UNIT-3

DATE : ___/___/_____
PAGE : ________

⇒ Pointers in C-language

→ Pointers in C-language is an Variable which Store The Address of another Variable.
→ SYNTAX ::: datatype * ptrname = & oldvariablename; (Eg:- int * b = & a)
→ The Datatype of The Pointer Variable and The Variable it Points must be Same.
→ * = Indirection operater, Dereference operater, Value at Address.
→ Dereferencing means refer To The Value of Variable To which it Points using *.

Q→

```
#include <stdio.h>
void main () {
        int num, *ptr;
        ptr = & num;   // Same as int * ptr = & num;
        printf ("Enter The Number ::: ");
        scanf ("%d", & num);   // Let, 10 is Entered
        printf ("Value of num is %d", *ptr);   // Output is 10...
        printf ("Address of num is %u", & num);   // output is Address of num....
}
```

OUTPUT = ____

★→ %u = To Print Address.

*(& num) ≡ num → 10 !!!

⇒ Generic Pointer and Null Pointer

1. Generic Pointer

→ Generic Pointers are Capable To Point To any datatype.
→ A Generic Pointer is a Pointer Variable That has Void as its datatype.
→ It is a Special Type of Pointer That is used To Point To Variables of any datatype.
→ malloc() and calloc() are 2 functions which Returns The Void / Generic Pointers.
→ malloc() and calloc() are Used for Dynamic Memory Allocation in C-language.
→ Generic Pointers Indicate That The Pointer is Basically Empty and due To This, It is Capable of holding any datatype Address in The Program.

★→ [int * ptr = NULL;]. In Some Cases, we prefer To have Null Pointer which is a Very Special Pointer That doesn't Point To any Variable. It means That Null Pointer doesn't Point to any Valid memory Address.

Good Write

DATE: ___/___/_____
PAGE: _____

→ Pointer Arithmatic

1. Pointer Addition

→ In Pointer Addition, If we add any Integer To a Pointer, Then, That Integer Multiplied with the Size of Integer will Get add To The Address of The old Pointer.

⊙ Eg:- ptr 2 = ptr 1 + Integer (Size of Int)

ptr = 1000 + 5 (4)

ptr = 1000 + 20 = 1020 = New Address in Memory

2. Pointer Subtraction

→ In Pointer Subtraction, we subtract 2 pointers and Result is an Integer Value.

⊙ Eg:- int val = ptr 2 - ptr 1 = 1020 - 1000 = 20

20 / Size of datatype = No. of Variables Can be Stored

3. Pointer Increment

→ In an 32-Bit Machine, Increment is by 2 whereas in 64-Bit, Increment is by 4.

⊙ Eg:-

```
#include <stdio.h>
Void main () {
    int number = 50; // Address is 3210 3412 and 64-Bit Machine.
    int * ptr = & number;
    printf ("Address is %u \n", ptr);
    ptr ++;
    printf ("After Increment, Address is %u", ptr);
}
```

∴ OUTPUT ⟶ Address is 3210 3412

After increment, Address is 3210 3416

→ As, The Given machine was a 64 Bit Machine and due To This, There is Increment of 4 in The Address when Increment of pointer is applied on a 64 Bit Machine.

Good Write

DATE: __/__/____
PAGE: _____

4. Pointer Decrement

→ This is Very Similar To Pointer Increment, In 32 Bit Machine, Decrement is by 2 and in 64 Bit Machine, Decrement is by 4.

◉ Eg:-
```
# include <Stdio.h>
Void main () {
    int number = 50; // Address is 3210 3412 and 64-Bit Machine
    int * ptr = & number;
    printf ("Address is %u \n", ptr);
    ptr --;
    printf ("Address after Decrement is %u", ptr);
}
```

∴ OUTPUT → Address is 3210 3412
Address after Decrement is 3210 3408

→ As, The Given Machine is 64-Bit Machine and due To This, There is a decrement of 4 in The Address when Decrement of pointer is applied on a 64-Bit Machine.

5. Pointer Comparison → { ==, >=, <=, >, <, != }

→ Pointer To Pointer (Double Pointer)

→ A Pointer To a Pointer is a Form of Multiple Indirection or a Chain of Pointers.
→ Normally The Pointer is Consisting of The Address of The Variable but when we define a Pointer To Pointer, The First Pointer Contains The Address of The 2nd Pointer which Points To The location That Contains The Exact Value.

◉ Eg:-
```
# include < Stdio. h>
Void main (){
    int Val = 3000;
    int * ptr = & Val;
    int * * ptr1 = & ptr;
    printf (" Value at Val is %d \n", Val); // 3000
    printf (" Value available at ptr is %d \n", *ptr); // 3000
    printf (" Value available at ptr 1 is %d", **ptr1); // 3000
}
```

Good Write

DATE: ___/___/____
PAGE: _____

➡ Pointer To Function

➡ Pointer may point to a Function in C-Language.

