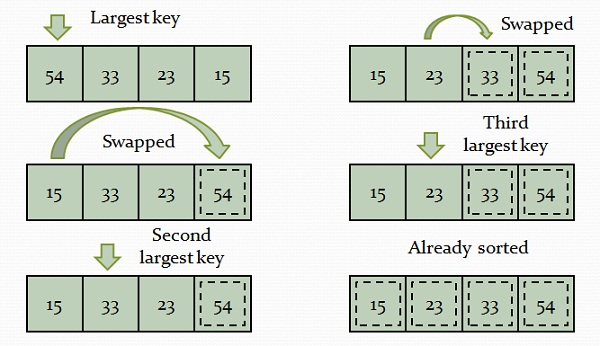
printf("%d\n", a[i]);

return 0;

}

Selection sort:

**Selection sort** has achieved slightly better performance and is efficient than bubble sort algorithm. Suppose we want to arrange an array in ascending order then it functions by finding the largest element and exchanging it with the last element, and repeat the following process on the sub-arrays till the whole list is sorted.



#include <stdio.h>

int main()

{

int array[100], n, c, d, position, swap;

  printf("Enter number of elements\n");

scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for ( c = 0 ; c < n ; c++ )

scanf("%d", &array[c]);

  for ( c = 0 ; c < ( n - 1 ) ; c++ )

{

position = c;

  for ( d = c + 1 ; d < n ; d++ )

{

if ( array[position] > array[d] )

position = d;

}

if ( position != c )

{

swap = array[c];

array[c] = array[position];

array[position] = swap;

}

}

  printf("Sorted list in ascending order:\n");

  for ( c = 0 ; c < n ; c++ )

printf("%d\n", array[c]);

  return 0;

}

7. Linear search & Binary search:

Linear search:

Linear search in C programming: The following code implements linear search (Searching algorithm) which is used to find whether a given number is present in an array and if it is present then at what location it occurs. It is also known as sequential search. It is straightforward and works as follows: We keep on comparing each element with the element to search until it is found or the list ends.

#include <stdio.h>

int main()

{

int array[100], search, c, n;

printf("Enter the number of elements in array\n");

scanf("%d", &n);

printf("Enter %d integer(s)\n", n);

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

scanf("%d", &array[c]);

printf("Enter a number to search\n");

scanf("%d", &search);

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

{

if (array[c] == search)

{

printf("%d is present at location %d.\n", search, c+1);

break;

}

}

if (c == n)

printf("%d isn't present in the array.\n", search);

return 0;

}

Binary search:

C program for binary search: This code implements binary search in C language. It can only be used for sorted arrays, but it's fast as compared to linear search. If you wish to use binary search on an array which isn't sorted, then you must sort it using some sorting technique say merge sort and then use the binary search algorithm to find the desired element in the list. If the element to be searched is found then its position is printed. The code below assumes that the input numbers are in ascending order.

#include <stdio.h>

int main()

{

int c, first, last, middle, n, search, array[100];

printf("Enter number of elements\n");

scanf("%d",&n);

printf("Enter %d integers\n", n);

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

scanf("%d",&array[c]);

printf("Enter value to find\n");

scanf("%d", &search);

first = 0;

last = n - 1;

middle = (first+last)/2;

while (first <= last) {

if (array[middle] < search)

first = middle + 1;

else if (array[middle] == search) {

printf("%d found at location %d.\n", search, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last)/2;

}

if (first > last)

printf("Not found! %d isn't present in the list.\n", search);

return 0;

}

1. What is a file?

Ans. A file is a computer resource for recording data discretely into a computer storage device.A file may

be designed to store a picture,a video,a computer program or a wide variety of other kinds of data.

By using computer programs, a person can open, read, change, and close a computer file. Typically,

files are organised in a file system,which keeps track of where the files are located on disk and

enables user access.

2.What are binary files and text files?

Ans.Text files are normal files that you can easily create using notepd or any other text editors.All the

Contents within these files will be plain text and you can easily edit or delete those contents.

On the other hand,binary files store the contents in the binary form(in the form of 0’s and 1’s).

They can hold higher amount of data,are not easily readable and hence provide better security

than text files.

3.What is a File pointer?

Ans.A file pointer is a pointer to a structure,which contains information about the file,including its name,

current position of the file,whether the file is being read or written and whether errors or end of

the file has occurred.The only declaration needed for a file pointer is symbolized by:

FILE \*fp;

This says that fp is the file pointer that points to a file structure.

4.What are the various file opening modes in C?

Ans.

|  |  |
| --- | --- |
| MODE | MEANING |
| r | Open a text file for reading |
| w | Create a text file for writing |
| a | Append to a text file |
| rb | Open a binary file for reading |
| wb | Create a binary file for writing |
| ab | Append to a binary file |
| r+ | Open a text file for readinf/writing |
| w+ | Create a text file for reading/writing |
| a+f or af+ | Append or create a text file for read/write |
| r+b or rb+ | Open a binary file for read/write |
| w+b or wb+ | Create a binary file for read/write |
| a+b or ab+ | Append a binary file for read/write |

5.What are the various file operations in C?

Ans.In C,the different operations that can be performed on files are:

* Creation of a new file
* Opening an existing file
* Reading from file
* Writing to a file
* Moving to a specific location in a file
* Closing a file

6.Explain the functions fputs() and fgets().

Ans.fputs() is a standard library function in C programming language that writes an array of characters to

a given file stream.The prototype of the function is as follows:

int fputs ( const char \* str, FILE \* stream );

fgets() is a standard library function in C that reads a file line by line.The prototype of the function is

as follows:

char \***fgets**(char \*string, int n, FILE \*fp)

7.Explain the function ferror().

Ans.ferror() is a C library function that tests to see if the error indicator has been set for the given

stream. The prototype of the function is as follows:

int ferror(FILE \*stream)

8.Explain the functions fread() and fwrite() in C.

Ans.The fread() function is used to read data from the given stream into the array pointed to by a

Pointer. The prototype of the function is as follows:

size\_t fread(void \*ptr, size\_t size, size\_t nmemb, FILE \*stream);

The fwrite() function is used to write records(sequences of bytes)to the file.The record may be an array or

a structure. The prototype of the function is as follows:

fwrite( ptr, int size, int n, FILE \*fp );

9.Explain the random access functions in C?

Ans.C supports mainly three functions for random access file processing.

* fseek()
* ftell()
* rewind()

fseek()

This is used for seeking the pointer position in the file at the specified byte.

Syntax: fseek( file pointer, displacement, pointer position);  
 Where  
 file pointer ---- It is the pointer which points to the file.  
 displacement ---- It is positive or negative.This is the number of bytes which are skipped backward (if negative) or forward( if positive) from the current position.This is attached with L because this is a long integer.

For eg: fseek( p,10L,0)  
  
 0 means pointer position is on beginning of the file,from this statement pointer position is skipped 10 bytes from the beginning of the file.

ftell()

This function returns the value of the current pointer position in the file.The value is count from the

beginning of the file.

Syntax:rewind(fptr);  
 Where fptr is a file pointer.

rewind()

This function sets the file position to the beginning of the file of the given stream.

Syntax:void rewind(FILE\* stream);

10.Explain the functions fprintf() and fscanf().

Ans.fprintf() function is used to write formatted data into a file.

Syntax: int fprintf(FILE \*stream, const char \*format [, argument, ...]) ;

fscanf() function is used to read formatted data from a file.

Syntax: int fscanf(FILE \*stream, const char \*format [, argument, ...]) ;

1-Pointer definition:

A pointer is a variable whose value is the address of another variable, i.e. address of the memory location. Any variable or constant must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is –

type \*variable name;

Here, type is the pointer's base type; it must be a valid C data type and variable name is the name of the pointer variable. The asterisk (\*) is used to designate a variable as a pointer.

some of the valid pointer declarations −

int \*ip; /\* pointer to an integer \*/ , double \*dp; /\* pointer to a double \*/..etc.

The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.We can also access the value at the address available in the pointer variable. This is done by using unary operator \* that returns the value of the variable located at the address specified by its operand. The following example makes use of these operations –

{ int var = 20; /\* actual variable declaration \*/

int \*p; /\* pointer variable declaration \*/

p = &var; /\* store address of var in pointer variable\*/

printf("Address stored in p variable: %x\n",p );

printf("Value of \*ip variable: %d\n", \*p );

}it produces the following result –

Address stored in p variable: bffd8b3c

Value of \*p variable: 20

2-Dereference operator (\*):

The dereference operator or indirection operator, denoted by "\*" is a unary operator that include pointer variables ,it refers to the variable being pointed. It operates on a pointer variable, and returns the value equivalent to the value at the pointer address. This is called "dereferencing" the pointer. For example,

int x;

int \*p;

x = 0

p = &x; // & takes the address of x

// now p == &x, so \*p == 0(the value stored in x)

3-Pointer arithemetic:

A pointer has an address, which is a numeric value. Therefore arithmetic operations can be performed on a pointer as on a numeric value. There are four arithmetic operators that can be used on pointers: ++, --, +, and -.when we perform any arithmetic function like increment on a pointer, changes occur as per the size of their primitive data type.

Size of data types on 16-bit Machine:

|  |
| --- |
| Type Size (in bytes)  int or signed int 2  char 1  float 4  double 8  long double 10 |

Examples for Pointer Arithmetic

int x,\*i,\*p;

i++; /\* the pointer will be of 2 bytes when we increment it\*/

x = 3 \* - \*i/ \*p; /\*same as(3\*(-(\*i)))/(\*p) \*/

4-Multiple indirection (pointer to pointer):

A pointer to a pointer is a form of multiple indirection , or a chain of pointers . This is done by placing an additional asterisk in front of its name. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value . The value is indirectly pointed to by a pointer. A variable there is a pointer to a pointer must be declared as such for example, the following declaration declares a pointer to a pointer of type int – 

int \*\*ptr1;

int\*ptr;

int var;

var = 3000;

ptr = &var;

printf("Value of var = %d\n", var );

printf("Value available at \*ptr = %d\n", \*ptr );

printf("Value available at \*\*pptr = %d\n", \*\*pptr);

//\*OUTPUT will be \*//

Value of var = 3000

Value available at \*ptr = 3000

Value available at \*\*pptr = 3000

5-Null pointer:

Null pointer has a value reserved for indicating that the pointer does not refer to a valid object .It is to initialize a pointer variable when that pointer variable isn’t assigned any valid memory address yet. NULL pointer is different from uninitialized pointer. We can pass a null pointer as function argument when we don’t want to pass any valid memory address //\*Defining NULL Value\*//

#define NULL 0

int \*ptr = NULL;

printf("The value of ptr is %u",ptr);

//\*Output :\*//

The value of ptr is 0

6-Pointer to function:

It is possible to declare a pointer pointing to a function which can then be used as an argument in another function. A pointer to a function is declared as follows,

type (\*pointer-name)(parameter);

An example :

int (\*sum)(); //legal declaration of pointer to function

int \*sum(); //This is incorrect it refers to a function returning a pointer type//

#include <stdio.h>

int sum(int x, int y)

{return x+y;

int main( )

{

int (\*fp)(int, int);

fp = sum;

int s = fp(10, 15);

printf("Sum is %d", s);

return 0;

}

//Output://

25

7-Array of pointers:

An array of pointers is a continuous block of memory that contains pointers to other memory locations. An array of pointers would be an array that holds memory locations.They essentially allow non-contiguous memory locations to be treated as if they were an actual array. when we want to maintain an array, which can store pointers to an int or char or any other data type available. Following is the declaration of an array of pointers to an integer −int \*ptr[MAX];

This declares ptr as an array of MAX integer point. Thus, each element in ptr, holds a pointer to an int value. For eg: following eg uses 2 integers, which are stored in an array of pointers, as follows −

#include<stdio.h>

const int MAX = 2;

int main ()

{int var[] = {10, 100};

int i, \*ptr[MAX];

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

{ptr[i] = &var[i]; /\* assign the address of integer. \*/

} for ( i = 0; i < MAX; i++)

{printf("Value of var[%d] = %d\n", i, \*ptr[i] );}

//Output//

Value of var[0] = 10

Value of var[1] = 100

1,5.Qus:Array one dimensional and Two dimensional, Initialisation

Array

Array is a sequenced collection of elements of same data type.

One dimensional array

A list of item can given one variable name using only one subscribt and such a variable is called a single-subscripted variable or a one dimensional array.

Syntax: type arrayName [ arraySize ];

The arraySize must be an integer constant greater than zero and type can be any valid C data type.

Example: double balance[10];

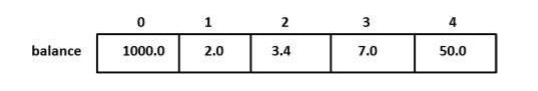
Here, balance is a variable array which is sufficient to hold up to 10 double numbers.

Initialisation: double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

Assign a single element of the array: balance[4] = 50.0;

Pictorial representation of the array



Accessing Array Elements:

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.