➡ The Declaration of the Pointer must be same as the Function. The Datatype of the Pointer and the Returntype of the Function must be same.

◉ Eg:-

```
# include < stdio.h >
int addition () {
        int a = 8, b = 9;
        return a+b ; // Will Return 17....
}
void main () {
        int result;
        int (* ptr) (); // Syntax of Pointer To Function.....
        ptr = & addition ; // Pointing the Pointer To the Function.....
        result = (* ptr) (); // result will Get 17 ...
        printf (" The Sum is %d", result); // Answer is 17...
}
```

➡ Structures and Unions

➡ Keyword for Structure is "struct" and Keyword for Union is "union".

➡ Structure is a Type of data which is user defined and Structure allows the user to Combine Together logically in various related data Items of various datatypes and it Basically represent a Record of various Data.

➡ All the Elements of an object of Structure are Stored in Contigious Memory locations.

➡ SYNTAX ::: struct structurename {
        // Variables
        // Variables
    } ; → Most Imp....

} Syntax of Structure .....

➡ SYNTAX ::: union Unionname {
        // Variables
        // Variables
    } ; → Most Imp....

} Syntax of Union .....

Good Write

DATE: __/__/____
PAGE: _____

Note → The Basic diff. b/w Structure and Union is That in struct, different Memory locations are There Whereas in Union, Single Memory Location is Present.

Struct	Union
→ A User Can deploy The Keyword Struct To define a Structure.	→ A User Can deploy The Keyword Union To define a Union.
→ The Implementation of struct in C - occurs internally because It Contains Seperate Memory locations allotted To Every Input Member.	→ Memory Allocation is done only for one member with The largest size among all Input members. It stores The same location among all These Members.
→ User Can Access Individual members at a Given Time.	→ User Can access only one member at a Given Time.
→ Structure don't have a shared location for all of its members.	→ Union doesn't have seperate location for Every member in it.
→ Altering The Value of a Single member doesn't affect The other Members of Structure.	→ When you alter The Value of a Single member, It affects The Value of other Members.

⊙ Eg:-

```
#include <stdio.h>
int main() {
    struct student Yash;
    printf("Enter The Roll Number::: ");
    scanf("%d", &Yash.roll);
    printf("Enter The Name ::: ");
    scanf("%s", Yash.name); // No '&' for Name of struct...
    printf("Roll Number is %d \n", Yash.roll);
    printf("Name is %s", Yash.name);
    return 0;
}
```

```
struct student {
    int roll;
    char name[20];
};
```

∴ OUTPUT →

```
Enter The Roll Number ::: 32
Enter The Name ::: YashPandey
Roll Number is 32
Name is YashPandey
```

Good Write

⇒ Array of Structure

DATE: __/__/____
PAGE: _____

→ We use an Array of Structure whereever we need to store the data of many Number of Entities, Then, we have to use array of Structure.

→ SYNTAX ::: Struct Strname objectname [number];

⊙ Eg:-

```
#include <stdio.h>
#include <string.h>
Struct Student {
        int roll;
        Char name[20];
};
int main () {
        Struct Student st[5]; // Array of Structure ....
        for (int i=0; i<5; i++){ //Loop To Input Data .....
                printf ("Enter The Roll Number ::: ");
                Scanf ("%d", &st[i].roll);
                printf ("Enter The Name ::: ");
                Scanf ("%s", st[i].name); // No. '&' in name
        }
        for (int i=0; i<5; i++) { //Loop To Display data ....
                printf ("The Roll N.o. is %d \n", st[i].roll);
                printf ("Name is %s \n", st[i].name);
        }
        return 0;
}
```

⇒ Self - Referential Structure (SRS)

→ Self-Referential Structure are Those Structures That Contains a Reference To data of it's Same Type. Self-Referential Structure plays an V. Imp. Role in Creating other Data-Structures like Linked List.

→ Self-Referential Structures are widely used in Dynamic Data Structures like Trees, Linked List and Graphs.

Good Write

⊙ Eg :- struct node {
int data1;
char data2;
struct node * link; // Link a pointer pointing to struct node....
}

DATE : __/__/____
PAGE : _____

→ The 2 Types of Self-Referential Structure (SRS) are :-

1. Single Link SRS
2. Multiple Link SRS

★ In Single Link SRS, There is only a Single Pointer That carries The Address To The Next Node and That Pointer Variable is of Structured Mode Type Itself whereas in Multi Link SRS, We will make use of 2 Pointers Pointing To The Same Structure.