Example: double salary = balance[9];

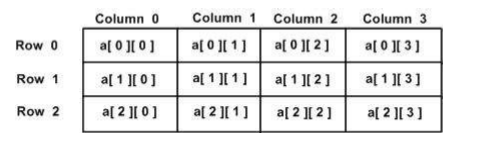
Two Dimensional array

A two-dimensional array is, in essence, a list of one-dimensional arrays.

Syntax: type arrayName [ x ][ y ];

Where type can be any valid C data type and arrayName will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns.

A two-dimensional array a, which contains three rows and four columns can be shown as follows:



Initializing Two-Dimensional Arrays:

int a[3][4] = { {0, 1, 2, 3} , {4, 5, 6, 7} , {8, 9, 10, 11}};

The nested braces, which indicate the intended row.

Another

int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};

Accessing Two-Dimensional Array Elements:

An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array.

int val = a[2][3];

The above statement will take the 4th element from the 3rd row of the array.

2,6.Qus:String,array of strings

String

Strings are actually one-dimensional array of characters terminated by a null character '\0'.

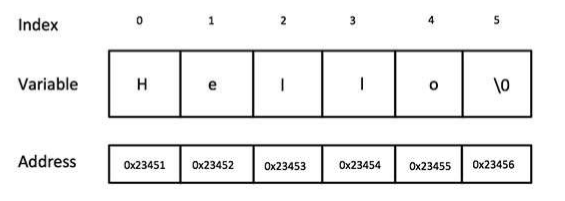
Initialisation:

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

Another form

char greeting[] = "Hello";

Memory representation of string



3.Qus:String handling functions

String Functions

1.Strcmp()

This function compares two strings identified by the arguments and has a value 0 if they are equal.If they are not, it has the numeric difference between the first nonmatching characters in the string

strcmp(s1, s2);

Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.

Example:

strcmp(“hai”,”hai”);

output:0

strcmp(“their”,”there”);

output:-9

2.Strcpy()

This function works almost like a string assignment operator.

strcpy(s1, s2);

Copies string s2 into string s1.

Strcpy(city,”delhi”);

Output of city:delhi

3.Strcat()

This function joins two strings together.

strcat(s1, s2);

Concatenates string s2 onto the end of string s1.

strcat(“good”,”\_moring”);

output:good\_mornig

4.Strlen()

This function counts and returns the number of characters in a string.

strlen(s1);

Returns the length of string s1.

strlen(“hai”);

output:3

4.Qus:passing array to a function

Passing array to a function

If you want to pass a single-dimension array as an argument in a function, you would have to declare a 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.

1.Formal parameters as a pointer:

Example: void myFunction(int \*param)

2. Formal parameters as a sized array:

Example: void myFunction(int param[10])

3. Formal parameters as an unsized array:

Example: void myFunction(int param[])

7.Qus: Multidimensional Arrays

Multidimensional Arrays

C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration.

Syntax: type name[size1][size2]...[sizeN];

Example: int threedim[5][10][4];

This is a three dimensional array

ANSI C does not specify any limit for array dimension. Most compailor permit seven to ten dimensions

1. write a c program to reverse a given array use only user defined function?

#include<stdio.h>

int main()

{

int n,c,d,a[100],b[100];

printf(“enter the array limit:”);

scanf(“%d”,&n);

printf(“Enter elements into array:”);

for(c=n-1,d=0;c>=0;c--,d++)

b[d]=a[c];

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

a[c]=b[c];

printf(“reversed array is\t:”);

for(c=o;c<n;c++)

printf(“%d\n”,a[c]);

return 0;

}

1. wap to find the factors of a given no using function?

#include<stdio.h>

factors(int);

factors(int n)

{

int i;

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

{

if(n%i==0)

{

printf(“%d”,i);

}

}

}

int main()

{

int n;

printf(“\n please enter a number:”);

scanf(“%d”,&n);

printf(“\n factors of given number are:\n”);

factors(n);

return 0;

}

1. largest among n numbers using pointer?

#include<stdio.h>

#include<conio.h>

int main()

{

int n,\*p,i,h=0;

printf(“enter a number limit:”);

scanf(“%d”,&n);

p=(int \*)malloc(sizeof(int));

if(p==null)

{

printf(“\n invalid entry:”);

exit(0);

}

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

{

printf(“enter”,%dn,”numbers”,i+1);

scanf(“%d”,p+i);

}

h=\*p;

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

{

if(\*(p+i)>h)

h=\*(p+i);

}

printf(“\n the largest number is %d\n”,h);

return 0;

}

1. concatenate two strings without std lib functions?

#include<stdio.h>

int main()

{

char str1[50],str2[50],i,j;

printf(“enter the first string”);

scanf(“%s”,&str1);

printf(“enter 2nd string:”);

scanf(“%s”,&str2);

for(i=0;str1[i]!=’\0’;++i);

for(j=0;str2[j]!=’\0’;++j,++i)

{

str1[i]=str2[j];

}

str1[i]=’\0’;

printf(“\noutput:%s”,str1);

return 0;

}

1. Explain pointers and its features?

A POINTER is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, a pointer must be declared first before using it to store any variable address.The general form of a pointer variable declaration is:

Type\*var-name;

Where, type is the pointer’s base type. It must be a valid C data type and var-name is the name of the pointer varible. The ‘\*’ operator used to declare a pointer is the same, which used for multiplication, is however used here to designate a variable as a pointer.

Features:

1. Pointer save memory space.
2. Execution time with pointers is faster, because data are manipulated with the address.

i.e; Direct access to memory location.

1. Memory is accessed efficiently with the pointers. The pointer assigns as well as releases the memory space. Memory is dynamically allocated.
2. We can access the elements of any type array, irrespective of its subscript range.
3. Pointers are more efficient in handling arrays and structures.
4. Pointers allow references to function and thereby helps in passing of function as arguments to other functions.
5. It reduces length of the program and its execution time as well.
6. Find the sum of elements of a matrix?

#include<stdio.h>

Void main()

{

Static int array[10][10];

Int I,j,m,n,sum=0;

Printf(“enter the order of the matrix\n:”);

Scanf(“%d%d”,&m,&n);

Printf(“Enter the elements of the matrix:”);

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

{

For(j=0;j<n;++j)

{

Scanf(“%d”,&array[i][j]);

}

}

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

{

For(j=0;j<n;j++)

}

Printf(“Sum of the %d row is=%d\n”,I,sum);

Sum=0;

}

Sum=0;

For(j=0;j<n;j++)

{

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

{

Sum=sum+array[i][j];

}

Printf(“Sum of the %d column is=%d\n”,j,sum);

Sum=0;

}

}

1. WAP to delete an element from an array?

#include<stdio.h>

int main()

{

int a[10],pos,c,n;

printf(“enter the array limit:”);

scanf(“%d”,&n);

printf(“enter %d elements into the array:”);

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

scanf(“%d”,&a[c]);

printf(“enter the location from which the element is to be deleted”);

scanf(“%d”,&pos);

if(pos>=n+1)

printf(“del not possible:”)

else

{

for(c=pos-1;c<n-1;c++)

array[c]=a[c+1];

printf(“resultant array is\n”);

for(c=0;c<n-1;c++)

printf(“%d\n”,a[c]);

}

return 0;

}

1. List any 3 input functions and any 3 output functions?

* Input Functions:-

1. Scanf.
2. Getchar().
3. Gets().

* Output Functions:-

1. Printf.
2. Putchar().
3. Puts()

* The standard input-output header file,named stdio.h contains the defination of the functions printf() and scanf(), which are used to display output on screen and to take input from user respectively.
* The getchar() function reads a character from the terminal and returns it as an integer.This function reads only single character at a time.We can use this method in a loop in case we want to read more than one character. The Putchar() function displays the character passed to it on the screen and returns the same character.This function too displays only a single character at a time.
* The gets() function reads a line from stdin(standard input) into the buffer pointed to by str pointer, until either a terminating newline or EOF occurs. The puts() function writes the string str and a trailing newline to stdout.
* Difference between scanf() and gets():-

The main difference between these two functions is that scanf() stops reading characters when it encounters a space, but gets() reads space as character too.

1. WAP to send 2 numbers into a user defined function, find the smallest among the two, using the function and return the result?

#include<stdio.h>

Int max(int n1,int n2);

Int min(int n1,int n2);

Int main()

{

Int n1,n2,min;

Printf(“Enter any 2 number:”);

Scanf(“%d%d”,%n2,&n2);

Min=min(n1,n2);

Printf(“\n minimum=%d\n”,min);

Return 0;

}

Int min(int n1;int n2)

{

Retrun(n1>n2)? n2 : n1;

}

1 .Constant and Volatile Qualifiers

Constant

'Const' qualifier will impose a restriction on the variable, such a way that its value can't be changed or modified. The main use of 'const' is to define constants in your program; so that it's values can't be changed

Constant  is used with a datatype declaration or definition to specify an unchanging value

* + Examples:
  + const int five = 5;
  + const double pi = 3.141593;
* const objects may not be changed
  + The following are illegal:
  + const int five = 5;
  + const double pi = 3.141593;
  + pi = 3.2;

five = 6;

volatile

If a variable is declared as 'volatile', its value can be changed from outside the program. Declaring a variable with 'volatile' is actually a hint to the compiler to not perform any optimizations on the access restrictions of the variable.

volatile specifies a variable whose value may be changed by processes outside the current program

One example of a volatile object might be a buffer used to exchange data with an external device:

int

check\_iobuf(void)

{

volatile int iobuf;

int val;

while (iobuf == 0) {

}

val = iobuf;

iobuf = 0;

return(val);

}

if iobuf had not been declared volatile, the compiler would notice that nothing happens inside the loop and thus eliminate the loop

* const and volatile can be used together
  + An input-only buffer for an external device could be declared as const volatile (or volatile const, order is not important) to make sure the compiler knows that the variable should not be changed (because it is input-only) and that its value may be altered by processes other than the current

2. Enumerated data type

* An enumerated type is one whose values are symbolic constant rather than literal.
* Declaration example: enum Jar\_Type {CUP, PINT, QUART, HALF\_GALLON, GALLON}; The above example declares a type called Jar\_Type Variables of this type are declared like this: enum Jar\_Type milk\_jug, gas\_can, medicine\_bottle;
* If there is only one declaration of variables of a particular enumerated type (i.e. no type name), both statements may be combined:

enum { CUP, PINT, QUARTER, HALF\_GALLON, GALLON} milk\_jug, gas\_can, medicine\_bottle;

* Variables declared with an enumerated type are actually stored as integers.
* Internally, the symbolic names are treated as integer constants

By default, CUP =0, PINT=1, QUART=2, etc.

* Caution: don’t mix them indiscriminately with integers – even though it is viable.

milk\_jug = -623;

int a = PINT;

* The variable cannot be assigned any values outside those specified in the initialization list for the declaration of the enum type

3.getchar() and putchar()

getchar() function

This Function reads a single character from the standard input device.There is no parameter within the parentheses.

Its syntax is:

char\_var = getchar();

where

char\_var is a character type variable to which an accepted character is assigned.

* The int getchar(void) function reads the next available character from the screen and returns it as an integer. This function reads only single Character at a time.you can use this method in the loop in case you want to read more than one character from the screen.

Example:

main()

{

char letter;

letter = getchar();

}

 putchar() Function:

This Function prints a single character on the screen. The character to be displayed is of type char.

Its syntax is:

putchar(ch\_var);

where ch\_var is a character variable which is enclosed within the parenthesis.

* The int putchar(int c) function puts the passed character on the screen and returns the same character.This function puts only single character at a time.you can use this method in the loop in case you want to display more than one character on the screen.

Example:

main()

{

char ch;

putchar(ch);

}

Example:

#include<stdio.h>

Int main()

{

Int c;

Printf(“enter a value “);

C=getchar();

Printf(“\nyou entered: “);

putchar(c);

return 0;

}

4.gets() and puts()

gets() function:

We use this function for storing a string value in memory. using this we don't need to use looping. and this function doesn't bound us about memory.

Syntax:

char str[size];

gets(str);

* gets() : Reads characters from the standard input and stores them as a string.
* The char \*gets(char \*s) function reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF(end of file).

Puts() function

* puts() : prints characters from the standard output.Just like printf statement.

The int puts(const char \*s) function writes the string ‘s’ and ‘a’ trailing newline to stdout.

Syntax:

puts(char\_array\_variable/string);

Difference between puts() and gets()

|  |  |
| --- | --- |
| Puts()function | gets()function |
| puts() stands for put string and is used to output a string | gets() stands for get string it is used to input a string |
| puts() is used to print this string. | As gets() is used to read strings which may include white spaces also |
| puts() is short for put\_string.it will print a given string. | gets() is short for get\_string. you type a string and it saves it in the memory. |

Example:

#include<stdio.h>

Int main()

{

Char str[100];

Printf(“enter a value:”);

gets( str );

printf(“\nyou entered:”);

puts(str);

return 0;

}

5.Formatted Input/Output printf and scanf

Concepts

* I/O is essentially done one character (or byte) at a time
* stream -- a sequence of characters flowing from one place to another
  + input stream: data flows from input device (keyboard, file, etc) into memory
  + output stream: data flows from memory to output device (monitor, file, printer, etc)
* Standard I/O streams (with built-in meaning)
  + stdin: standard input stream (default is keyboard)
  + stdout: standard output stream (defaults to monitor)
  + stderr: standard error stream
* stdio.h -- contains basic I/O functions
  + scanf: reads from standard input (stdin)
  + printf: writes to standard output (stdout)
  + There are other functions similar to printf and scanf that write to and read from other streams
  + How to include, for C or C++ compiler
  + #include <stdio.h> // for a C compiler
  + #include <cstdio> // for a C++ compiler
* Formatted I/O -- refers to the conversion of data to and from a stream of characters, for printing (or reading) in plain text format
  + All text I/O we do is considered formatted I/O
  + The other option is reading/writing direct binary information (common with file I/O, for example)

Printf()

* The basic format of a printf function call is:
* printf (format\_string, list\_of\_expressions);

where:

* + format\_string is the layout of what's being printed
  + list\_of\_expressions is a comma-separated list of variables or expressions yielding results to be inserted into the output
* To output string literals, just use one parameter on printf, the string itself

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

printf("Greetings, Earthling\n\n");

conversion specifier

A conversion specifier is a symbol that is used as a placeholder in a formatting string. For integer output (for example), %d is the specifier that holds the place for integers.

Here are some commonly used conversion specifiers (not a comprehensive list):

%d int (signed decimal integer)

%u unsigned decimal integer

%f floating point values (fixed notation) - float, double

%e floating point values (exponential notation)

%s string

%c character

scanf

Basics

* To read data in from standard input (keyboard), we call the scanf function. The basic form of a call to scanf is:
* scanf(format\_string, list\_of\_variable\_addresses);
  + The format string is like that of printf
  + But instead of expressions, we need space to store incoming data, hence the list of variable addresses
* If x is a variable, then the expression &x means "address of x"
* scanf example:
* int month, day;
* printf("Please enter your birth month, followed by the day: ");
* scanf("%d %d", &month, &day);

Conversion Specifier

Mostly the same as for output. Some small differences

* + Use %f for type float, but use %lf for types double and long double
* The data type read, the conversion specifier, and the variable used need to match in type
* White space is skipped by default in consecutive numeric reads. But it is not skipped for character/string inputs.

Example:

#include <stdio.h>

int main()

{

int i;

float f;

char c;

printf("Enter an integer and a float, then Y or N\n> ");

scanf("%d%f%c", &i, &f, &c);

printf("You entered:\n");

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

}

6.Prototype of printf()

The function int printf(const char \*format, ...); writes a formatted string to stdout(standard output device). If format contains format specifiers (subsequences beginning with %), the additional arguments following format are inserted after formatting in the resulting string by replacing their respective format specifiers.  
Function prototype of printf:

int printf(const char \*format, ...);

prototype of scanf()

The function int scanf(const char \*format, ...); reads formatted data from stdin(standard input device) and stores them in the variables pointed by the additional arguments. Additional arguments must point to variables of the same type as specified in the format.  
  
Function prototype of scanf:

int scanf(const char \*format, ...);

7.Different format specifiers

Format specifiers can be defined as the operators which are used in association with printf() function for printing the data that is referred by any object or any variable. When a value is stored in a particular variable, then you cannot print the value stored in the variable straightforwardly without using the format specifiers. You can retrieve the data that are stored in the variables and can print them onto the console screen by implementing these format specifiers in a printf() function.

Format specifiers start with a percentage % operator and followed by a special character for identifying the type of the data.

There are mostly six types of format specifiers that are available in C.

List of format specifiers in C

List of format specifier in c

|  |  |
| --- | --- |
| Format specifier | Description |
| %d | Integer format specifier |
| %f | Float format specifier |
| %c | Character format specifier |
| %s | String format specifier |
| %u | Unsigned Integer Format Specifier |
| %ld | Long Int Format Specifier |

Integer format specifier%d

The %d format specifier is implemented for representing integer values. This is used with printf() function for printing the integer value stored in the variable.

Syntax:

printf("%d",<variable name>);

float format specifier %f

The %f format specifier is implemented for representing fractional values. This is implemented within printf() function for printing the fractional or floating value stored in the variable. Whenever you need to print any fractional or floating data, you have to use %f format specifier.

Syntax:

printf("%f", <variable name>);

character format specifier %c

The %c format specifier is implemented for representing characters. This is used with printf() function for printing the character stored in a variable. When you want to print a character data, you should incorporate the %c format specifier.

Syntax:

printf("%c",<variable name>);

string format specifier %s

The %s format specifier is implemented for representing strings. This is used in printf() function for printing a string stored in the character array variable. When you have to print a string, you should implement the %s format specifier.

Syntax:

printf("%s",<variable name>);

unsigned integer format specifier %u

The %u format specifier is implemented for fetching values from the address of a variable having unsigned decimal integer stored in the memory. This is used within printf() function for printing the unsigned integer variable.

Syntax:

printf("%u",<variable name>);

Long int format specifier %ld

The %ld format specifier is implemented for representing long integer values. This is implemented with printf() function for printing the long integer value stored in the variable.

Syntax:

printf("%ld",<variable name>);

8.precision specifier

Width and Precision Specifiers

The minimum field width can be specified as a decimal digit string following any flag specifier, in which case the field width is set to the specified number of columns. The field width can also be specified as asterisk (\*) in which case an additional argument of type int is accessed to determine the field width. For example, to print an integer x in a field width determined by the value of the int variable w, you would write the D statement:

printf("%\*d", w, x);

The field width can also be specified using a ? character to indicate that the field width should be set based on the number of characters required to format an address in hexadecimal in the data model of the operating system kernel. The width is set to 8 if the kernel is using the 32-bit data model, or to 16 if the kernel is using the 64-bit data model. The precision for the conversion can be specified as a decimal digit string following a period (.) or by an asterisk (\*) following a period. If an asterisk is used to specify the precision, an additional argument of type int prior to the conversion argument is accessed to determine the precision. If both width and precision are specified as asterisks, the order of arguments to printf for the conversion should appear in the following order: width, precision, value.

1. How does x++ differ from ++x ? Explain with example.

Ans : Both ++x and x++ are used to increment variable x by 1. The prime difference is that ,

++x i.e. pre-increment operator uses the principle ‘change-then-use’ while, x++ i.e. post-increment operator uses the principle ‘use-then-change’.

For example, the following code;

int x=5, y=6;  
printf(“X=%d and Y=%d”,x++,++y);   
/\* x=5, y=7 y’s value is updated and used, x’s value gets updated but old value is used here\*/

printf (“X=%d and Y=%d”,x,y);

/\* x=6, y=7 y’s updated value is displayed and x’s updated value is used \*/

1. How is an array name interpreted when it is passed to a function ?

Ans : When we are passing an array as an argument the compiler will point towards the starting address of the array i.e. the array is not passed as a copy but only the starting address of the is passed to the function definition.

When passing an array as a parameter, this

void arraytest (int a[]) means exactly the same as void arraytest (int \*a)

In this example an array is passed to the function array test. When the function is called in the main function the address of the starting element is passed to the function definition.

1. How do you interpret the following function declaration

int\* p( char a[])

Ans : This function definition declares an integer function p which accepts a character array and returns a pointer to integer value.

1. Discuss the different parameters that are passed to main function as command line argument.

Ans :

1. How can you access structure member using a pointer to structure variable?

Ans : There are two ways of accessing members of structure using pointer:

Using indirection (\*) operator and dot(.) operator.  
Using arrow (->) operator or membership operator.

Eg :

struct dog{

char name[10], breed[10],color[10];

int age;

};

struct dog spike;

// declaring a pointer to a structure of type struct dog

struct dog \*ptr\_dog

Using indirection (\*) operator and dot(.) operator.

At this point ptr\_dog points to the structure variable spike, so by dereferencing it we will get the contents of the spike. This means spike and \*ptr\_dog are functionally equivalent. To access a member of structure write \*ptr\_dog followed by a dot(.) operator, followed by the name of the member. For example: (\*ptr\_dog).name - refers to the name of dog  
(\*ptr\_dog).breed - refers to the breed of dog and so on.

Parentheses around \*ptr\_dog are necessary because the precedence of dot(.) operator is greater than that of indirection (\*) operator.

The above method of accessing members of the structure using pointers is slightly confusing and less readable, that's why C provides another way to access members using the arrow (->) operator. To access members using arrow (->) operator write pointer variable followed by -> operator, followed by name of the member.

Using arrow (->) operator or membership operator.

The above method of accessing members of the structure using pointers is slightly confusing and less readable, that's why C provides another way to access members using the arrow (->) operator. To access members using arrow (->) operator write pointer variable followed by -> operator, followed by name of the member.

ptr\_dog->name - refers to the name of dog

ptr\_dog->breed - refers to the breed of dog

1. Write a program to concatenate two strings without using library functions.

Ans :

#include<stdio.h>

#include<string.h>

void concat(char[], char[]);

int main() {

char s1[50], s2[30];

printf("\nEnter String 1 :");

gets(s1);

printf("\nEnter String 2 :");

gets(s2);

concat(s1, s2);

printf("\nConcated string is :%s", s1);

return (0);

}

void concat(char s1[], char s2[]) {

int i, j;

i = strlen(s1);

for (j = 0; s2[j] != '\0'; i++, j++) {

s1[i] = s2[j];

}

s1[i] = '\0';

}

1. What is the use of indirection operator ?

Ans : An indirection operator is an operator used to obtain the value of a variable to which a pointer points. While a pointer pointing to a variable provides an indirect access to the value of the variable stored in its memory address, the indirection operator dereferences the pointer and returns the value of the variable at that memory location. The indirection operator is a unary operator represented by the symbol (\*).

The indirection operator can be used in a pointer to a pointer to an integer, a single-dimensional array of pointers to integers, a pointer to a char, and a pointer to an unknown type.

The indirection operator is also known as the dereference operator.

The (\*) symbol is used in declaring pointer types and in performing pointer indirection, while the ‘address-of’ operator () returns the address of a variable. Hence, the indirection operator and the address-of operator are inverses of each other.   
  
C# allows using pointers only in an unsafe region, which implies that the safety of the code within that region is not verified by the common language runtime (CLR). In the unsafe region, the indirection operator is allowed to read and write to a pointer. The following C# statements illustrate the usage of the indirection operator:

* int a = 1, b; // line 1
* int \*pInt = &a; // line 2
* b = \*pInt; // line 3

In the first line above, a and b are integer variables and a is assigned a value of 1. In line 2, the address of a is stored in the integer pointer pInt (line 2). The dereference operator is used in line 3 to assign the value at the address pointed to by pInt to the integer variable b.

1. Write a program

Ans :

#include<stdio.h>

#include<conio.h>

#include<process.h>

#include<ctype.h>

read\_file(char\*);

void main()

{

char ch,\*filename="file.txt";

FILE \*fpt;

int space,digit,vow,conso;

clrscr();

fpt=fopen(filename,"a+");

if(!fpt)

{

printf("\n\"%s\" Not Found . terminating......",filename);

exit(0);

}

spl=digit=conso=vow=0;

while((ch=fgetc(fpt))!=EOF)

{

if(isdigit(ch))

digit++;

else if(isalpha(ch))

{ switch(tolower(ch))

{

case 'a' : vow++;

break;

case 'e' : vow++;

break;

case 'i' : vow++;

break;

case 'o' : vow++;

break;

case 'u' : vow++;

break;

default : conso++;

break;

}

}

}

else

spl++;

fflush(fpt);

fprintf(fpt,"\nNo of vowels : %d",vow);

fprintf(fpt,"\nNo of consonants : %d",conso);

fprintf (fpt,"\nNo of digits : %d",digit);

fprintf(fpt,"\nno of Special Characters : %d",spl);

fclose(fpt);

printf("\n FILENAME : \"%s\"",filename);

read\_file(filename);

getch();

}

read\_file(char \*filename)

{

char ch;

FILE \*read=fopen(filename,"r");

if(!read)

{

printf("\n \"%s\" file could not be read!!---\n",filename);

getch();

exit(0);

}

printf("\nCONTENTS OF \"%s\"\n",filename);

while(1)

{

ch=fgetc(read);

if(ch!=EOF)

printf("%c",ch);

else

break;

}

fclose(read);

//getch();

}

1. General form of a function?

Syntax of a function :

return \_ type function\_ name (argument list)

{

Set of statements – Block of code

}

Return \_type: Return type can be of any data type such as int , double, char, void, short etc.

Function \_name: It can be anything, however it is advised to have a meaningful name for the functions.

Argument list: Argument list contains variables names along with their data types. Block of code: Set of C statements, which will be executed whenever a call will be made to the function.

1. What are formal and actual arguments?

Actual arguments :

Arguments which are mentioned in the function call is known as the actual arguments.

Formal arguments :

The formal arguments are the arguments in a function declaration. Formal arguments are a copy of the actual arguments. The scope of formal arguments is local to the function definition in which they are used. Formal arguments belong to the called function. A change in formal arguments would not be reflected in the actual arguments.

1. What are function return type?

The result of a function is called its return value and the data type of the return value is called the return type. 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. The default return value from a function is int.

1. Function declaration and definitions.

Function declaration is also known as function prototype .Name of parameters are not compulsory in function declaration only their type is required. If function definition is written before main function then function declaration is not required whereas, If function definition is written after main function then we must write function declaration before main function. Function declaration in C always ends with a semicolon.

For example:

int Sum(int, int)

A function definition specifies the name of the function, the types and number of parameters it expects to receive, and its return type. A function definition also includes a function body with the declarations of its local variables, and the statements that determine what the function does.

1. What is function prototype?

In C programming all functions must be declared before they are used. This is accomplished using function prototype. Prototypes enable complier to provide stronger type checking. Function prototypes help us trap bugs before they occur. In addition, they help verify that your program is working correctly.

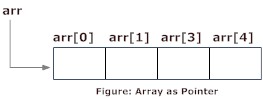
Syntax:

return\_type function\_name (type arg1, type arg2......);

1. Relationship between array and pointer?

In C programming, name of the array always points to address of the first element of an array .In the above example, arr and &arr[0] points to the address of the first element.

Since, the addresses of both are the same, the values of arr and &arr[0] are also the same. arr[0] is equivalent to \*arr (value of an address of the pointer).



1. Difference between structure and union?

|  |  |
| --- | --- |
| STRUCTURE | UNION |
|  |  |
| keyword struct is used to define a structure. | keyword union is used to define a union. |
| When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is greater than or equal to the sum of sizes of its members. | While in case of Union when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of union is equal to the size of largest member. |
|  |  |
| Each member within a structure is assigned unique storage area of location. | In case of union memory allocated is shared by individual members of union. |
| In Structure altering the value of a member will not affect other members of the structure. | In union altering the value of any of the member will alter other member values. |
| Members of structure can be accessed individually at any time. | At a time, only one member of union can be accessed. |
| Several members of a structure can initialize at once. | First member of a union can be initialized at a time. |
| Example:  #include <stdio.h>  struct Employee  {  int age;  char Name[50];  char Department[20];  float Salary;  };  int main()  {  struct Employee emp1;  printf("Size of Employee Structure= %d", sizeof (emp1) );  return 0;  }  Output:  Size of Employee Structure =80 | Example:  #include <stdio.h>  union Person  {  int age;  char Name[50];  char Department[20];  float Salary;  };  int main()  {  union Person Person1;  printf("Size of Person Union = %d", sizeof (Person1));  return 0;  }  Output:  Size of Person Union=52 |

Consider an array:

int arr[4];

1. Difference between call by value and call by reference with example.

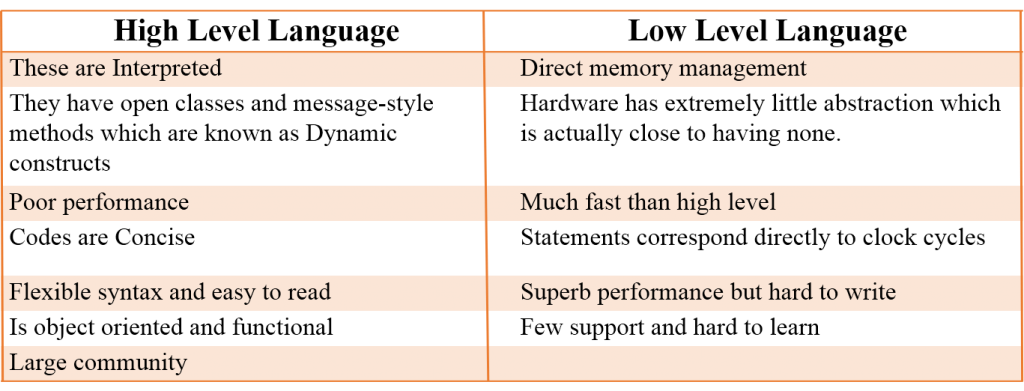
|  |  |
| --- | --- |
| CALL BY VALUE | CALL BY REFERENCE |
| Copy of actual arguments is passed to respective formal arguments. | Address of the actual arguments is passed to the formal arguments. |
| Changes made inside the function are not reflected on other functions | Any changes made in the formal arguments will also reflect in the actual arguments. |
| Actual and formal arguments will be created in different memory location. | Actual and formal arguments will be created in same memory location. |
| Original value is not modified. | Original value is modified. |
| Two copies created for the same variable which is not memory efficient. | It does not create duplicate data which is memory efficient |
| Example:  #include <stdio.h>  void swapByValue(int, int);  int main()  {  int n1 = 10, n2 = 20;  swapByValue(n1, n2);  printf("n1: %d, n2: %d\n", n1, n2);  }  void swapByValue(int a, int b)  {  int t;  t = a;  a = b;  b = t;  }    Output :  n1: 10, n2: 20 | Example:  #include <stdio.h>  void swapByReference(int\*, int  int main()  {  int n1 = 10, n2 = 20;  swapByReference(&n1, &n2);  printf("n1: %d, n2: %d\n", n1, n2);  }  void swapByReference(int \*a, int \*b)  {  int t;  t = \*a;  \*a = \*b;  \*b = t;  }    Output :  n1: 20, n2: 10 |

# What are High-level language and Low-level language

Python, C, Fortran or Pascal. Such languages are considered as high-level language because they are closer to human languages and much further from machine languages. When I say human language, I don’t mean what we talk in our day-to-day life. It means the code is something we can understand by knowing some basics in programming. The code written is almost readable by humans, something that can be read and pronounced.

Low-level languages :

Low-level languages those languages which are extremely close to machine language. They are also known as Assembly languages. The closest languages after Assembly to Machine language are C and C++. Some people even call C and C++ as low level languages. Machine code is known as low level because unlike high level programming languages it doesn’t need anything else like compilers or something. It runs directly on the processor and they are extremely architecture specific.



# 2. Compiler and Interpreter

# A compiler takes entire program and converts it into object code which is typically stored in a file. The object code is also refereed as binary code and can be directly executed by the machine after linking. Examples of compiled programming languages are C and C++.

# An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code. Examples of interpreted languages are Perl and Python.

|  |  |  |
| --- | --- | --- |
| No | Compiler | Interpreter |
| 1 | Compiler Takes Entire program as input | Interpreter Takes Single instruction as input . |
| 2 | Intermediate Object Code is Generated | No Intermediate Object Code is Generated |
| 3 | Conditional Control Statements are Executes faster | Conditional Control Statements are Executes slower |
| 4 | Memory Requirement : More(Since Object Code is Generated) | Memory Requirement is Less |
| 5 | Program need not be compiledevery time | Every time higher level program is converted into lower level program |
| 6 | Errors are displayed after entire program is checked | Errors are displayed for every instruction interpreted (if any) |
| 7 | Example : C Compiler | Example : BASIC |

1. **What are High-level language and Low-level language**

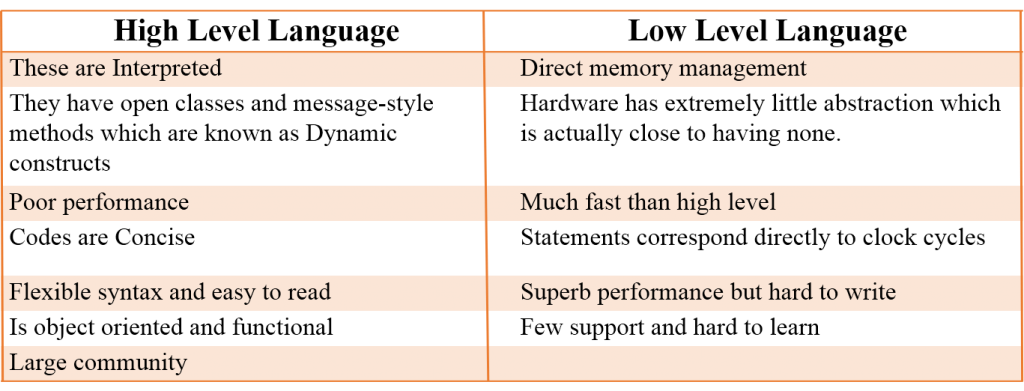
**High Level Languages:**

Python, C, Fortran or Pascal. Such languages are considered as high-level language because they are closer to human languages and much further from machine languages. When I say human language, I don’t mean what we talk in our day-to-day life. It means the code is something we can understand by knowing some basics in programming. The code written is almost readable by humans, something that can be read and pronounced.

**Low-level languages:**

Low-level languages those languages which are extremely close to machine language. They are also known as Assembly languages. The closest languages after Assembly to Machine language are C and C++. Some people even call C and C++ as low level languages. Machine code is known as low level because unlike high level programming languages it doesn’t need anything else like compilers or something. It runs directly on the processor and they are extremely architecture specific.

Characteristics of High-level and Low-level Languages:

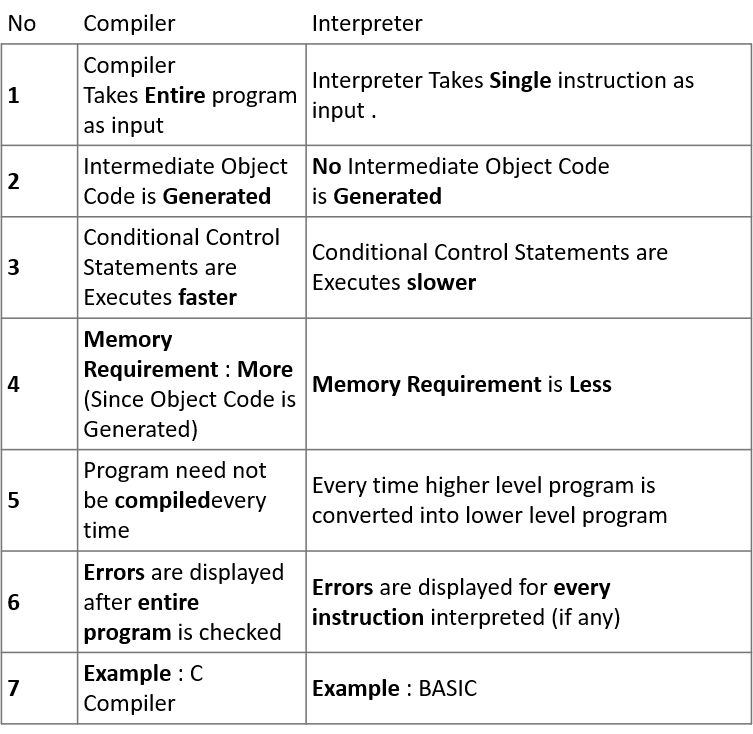
[](https://www.educba.com/academy/wp-content/uploads/2015/11/21.png)

**2.Compiler and Interpreter**

A **compiler**takes entire program and converts it into object code which is typically stored in a file. The object code is also refereed as binary code and can be directly executed by the machine after linking. Examples of compiled programming languages are C and C++.

An **Interpreter**directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code. Examples of interpreted languages are Perl and Python.

Difference between Compiler and Interpreter



Let's consider the following program

**#include <stdio.h>**

**/\*global variables\*/**

**inta,b;**

**/\*function to set values to the global variables\*/**

**void setValues(void)**

**{**

**a=100; b=200;**

**}**

**intmain()**

**{**

**/\*local variables\*/**

**intx,y;**

**x=10;**

**y=20;**

**setValues();**

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

**printf("x=%d, y=%d\n",x,y);**

**return 0;**

**}**

Output

a=100, b=200

X=10, y=20

In this program  
**Global variables** are: a and b  
**Local variables** are: x and y

**3. Rules for an Identifier**

In C programming, identifiers are names given to C entities, such as variables, functions, structures etc. Identifier are created to give unique name to C entities to identify it during the execution of program.

**Rules for writing an identifier**

\* A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.

\* The first letter of an identifier should be either a letter or an underscore. However, it is discouraged to start an identifier name with an underscore .

\* There is no rule on length of an identifier. However, the first 31 characters of identifiers are discriminated by the compiler.

**4. Difference between Local and Global variables in C.**

There are two parts of the function block :

**Declaration part** - Region where we declare all variables which are going to be used within the function.

**Executable part** - Other statements except the declarations are the executable statements.

Local variables

Local variables are the variables which are declared or defined within the declaration part of the function block.

Global variables

Global variables are the variables which are declared or defined below the header files inclusion section or before the main () function.

**5. Conditional or Ternary operators in C.**

Conditional or Ternary operators

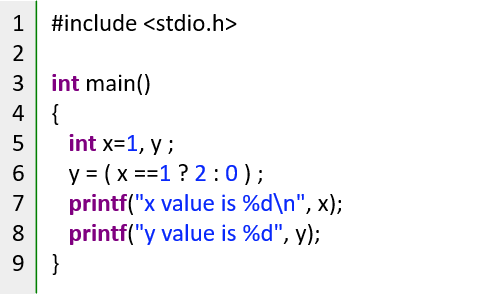
* Conditional operators return one value if condition is true and returns another value is condition is false.
* This operator is also called as ternary operator.

Syntax     :        (Condition? true\_value: false\_value);

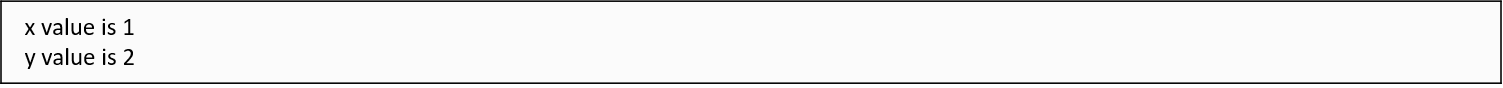
Example :         (A > 100  ?  0  :  1);

In above example, if A is greater than 100, 0 is returned else 1 is returned. This is equal to if else conditional statements.

Example program for conditional/ternary operators in c:



OUTPUT:



**6.Header Files in C Programming**

A header file in C programming language is a file with .h extension which contains a set of common function declarations and macro definitions which can be shared across multiple program files. C language provides a set of in build header files which contains commonly used utility functions and macros.

*For Example:*  
stdio.h header file contains standard Input and Output functions.

string.h header file contains string handling functions.

Types of Header Files in C

* + User defined header files.
  + In-built header files.

Syntax to Include Header File in C Program

***#include <Header\_file\_name>****/\*U*sed to include in-built system header files. *\*/*

***#include "Header\_file\_name“*** */\*U*sed to include user-defined system header files.*\*/*

**7.Macro Preprocessor**

The **C preprocessor** or **cpp** is the macro preprocessor for the C and C++ computer programming languages. The preprocessor provides the ability for the inclusion of header files, macroexpansions, conditional compilation, and line control.

Macro preprocessors are vital for processing all programs that contain macro definitions and/or calls. Language translators such as assemblers and compilers cannot directly generate the target code from the programs containing definitions and calls for macros. Therefore, most language processing activities by assemblers and compilers preprocess these programs through macro processors. A macro preprocessor essentially accepts an assembly program with macro definitions and calls as its input and processes it into an equivalent expanded assembly program with no macro definitions and calls. The macro preprocessor output program is then passed over to an assembler to generate the target object program.

**8. The Infinite Loop in C.**

A loop that repeats indefinitely and never terminates is called an **Infinite loop**.

Loops that go on executing forever and never terminates because of lack of an exit condition are called infinite loops. Sometimes we create these loops by mistake, while sometimes we deliberately create them in our program. Let’s take some examples and see what kind of mistakes can lead to infinite loop.

Example with a for loop:

**inti;**

**for (i=0;i<100;i--)**

**{**

**printf(“%d\n”,i);**

**}**

This loop is an infinite loop. Here is why ? According to condition given, the loop will execute until (i< 100). Initially, the value of i is 0 and after each iteration, its value is decremented in the update expression (i--), so the value of i will never be greater than 100. Hence the condition (i< 100) will always be true. To make it finite we should use i++ instead of i--.

**9. Return Statement.**

The return statement is used to return some value or simply pass the control to the calling function. The return statement can be used in the following two ways.

1. return;
2. Return expression

The first form of the return statement is used to terminate the function and pass the control to the calling function. No value from the called function is returned when this form of the return statement is used.

Example using return statement

**#include <stdio.h>**

**Intfactorial(int x);**

**Intmain()**

**{**

**int n;**

**printf(“Enter number : “);**

**scanf(“%d”,&n);**

**if(n<0)**

**printf(“\nfactorial is only defined for positive numbers”);**

**else**

**printf(“\n%d!=%d”,n,factorial(n)):**

**return 0;**

**}**

**intfactorial(int n)**

**{**

**if (n==0)**

**return 1:**

**int f=1,i;**

**for(i=n;i>0;i--)**

**f=f\*i;**

**return f;**

**}**

Output

Enter number :5

5!=120

**10. goto statement in C.**

A **goto** statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

Example

**#include <stdio.h>**

**int main ()**

**{**

**/\* local variable definition \*/**

**int a = 10;**

**/\* do loop execution \*/**

**LOOP:do**

**{**

**if( a == 15)**

**{**

**/\* skip the iteration \*/**

**a = a + 1;**

**goto LOOP;**

**}**

**printf("value of a: %d\n", a);**

**a++;**

**}**

**while( a< 20 );**

**return 0;**

**}**

Output

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 16

value of a: 17

value of a: 18

value of a: 19

**11. What are the differences between break and continue statements in C language?**

|  |  |
| --- | --- |
| **break** | **continue** |
| A break can appear in both switch and loop (for, while, do) statements. | A continue can appear only in loop (for, while, do) statements. |
| A break causes the switch or loop statements to terminate the moment it is executed. Loop or switch ends abruptly when break is encountered. | A continue doesn't terminate the loop, it causes the loop to go to the next iteration. All iterations of the loop are executed even if continue is encountered. The continuestatement is used to skip statements in the loop that appear after the continue. |
| The break statement can be used in both switch and loop statements. | The continue statement can appear only in loops. You will get an error if this appears in switch statement. |
| When a break statement is encountered, it terminates the block and gets the control out of the switch or loop. | When a continue statement is encountered, it gets the control to the next iteration of the loop. |

**Example using break**

**inttrim(char s[])**

**{**

**int n;**

**for (n = strlen(s)-1; n >= 0; n--)**

**if (s[n] != ' ' && s[n] != '\t' && s[n] != '\n')**

**break;**

**s[n+1] = '\0';**

**return n;**

**}**

**Example using continue**

**#include <stdio.h>**

**intmain()**

**{**

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

**inti, sum = 0;**

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

**{**

**if (a[i] < 0)**

***/\* skip negative elements \*/***

**continue;**

**sum += a[i];**

***/\* sum positive elements \*/***

**}**

**printf("Sum of positive elements: %d\n", sum);**

**}**

**OUTPUT**

**======**

**Sum of positive elements: 30**

**12. exit statement**

The C library function **void exit(int status)** terminates the calling process immediately.

Declaration

The following example shows the usage of exit() function.

**#include <stdio.h>**

**#include <stdlib.h>**

**int main ()**

**{**

**printf("Start of the program....\n");**

**printf("Exiting the program....\n");**

**exit(0);**

**printf("End of the program....\n");**

**return(0);**

**}**

Output

Start of the program....

Exiting the program....

**13.typedef in C**

typedef is a keyword used in C language to assign alternative names to existing datatypes. Its mostly used with user defined datatypes, when names of the datatypes become slightly complicated to use in programs.

For example

**#include <stdio.h>**

**#include <string.h>**

**typedef struct Books**

**{**

**char title[50];**

**char author[50];**

**char subject[100];**

**intbook\_id;**

**} Book;**

**intmain( )**

**{**

**Book book;**

**strcpy( book.title, "C Programming");**

**strcpy( book.author, "Nuha Ali");**

**strcpy( book.subject, "C Programming Tutorial");**

**book.book\_id = 6495407;**

**printf( "Book title : %s\n", book.title);**

**printf( "Book author : %s\n", book.author);**

**printf( "Book subject : %s\n", book.subject);**

**printf( "Book book\_id : %d\n", book.book\_id);**

**return 0;**

**}**

Output

Book title : C Programming

Book author :Nuha Ali

Book subject : C Programming Tutorial

|  |  |
| --- | --- |
| Book book\_id : 6495407 |  |

**1. Differentiate between machine language, assembly language and high level languages? What is the difference between compiler and assembler? (5)**

**Machine Language:** Each computer can only understand programs written in its own ML is provided by the computer manufacturer. Translation is needed when executing programs written in Pascal or BASIC. Written at the most basic level of computer operation.

**Assembly language**: Also provided by the manufacturer. One instruction for each computer operation. Instruction codes are represented by mnemonics. The code must be assembled into machine language for execution.

**High level languages:** made programming much more convenient. Written using common names and words, usually human languages. Problem oriented languages ie designed for specific problems.

**Difference Between Compiler And Assembler**

The compiler is a simple program which converts the source code written by the humans to a machine language. While the assembler has a little bit different work, it converts the assembly language to the machine language.

Compilers work more directly than the assemblers. The compilers can convert the human written code in the machine language directly. But the assembler can’t do this at once. It converts a source code to an object code first then it converts the object code to the machine language with the help of the linker programs.

**2. Write a C program to find the LCM and HCF of any two numbers entered by user. (5**

#include<conio.h>

void main()

{

int a, b, x, y, t, hcf, lcm;

printf("Enter two number : ");

scanf("%d%d",&x, &y);

a=x;

b=y;

while(b!=0)

{

t=b;

b=a%b;

a=t;

}

hcf=a;

lcm=(x\*y)/hcf;

printf("HCF = %d\n",hcf);

printf("LCM = %d",lcm);

getch();

}

**3. Write a C program to reverse a string. (5)**

#include <stdio.h>

int main()

{

char s[1000], r[1000];

int begin, end, count = 0;

printf("Input a string**\n**");

gets(s);

  while (s[count] != '**\0**')

count++;

  end = count - 1;

  for (begin = 0; begin < count; begin++)

{

r[begin] = s[end];

end--;

}

  r[begin] = '**\0**';

  printf("%s**\n**", r);

  return 0;

}

**4. What are the 4 basic storage classes in C? (5)**

Storage class in C decides the part of storage to allocate memory for a variable, it also determines the scope of a variable. All variables defined in a C program get some physical location in memory where variable's value is stored. Memory and CPU registers are types of memory locations where a variable's value can be stored. The storage class of a variable in C determines the life time of the variable if this is 'global' or 'local'. Along with the life time of a variable, storage class also determines variable's storage location (memory or registers), the scope (visibility level) of the variable, and the initial value of the variable. There are four storage classes in C those are *automatic*, *register*, *static*, and *external*

There are four storage classes in C they are as follows:

1. Automatic Storage Class
2. Register Storage Class
3. Static Storage Class
4. External Storage Class

Now, let us discuss these storage classes one by one.

**Automatic Storage Class**

A variable defined within a function or block with auto specifier belongs to automatic storage class. All variables defined within a function or block by default belong to automatic storage class if no storage class is mentioned. Variables having automatic storage class are local to the block which they are defined in, and get destroyed on exit from the block.

**Register Storage Class**

The register specifier declares a variable of register storage class. Variables belonging to register storage class are local to the block which they are defined in, and get destroyed on exit from the block. A registerdeclaration is equivalent to an auto declaration, but hints that the declared variable will be accessed frequently; therefore they are placed in CPU registers, not in memory. Only a few variables are actually placed into registers, and only certain types are eligible; the restrictions are implementation-dependent. However, if a variable is declared register, the unary & (address of) operator may not be applied to it, explicitly or implicitly. Register variables are also given no initial value by the compiler.

**Static Storage Class**

The static specifier gives the declared variable static storage class. Static variables can be used within function or file.Unlike global variables, static variables are not visible outside their function or file, but they maintain their values between calls. The static specifier has different effects upon local and global variables. See the following flavours of static specifier.

When static specifier is applied to a local variable inside a function or block, the compiler creates permanent storage for it, much as it creates storage for a global variable but static local variable remains visible only to the function or block in which it is defined. In simple terms, a static local variable is a local variable that retains its value between function calls. For example, the following program code defines static variable i at two places in two blocks inside function staticDemo(). Function staticDemo() is called twice within from main function. During second call static variables retain their old values and they are not initialized again in second call of staticDemo().

When static specifier is applied to a global variable or a function then compiler makes that variable or function known only to the file in which it is defined. A static global variable has *internal linkage* that means even though the variable is global; routines in other files have no knowledge of it and cannot access and alter its contents directly.

The following C program defines one static global variable gInt and a static function staticDemo(), for the variable and function are defined static they cannot be used outside the file (translation unit) staticdemo.c..

**External Storage Class**

The extern specifier gives the declared variable external storage class. The principal use of extern is to specify that a variable is declared with external linkage elsewhere in the program. To understand why this is important, it is necessary to understand the difference between a declaration and a definition. A declaration declares the name and type of a variable or function. A definition causes storage to be allocated for the variable or the body of the function to be defined. The same variable or function may have many declarations, but there can be only one definition for that variable or function.

When extern specifier is used with a variable declaration then no storage is allocated to that variable and it is assumed that the variable has already been defined elsewhere in the program. When we use extern specifier the variable cannot be initialized because with extern specifier variable is declared, not defined.

**5. Differentiate between structure and union. (5)**

A structure is a user-defined data type available in C that allows to combining data items of different kinds. Structures are used to represent a record.  
**Defining a structure:** To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows:

struct [structure name]

{

member definition;

member definition;

...

member definition;

};

A union is a special data type available in C that allows storing different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple purposes.  
**Defining a Union:** To define a union, you must use the **union** statement in the same way as you did while defining a structure. The union statement defines a new data type with more than one member for your program. The format of the union statement is as follows:

union [union name]

{

member definition;

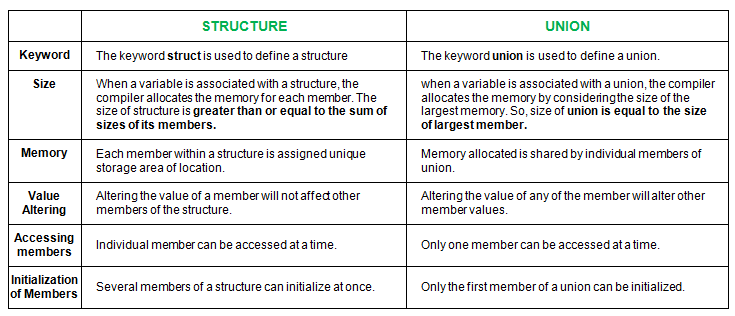
member definition;

...

member definition;

};

**Differences**



**6. Write a C program to add two variables using pointers. (5)**

#include <stdio.h>

int main()

{

int first, second, \*p, \*q, sum;

printf("Enter two integers to add**\n**");

scanf("%d%d", &first, &second);

p = &first;

q = &second;

sum = \*p + \*q;

printf("Sum of entered numbers = %d**\n**",sum);

return 0;

}

**7. Explain how a new file is opened. What are the 3 modes while opening an existing file. (5)**

A file represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data.

**fopen():**The fopen() function is used to open a file and associates an I/O stream with it. Thisfunction takes two arguments. The first argument is a pointer to a string containing name of the file to be opened while the second argument is the mode in which the file is to be opened.

**File \*fopen(const char \*path, const char \*mode);**

The three file opening modes are read, write and append.

* ‘r’ : Open text file for reading. The stream is positioned at the beginning of the file.
* ‘w’ : Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
* ‘a’ : Open for appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.

**8. Write a Python program to reverse a given integer. (5)**

n=int(input("Enter number: "))

rev=0

**while**(n>0):

dig=n%10

rev=rev\*10+dig

n=n//10

**print**("Reverse of the number:",rev)

PART B

**9 a) . Explain any five kinds of operators in C. (5)**

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions.

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Bitwise Operators
* Assignment Operators
* Special Operators

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_arithmetic_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Adds two operands. | A + B = 30 |
| − | Subtracts second operand from the first. | A − B = -10 |
| \* | Multiplies both operands. | A \* B = 200 |
| / | Divides numerator by de-numerator. | B / A = 2 |
| % | Modulus Operator and remainder of after an integer division. | B % A = 0 |
| ++ | Increment operator increases the integer value by one. | A++ = 11 |
| -- | Decrement operator decreases the integer value by one. | A-- = 9 |

**Relational Operators**

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_relational_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | Checks if the values of two operands are equal or not. If yes, then the condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true. | (A <= B) is true. |

**Logical Operators**

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_logical_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. | (A && B) is false. |
| || | Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true. | (A || B) is true. |
| ! | Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false. | !(A && B) is true. |

**Bitwise Operators**

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **p** | **q** | **p & q** | **p | q** | **p ^ q** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100 B = 0000 1101

-----------------

A&B = 0000 1100 , A|B = 0011 1101 , A^B = 0011 0001, ~A = 1100 0011

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

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_bitwise_operators.htm)

|  |  |  |
| --- | --- | --- |
| **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 |

**Assignment Operators**

The following table lists the assignment operators supported by the C language −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_assignment_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Simple assignment operator. Assigns values from right side operands to left side operand | C = A + B will assign the value of A + B to C |
| += | Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand. | C -= A is equivalent to C = C - A |
| \*= | Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand. | C \*= A is equivalent to C = C \* A |
| /= | Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand. | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand. | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator. | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator. | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator. | C &= 2 is same as C = C & 2 |
| ^= | Bitwise exclusive OR and assignment operator. | C ^= 2 is same as C = C ^ 2 |
| |= | Bitwise inclusive OR and assignment operator. | C |= 2 is same as C = C | 2 |

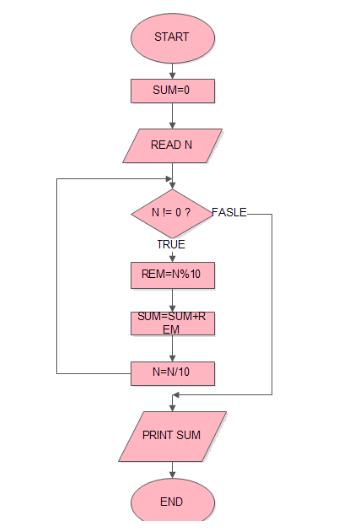
**Special Operators ↦ sizeof & ternary**

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_sizeof_operator.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| sizeof() | Returns the size of a variable. | sizeof(a), where a is integer, will return 4. |
| & | Returns the address of a variable. | &a; returns the actual address of the variable. |
| \* | Pointer to a variable. | \*a; |
| ? : | Conditional Expression. | If Condition is true ? then value X : otherwise value Y |

**b) Draw a flowchart to find the sum of digits of an integer. (5)**



10 a) If more than one kind of operator is present in an expression, explain the order of precedence. (5)

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 \* 2; here, x is assigned 13, not 20 because operator \* has a higher precedence than +, so it first gets multiplied with 3\*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

|  |  |  |
| --- | --- | --- |
| **Category** | **Operator** | **Associativity** |
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)\* & sizeof | Right to left |
| Multiplicative | \* / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | | | Left to right |
| Logical AND | && | Left to right |
| Logical OR | || | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= \*= /= %=>>= <<= &= ^= |= | Right to left |
| Comma | , | Left to right |

b) Write a C program to print the prime numbers between 101 and 500. Those numbers whose sum of digits is 5 need not be printed. Use ‘while’ loop in the program (5).

#include <stdio.h>

int main()

{

int low = 101, high =500, i, d,sum,num,flag;

while (low < high)

{

flag = 0;

for(i = 2; i <= low/2; ++i)

{

if(low % i == 0)

{

flag = 1;

break;

}

}

if (flag == 0)

{

num=low;

while (num >0){

d=num%10;

sum=sum+d;

num=num/10;

}

If (sum > 5)

printf("%d ", low);

}

++low;

}

**11 a) Explain ‘switch’ and ‘go to’ statements in C with the help of examples (5)**

The **switch case statement** is used when we have multiple options and we need to perform a different task for each option.

switch (variable or an integer expression)

{

case constant:

//C Statements

;

case constant:

//C Statements

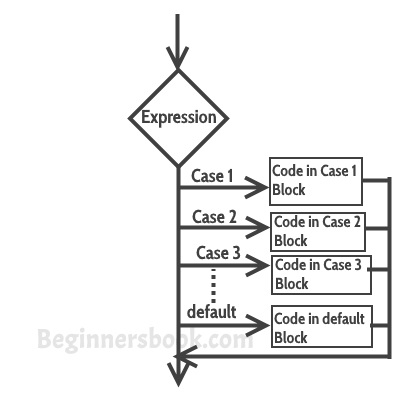
;

default:

//C Statements

;

}



### Example of Switch Case in C

#include <stdio.h>

int main()

{

int num=2;

switch(num+2)

{

case 1:

printf("Case1: Value is: %d", num);

case 2:

printf("Case1: Value is: %d", num);

case 3:

printf("Case1: Value is: %d", num);

default:

printf("Default: Value is: %d", num);

}

return 0;

}

**Output:** Default: value is: 2

**Explanation:** In switch I gave an expression, you can give variable also. I gave num+2, where num value is 2 and after addition the expression resulted 4. Since there is no case defined with value 4 the default case is executed.

**goto statement** The goto statement is rarely used because it makes program confusing, less readable and complex. Also, when this is used, the control of the program won’t be easy to trace, hence it makes testing and debugging difficult.When a goto statement is encountered in a C program, the control jumps directly to the label mentioned in the goto stateemnt  
**Syntax of goto statement in C**

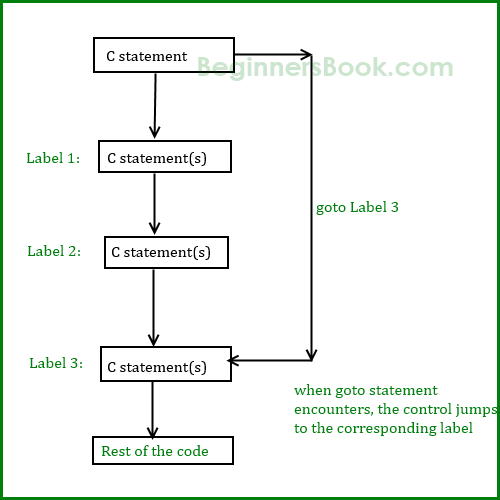
goto label\_name;

..

..

label\_name: C-statements

### Flow Diagram of goto



### Example of goto statement

#include <stdio.h>

int main()

{

int sum=0;

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

sum = sum+i;

if(i==5){

goto addition;

}

}

addition:

printf("%d", sum);

return 0;

}

Output: 15

**Explanation:** In this example, we have a label addition and when the value of i (inside loop) is equal to 5 then we are jumping to this label using goto. This is reason the sum is displaying the sum of numbers till 5 even though the loop is set to run from 0 to 10.

**b) Write a C program to print the following pattern using ‘for’ loop (5)**

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

#include <stdio.h>

#include <conio.h>

void main() {

int i,j,k;

clrscr();

for (i=1; i<=5; i++) {

for (j=5; j>=i; j--) {

printf(" ");

}

for (k=1; k<=i; k++) {

printf("\*");

}

printf("\n");

}

getch();

}

PART C

12 a) Explain how a 3 dimensional array is declared and initialized. How is a character

array different from a string (5)

Declare a three-dimensional (3d) array.

Data type array name[size][size][size] ;Eg : float y[2][4][3];

Initializing a three dimentaional array

int arr[2][3][4] = { { {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4} },

{ {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4} } };

| **BASIS FOR COMPARISON** | **CHARACTER ARRAY** | **STRING** |
| --- | --- | --- |
| Basic | Character array is collection of variables, of character data type. | String is class and variables of string are the object of class "string". |
| Syntax | char array\_name [size]; | string string\_name; |
| Indexing | An individual character in a character array can be accessed by its index in array. | In string the particular character can be accessed by the function "string\_name.charAt(index)". |
| Data Type | A character array does not define a datatype. | A string defines a datatype in C++. |
| Operators | Operators in C++ can not be applied on character array. | You may apply standard C++ operator on the string. |
| Boundary | Array boundaries are easily overrun. | Boundaries will not overrun. |
| Access | Fast accessing. | Slow accessing. |

b) Write a C program to count the number of characters, words and lines in a text (5)

#include <stdio.h>

int main(){

char ch;

unsigned int long linecount, wordcount, charcount;

int u;

linecount=0;

wordcount=0;

charcount=0;

while((ch=getc(stdin))!=EOF){

if (ch !='\n') {++charcount;}

if (ch==' ' || ch=='\n') {++wordcount;}

if (ch=='\n') {++linecount;}

}

if(charcount>0){

++wordcount;

++linecount;

}

printf( "%lu %lu %lu\n", charcount, wordcount, linecount );

return 0;

}

**13 a) Write a C program to find the product of two matrices (5)**

#include <stdio.h>

int main()

{

int m, n, p, q, c, d, k, sum = 0;

int first[10][10], second[10][10], multiply[10][10];

printf("Enter the number of rows and columns of first matrix**\n**");

scanf("%d%d", &m, &n);

printf("Enter the elements of first matrix**\n**");

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

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

scanf("%d", &first[c][d]);

printf("Enter the number of rows and columns of second matrix**\n**");

scanf("%d%d", &p, &q);

if (n != p)

printf("Matrices with entered orders can't be multiplied with each other.**\n**");

else

{

printf("Enter the elements of second matrix**\n**");

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

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

scanf("%d", &second[c][d]);

for (c = 0; c < m; c++) {

for (d = 0; d < q; d++) {

for (k = 0; k < p; k++) {

sum = sum + first[c][k]\*second[k][d];

}

multiply[c][d] = sum;

sum = 0;

}

}

printf("Product of entered matrices:-**\n**");

for (c = 0; c < m; c++) {

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

printf("%d**\t**", multiply[c][d]);

printf("**\n**");

}

}

return 0;

}

**b) Explain the differences between pass by value and pass by reference with the help of examples. (5)**

In c we can pass the parameters in a function in two different ways.

**Pass by value**: In this approach we pass copy of actual variables in function as a parameter. Hence any modification on parameters inside the function will not reflect in the actual variable. For example:

#include<stdio.h>

int main(){

    int a=5,b=10;

    swap(a,b);

    printf("%d      %d",a,b);

    return 0;

}

void swap(int a,int b){

    int temp;

    temp =a;

    a=b;

    b=temp;

}

Output: 5    10

**Pass by reference**: In this approach we pass memory address actual variables in function as a parameter. Hence any modification on parameters inside the function will reflect in the actual variable. For example:

#incude<stdio.h>

int main(){

    int a=5,b=10;

    swap(&a,&b);

    printf("%d %d",a,b);

    return 0;

}

void swap(int \*a,int \*b){

    int  \*temp;

    \*temp =\*a;

    \*a=\*b;

    \*b=\*temp;

}

Output: 10 5

**14 a) Explain recursive function with the help of an example program. (5)**

Recursion- is a process of function calling itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

|  |  |
| --- | --- |
| #include<stdio.h>  void Foo(int n)  { printf(“%d”,n);  if(n==0)  Return 0;  else  Foo(n-1);}  int main()\{  Foo(10);  } | This function Foo is used to print n numbers in descending order. Here first time when we call Foo it will print the value of n. Then check the condition if the condition is true the execution stops, other wise it will call Foo again with n-1 as the argument.. |

**b) Write a program to print all prime numbers between any two numbers entered by user, using functions. (5)**

#include <stdio.h>

int checkPrimeNumber(int n);

int main()

{

int n1, n2, i, flag;

printf("Enter two positive integers: ");

scanf("%d %d", &n1, &n2);

printf("Prime numbers between %d and %d are: ", n1, n2);

for(i=n1+1; i<n2; ++i) {

// i is a prime number, flag will be equal to 1

flag = checkPrimeNumber(i);

if(flag == 1)

printf("%d ",i);

}

return 0;

}

// user-defined function to check prime number

int checkPrimeNumber(int n)

{

int j, flag = 1;

for(j=2; j <= n/2; ++j)

{

if (n%j == 0)

{

flag =0;

break;

}

}

return flag;

}

**15 a) What is a pointer. How can you access a variable using chain of pointers? (5)**

**b) Write a C program to sort an array using pointers. (5)**

#include <stdio.h>

void main()

{

int \*a,i,j,tmp,n;

printf("\n\n Pointer : Sort an array using pointer :\n");

printf("--------------------------------------------\n");

printf(" Input the number of elements to store in the array : ");

scanf("%d",&n);

printf(" Input %d number of elements in the array : \n",n);

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

{

printf(" element - %d : ",i+1);

scanf("%d",a+i);

}

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

{

for(j=i+1;j<n;j++)

{

if( \*(a+i) > \*(a+j))

{

tmp = \*(a+i);

\*(a+i) = \*(a+j);

\*(a+j) = tmp;

}

}

}

printf("\n The elements in the array after sorting : \n");

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

{ printf(" element - %d : %d \n",i+1,\*(a+i));

}

printf("\n");

}

**16 a) Write a C program to store the name and roll numbers of 10 students using structure. The name has to be then printed in the ascending order of roll numbers. (5)**

**b) Explain the basic control statements in Python. (5)**

**17 a) Write a C program to copy the contents of one file to another. (5)**

#include <stdio.h>

#include <stdlib.h> // For exit()

 int main()

{

    FILE \*fptr1, \*fptr2;

    char filename[100], c;

     printf("Enter the filename to open for reading \n");

    scanf("%s", filename);

     // Open one file for reading

    fptr1 = fopen(filename, "r");

    if (fptr1 == NULL)

    {

        printf("Cannot open file %s \n", filename);

        exit(0);

    }

     printf("Enter the filename to open for writing \n");

    scanf("%s", filename);

     // Open another file for writing

    fptr2 = fopen(filename, "w");

    if (fptr2 == NULL)

    {

        printf("Cannot open file %s \n", filename);

        exit(0);

    }

     // Read contents from file

    c = fgetc(fptr1);

    while (c != EOF)

    {

        fputc(c, fptr2);

        c = fgetc(fptr1);

    }

     printf("\nContents copied to %s", filename);

     fclose(fptr1);

    fclose(fptr2);

    return 0;

}

**b) How can a random part in a file be accessed? (5)**

There is no need to read each record sequentially, if we want to access a particular record.C supports these functions for random access file processing.

1. fseek()
2. ftell()
3. rewind()

**fseek():**  
This function is used for seeking the pointer position in the file at the specified byte.  
**Syntax:** fseek( file pointer, displacement, pointer position);  
Where  
**file pointer ----** It is the pointer which points to the file.  
**displacement ----** It is positive or negative.This is the number of bytes which are skipped backward (if negative) or forward( if positive) from the current position.This is attached with L because this is a long integer.  
  
**Pointer position:**  
This sets the pointer position in the file.

Value pointer position

0 Beginning of file.

1 Current position

2 End of file

#### Ex:

1) fseek( p,10L,0)  
0 means pointer position is on beginning of the file,from this statement pointer position is skipped 10 bytes from the beginning of the file.  
2)fseek( p,5L,1)  
1 means current position of the pointer position.From this statement pointer position is skipped 5 bytes forward from the current position.  
3)fseek(p,-5L,1)  
From this statement pointer position is skipped 5 bytes backward from the current position.

#### ftell()

This function returns the value of the current pointer position in the file.The value is count from the beginning of the file.  
**Syntax:** ftell(fptr);  
Where fptr is a file pointer.

#### rewind()

This function is used to move the file pointer to the beginning of the given file.  
**Syntax:** rewind( fptr);  
Where fptr is a file pointer.

1. **a) Discuss various data types used in C with examples**
2. char – a single byte can hold one character

an character variable is declared and initialized as char a =’x’;

1. int – integer is a data type used to represent integer values. We can declare a variable as an integer type or define the result of a function or expression as of integer type. The amount of space occupied by the integer data type depends on the computer system you are using - typical ranging from 4 bytes to over 64 bytes.  The different variations of integer datatypes are

an integer variable is declared and initialized as int a = 5;

1. float – a single precision floating point number
2. double - a double precision floating point number

**b) What do you understand by the term Keyword?**

**Keywords:** each word n c programming is either keyword or an identifier.Keywords have predefined meaning and cannot be changed by the user. For example

i. auto ii. else iii. double iv. float v. Break

**2. With suitable example discuss the use break and continue statements.**

**break Statement:** A break statement terminates the execution of the loop and thecontrol is transferred to the immediately following loop.

Eg:

while(1)

{

scanf (“%d”,&i);

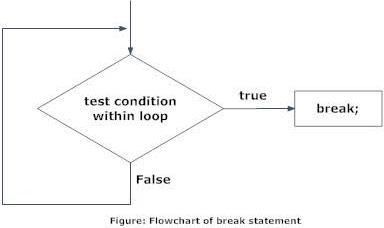
if (i==-1)

break;

sum+=i;

num++;

}



1. **continue Statement:** is used to bypass the remainder of the current pass through a loop.The loop does not terminate when a continue statement is encountered. instead the remaining loop statements are skipped and the computation directly proceeds to next pass through loop. Eg.

void main();

{

int i=1,num, sum=0;

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

{

printf(“ Enter an integer”);

scanf(“%d”,&num);

if(num<0)

{

puts;

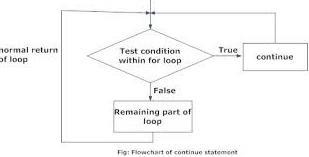
continue;

}

sum+=num;

}

printf(“ the sum of the positive integers entered =%i”,sum);



}

**xplanation with suitable examples: 3Marks**

**Flowchart: 1 mark each (2 Marks)**

1. What do you mean by arrays? How they are initialized with declaration.

**Arrays:** is a sequence of data in memory, wherein all data are of the same type and areplaced in physically adjacent location.

**Declaration**

Data type of Array Name of Array [size of Array]

For eg. int age[5]

This statement states that age is an array of 5 integers

**Initialization**

int Array[4]={1,2,3,4}

**Explanation - 3marks**

**Declaration and Initialization with example – 2marks**

**4**.With suitable example explain what do you understand by recursion.

Recursion- is a process of function calling itself.

#include<stdio.h>

void Foo(int n)

{

printf(“%d”,n);

if(n==0)

Return 0;

else

Foo(n-1);}

int main()\{

Foo(10);

}

Explanation – 2.5Marks

Example – 2.5Marks

1. Distinguish between structures and unions with an example

a. The keyword “struct” is used to define Structure.The keyword “union” is used to define union

b. The amount of memory required to store a structure variable is the sum of size of all the members.On the other hand in case of unions it is always equal to that required by its largest member

c. In structures each member have their own memory space.In union one block is used by all the members of the union.

d. Both are container data type and can contain objects of any type, including other structures and unions or arrays as their members.

e. In structures individual member can be accessed at a time.

In union only one member can be accessed at a time.

f. Several members of a structure can initialize at once.Only the first member of a union can be initialized

**Any 5points 5Marks**

**6**. Describe pointer variables

A pointer is a [programming language](https://en.wikipedia.org/wiki/Programming_language) object, whose value refers to (or "points to") another value stored elsewhere in the [computer memory](https://en.wikipedia.org/wiki/Computer_memory) using its [memory address.](https://en.wikipedia.org/wiki/Memory_address) A pointer references a location in memory, and obtaining the value stored at that location is known as [dereferencing](https://en.wikipedia.org/wiki/Dereference_operator) the pointer.

**Declaration**

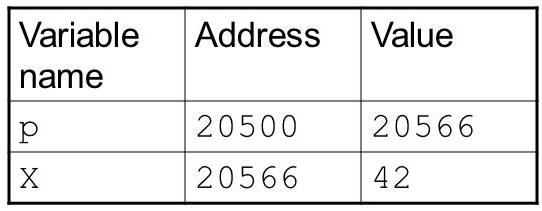
data\_type \*pointer name;

Pointer Initialization is the process of assigning address of a variable to pointer variable. Pointer variable contains address of variable of same data type. In C language address operator & is used to determine the address of a variable. The & (immediately preceding a variable name) returns the address of the variable associated with it.

int \*p;

int x=42;

p=&x;



**Explanation with suitable example – 5marks**

1. Explain any 3 file handling operations in C programming

A file represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data.

**fopen():**The fopen() function is used to open a file and associates an I/O stream with it. Thisfunction takes two arguments. The first argument is a pointer to a string containing name of the file to be opened while the second argument is the mode in which the file is to be opened.

**File \*fopen(const char \*path, const char \*mode);**

* r’ : Open text file for reading. The stream is positioned at the beginning of the file.
* ‘r+’ : Open for reading and writing. The stream is positioned at the beginning of the file.
* ‘w’ : Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
* ‘w+’ : Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.
* ‘a’ : Open for appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.
* ‘a+’ : Open for reading and appending (writing at end of file). The file is created if it does not exist. The initial file position for reading is at the beginning of the file, but output is always appended to the end of the fil

**fread() and fwrite():**The functions fread/fwrite are used for reading/writing data from/to thefile opened by fopen function. These functions accept three arguments. The first argument is a pointer to buffer used for reading/writing the data. The data read/written is in the form of ‘nmemb’ elements each ‘size’ bytes long.

**size\_t fread(void \*ptr, size\_t size, size\_t nmemb, FILE \*stream);**

**size\_t fwrite(const void \*ptr, size\_t size, size\_t nmemb, FILE \*stream);**

**fseek()**

**fclose()**

**Any two with explanation 2.5 each 5 marks**

**PART B** (**ANSWER ANY TWO**)

**10**. Write the algorithm and draw the flowchart to find the average height of boys andgirls in a class from a given set of student data

Step 1: Initialize counters t accumulate totals and store zeros in them

Step 2: Repeat steps 3 and 4 until end of slips reached

Step 3: Read a slip

Step 4: if gender code =1

Total boy height = Total boy height+ Height

Total boys= Total boys+1

Else

Total girl height = Total girl height+ Height

Total girls= Total girls+1

Step 5: Average boy height = Total boy height / Total Boys

Average girl height = Total girl height / Total girls

Step 6: Print Total Boys, Average boy Height

Total Girls, Average girl Height

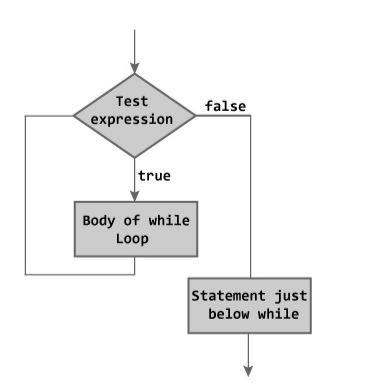
Step 7: Stop

1. Discuss While, Do while and For statement using suitable examples

**while Loop:** the while loop is often used when a number of times the loop is to be executedis not known in advance. The syntax of a while loop is:

while(test expression)

{ //codes}



Flow chart for while Loop

#include<stdio.h>

int main()

{ int n; long fact;

printf(“Enteran integer”); scanf(“%d”,&n); fact=1;

while(n>0)

{fact=\*n; --n;}

Printf(“factorial=%d”,fact);

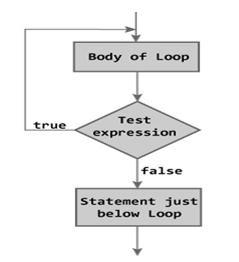
Return 0;}

**do while Loop:** The do..while loop is similar to the while loop with one important difference.The body of do...while loop is executed once, before checking the test expression. Hence, the do...while loop is executed at least once. Syntax for do while is

do

{codes}

while(test expression);



include<stdio.h>

int main()

{ double n, sum=0; Do

{

printf(“Enteran integer”); scanf(“%lf”,&n); sum+=n;

}

while(n!=0)

Printf(“sum=%lf”,sum); Return 0;}

12. Write a C program to sort the values of an array in descending order

#include <stdio.h>

int main()

{

int arr[100];

int size, i, j, temp;

printf("Enter size of array: ");

scanf("%d", &size);

printf("Enter elements in array: ");

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

{

scanf("%d", &arr[i]);

}

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

{

for(j=i+1; j<size; j++)

{

if(arr[i] < arr[j])

{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}}

printf("\nElements of array in sorted descending order: ");

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

{

printf("%d\t", arr[i]);

}

return 0;}

13. Write a C program to find the transpose of a matrix

#include<stdio.h>

#include<conio.h>

void main()

{int A[2][3] , B[3][2];

int i, j;

clrscr();

printf(" Enter the elements of A\n");

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

{

for(j=0 ; j<3 ; j++)

{

scanf("%d" , &A[i][j] );

}}

printf(" Matrix is\n");

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

{

for(j=0 ; j<3 ; j++)

{

printf("%d\t" , A[i][j] );

}

printf("\n");

}

for(i=0 ; i<3 ; i++)

{

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

{B[i][j] = A[j][i];

}

}

printf(" After Transpose\n");

for(i=0 ; i<3 ; i++)

{

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

{

printf("%d\t" , B[i][j] );

}

printf("\n");

}getch();}

14. a. With proper examples explain the storage classes in C

Storage class in C decides the part of storage to allocate memory for a variable, it also determines the scope of a variable. All variables defined in a C program get some physical location in memory where variable's value is stored. Memory and CPU registers are types of memory locations where a variable's value can be stored. The storage class of a variable in C determines the life time of the variable if this is 'global' or 'local'. Along with the life time of a variable, storage class also determines variable's storage location (memory or registers), the scope (visibility level) of the variable, and the initial value of the variable. There are four storage classes in C those are *automatic*, *register*, *static*, and *external*

There are four storage classes in C they are as follows:

* Automatic Storage Class
* Register Storage Class
* Static Storage Class
* External Storage Class

Now, let us discuss these storage classes one by one.

**1. Automatic Storage Class**

A variable defined within a function or block with auto specifier belongs to automatic storage class. All variables defined within a function or block by default belong to automatic storage class if no storage class is mentioned. Variables having automatic storage class are local to the block which they are defined in, and get destroyed on exit from the block.

**2. Register Storage Class**

The register specifier declares a variable of register storage class. Variables belonging to register storage class are local to the block which they are defined in, and get destroyed on exit from the block. A registerdeclaration is equivalent to an auto declaration, but hints that the declared variable will be accessed frequently; therefore they are placed in CPU registers, not in memory. Only a few variables are actually placed into registers, and only certain types are eligible; the restrictions are implementation-dependent. However, if a variable is declared register, the unary & (address of) operator may not be applied to it, explicitly or implicitly. Register variables are also given no initial value by the compiler.

**3. Static Storage Class**

The static specifier gives the declared variable static storage class. Static variables can be used within function or file.Unlike global variables, static variables are not visible outside their function or file, but they maintain their values between calls. The static specifier has different effects upon local and global variables. See the following flavours of static specifier.

* When static specifier is applied to a local variable inside a function or block, the compiler creates permanent storage for it, much as it creates storage for a global variable but static local variable remains visible only to the function or block in which it is

defined. In simple terms, a static local variable is a local variable that retains its value between function calls. For example, the following program code defines static variable i at two places in two blocks inside function staticDemo(). Function staticDemo() is called twice within from main function. During second call static variables retain their old values and they are not initialized again in second call of staticDemo().

* When static specifier is applied to a global variable or a function then compiler makes that variable or function known only to the file in which it is defined. A static global variable has *internal linkage* that means even though the variable is global; routines in other files have no knowledge of it and cannot access and alter its contents directly.

The following C program defines one static global variable gInt and a static function staticDemo(), for the variable and function are defined static they cannot be used outside the file (translation unit) staticdemo.c..

* **4. External Storage Class**
* The extern specifier gives the declared variable external storage class. The principal use of extern is to specify that a variable is declared with external linkage elsewhere in the program. To understand why this is important, it is necessary to understand the difference between a declaration and a definition. A declaration declares the name and type of a variable or function. A definition causes storage to be allocated for the variable or the body of the function to be defined. The same variable or function may have many declarations, but there can be only one definition for that variable or function.
* When extern specifier is used with a variable declaration then no storage is allocated to that variable and it is assumed that the variable has already been defined elsewhere in the program. When we use extern specifier the variable cannot be initialized because with extern specifier variable is declared, not defined.

b. Differentiate the user defined and library functions.

Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library.

Another type is called a user-defined function. This is a function which the programmer creates and uses in a C program. User has the freedom to choose function name nad return data type

Each library function in C performs specific operation.

We can make use of these library functions to get the pre-defined output instead of writing our own code to get those outputs.

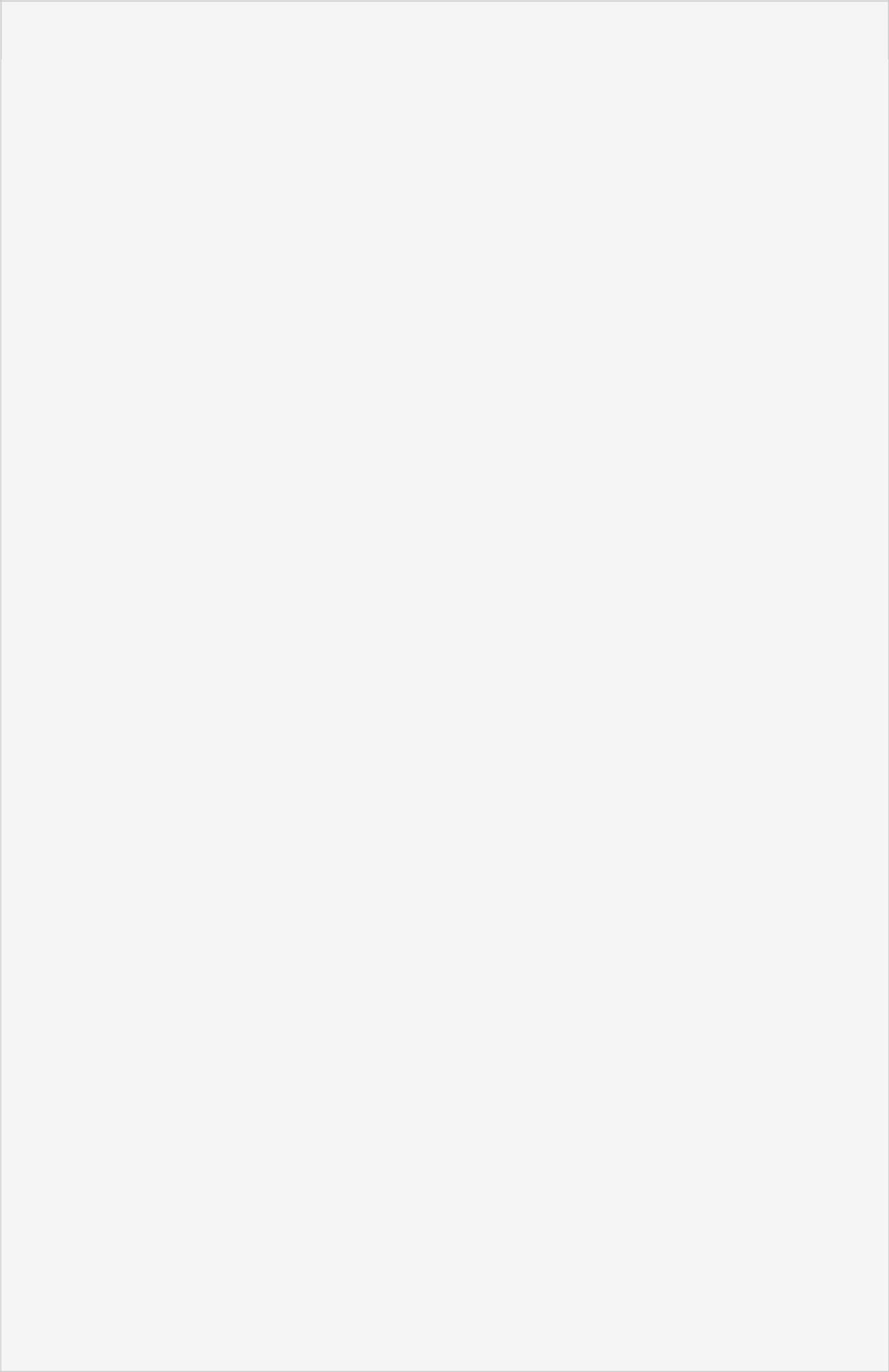
These library functions are created by the persons who designed and created C compilers.

All C standard library functions are declared in many header files which are saved as file\_name.h.

Actually, function declaration, definition for macros are given in all header files.

We are including these header files in our C program using “#include<file\_name.h>” command to make use of the functions those are declared in the header files.

When we include header files in our C program using “#include<filename.h>” command, all C code of the header files are included in C program. Then, this C program is compiled by compiler and executed

1. Write a C program to perform the file handling operation to read series integer number and write all odd no to a file to be called ODD and even numbers to EVEN.

#include<stdio.h>

#include<conio.h>

void main()

{FILE \*fptr1, \*fptr2, \*fptr3;

int n, i, num;

clrscr();

printf("Enter number of values : ");

scanf("%d", &n);

printf("\nEnter the values : ");

fptr1 = fopen("NUMBERS.DAT", "w");

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

{scanf("%d", &num);

putw(num, fptr1); }

fclose(fptr1);

fptr1 = fopen("NUMBERS.DAT", "r");

fptr2 = fopen("ODD.DAT", "w");

fptr3 = fopen("EVEN.DAT", "w");

while((num = getw(fptr1)) != EOF)

{if(num % 2 == 0)

putw(num, fptr3) ; }

else{

putw(num, fptr2)}}

fclose(fptr1);

fclose(fptr2);

fclose(fptr3);

fptr2 = fopen("ODD.DAT", "r");

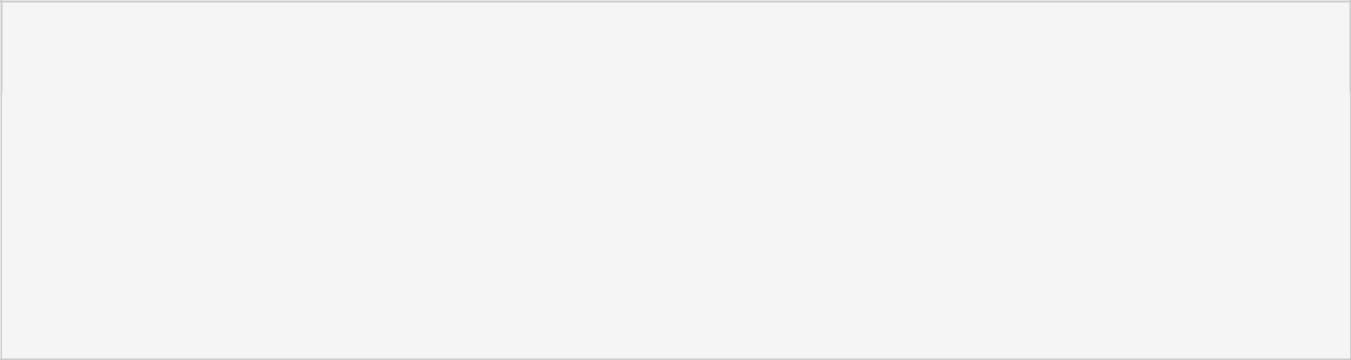
fptr3 = fopen("EVEN.DAT", "r");

printf("\nContents of ODD file is : ");

while((num = getw(fptr2)) != EOF)

{printf("%d\t", num) ;}

printf("\n\nContents of EVEN file is : ") ;

while((num = getw(fptr3)) != EOF)

{printf("%d\t", num);}

fclose(fptr2);

fclose(fptr3);

getch();}

16. Using functions write a program to swap the contents of two memory locations.

#include<stdio.h>

#include<conio.h>

void swap(int \*num1, int \*num2);

void main()

{ int x, y;

printf("\nEnter First number : ");

scanf("%d", &x);

printf("\nEnter Second number : ");

scanf("%d", &y); printf("\nBefore Swaping x = %d and y = %d", x, y); swap(&x, &y);

printf("\nAfter Swaping x = %d and y = %d", x, y); getch();}

void swap(int \*num1, int \*num2)

{int temp;

temp = \*num1;

\*num1 = \*num2;

\*num2 = temp;}