→ Typedef in C-Language

→ Typedef is used for Giving a New Name To an Existing datatype.
→ It is a Keyword which is used in C-Prog To Provide and Create The Existing datatypes with a New Name.
→ When The Names of The Datatype becomes difficult to Use in The Program, Then Typedef is used with a user defined datatype. (Alias = new)
→ SYNTAX ::: typedef <Existing name> <Alias name>;

⊙ Eg :- #include <stdio.h>
typedef struct structure { —— ①
int roll;
char name[20];
} Class;
void main() {
Class yash; // Created object of ① with Keyword Class.....
yash.roll = 32;
printf("Roll Number is %d", yash.roll);
}

∴ OUTPUT ⟶ Roll Number is 32.

Good Write

DATE: __/__/____
PAGE: _____

→ The Applications of Typedef Keyword are:-

1. Typedef is used to rename the Variablename.
2. Increases the Code Readability of the Program.
3. Multiple Pointers in single Link can be Created.

⇒ Enumulation

→ Enumulation of Enum in C is a special kind of datatype which is defined by user.
→ It is Consist of Const. Integrals that are Given by user.
→ If Value of an Variable in Enum is not defined, Then, The Value of That Variable is 1 more Than the Previous Variable.

◎ Eg:-

```
# include < stdio.h >

int main ( ) {
    Enum { Red = 2, Blue, Black = 5, Green = 7, Yellow, Purple, White = 15 };
    printf (" %d are Red ", Red);      ——→ 2
    printf (" %d are Blue ", Blue);    ——→ 3
    printf (" %d are Black ", Black);  ——→ 5
    printf (" %d are Green ", Green);  ——→ 7
    printf (" %d are Yellow ", Yellow); ——→ 8
    printf (" %d are Purple ", Purple); ——→ 9
    printf (" %d are White ", White);  ——→ 15
    return 0;
}
```

⇒ Nested Structure

→ A Structure Inside another Structure is Called a Nested Structure.

Easy

Q → Write a Code to Understand Structure inside Structure ?

DATE: __/__/____
PAGE: _____

→
```
# include < stdio.h >
struct dateofbirth {
        int day ;
        int month ;
        int year ;
};
struct student {
        int roll;
        char name [20];
        struct dateofbirth DOB;
};
void main ( ) {
        struct student Yash;
        Yash . roll = 32;
        strcpy ( Yash . name , "YashLandey");
        Yash . DOB . day = 29 ;
        Yash . DOB . month = 7 ;
        Yash . DOB . year = 2003;
        printf ( "Roll N.O. is %d", Yash . roll); ——→ 32
        printf ( "Name is %s", Yash . name ); ——→ YashLandey
        printf ("Day of DOB is %d", Yash . DOB . day); ——→ 29
        printf ("Month of DOB is %d", Yash . DOB . month); ——→ 7
        printf ("Year of DOB is %d", Yash . DOB . year); ——→ 2003
}
```

∴ OUTPUT →
```
Roll N.O. is 32.
Name is YashLandey.
Day of DOB is 29.
Month of DOB is 7.
Year of DOB is 2003.
```

IMP.

DATE : __/__/____
PAGE : _____

⇒ File Handling in C

→ Sometimes, it is not Enough To display The data on The Console because The data is To Big To Be displayed may be because of V. large Data and due To This, we Can only display a Limited Amount of data on The Console. Thus, We need File Handling in C for These Cases.

→ The different operations on File Handling are :-

1. Creation of a file.
2. Opening of a file.
3. Reading of a file.
4. Writing into a file.
5. Deleting The file.

} Basic Operations Associated on Files.....

★ We use Pointers To Access The File.

⇒ Functions Used in File Handling

1. Fopen() ⟶ Open new of Existing file.
2. Fprintf() ⟶ Used for Writing The Data into The file.
3. Fscanf() ⟶ Used To Read The Data from The file.
4. Fputc() ⟶ Write a Single Character into The file.
5. Fgetc() ⟶ Read a Single Character from The file.
6. Fseek() ⟶ Set The File Pointer To a Particular Position.
7. Fputw() ⟶ Write an Integer To a file.
8. Fgetw() ⟶ Read an Integer To a file.
9. Rewind() ⟶ Set The File Pointer To The Beginning of The file.
10. Ftell() ⟶ Returns The Current Position of The File.

⇒ Fopen() fn

→ SYNTAX ::: File * Fopen (Const Charname * Filename, Const Charname * mode);

◉ Eg :- File * Fp;

Good Write Fp ("Yash.c", "r");

DATE: __/__/____
PAGE: _____

→ The Working of Fopen() is as follow:-

1. Firstly, It Searches The File To be Opened.
2. If it is There, Then, it Opens it Otherwise it will Create it.
3. It load The file from The disk and Place it on The Buffer. The Buffer is used To Provide Efficiency for The read Operation.
4. It Sets up a Character Pointer Which Point To The 1st Character of File.

✳ Modes of File Handling are:-

(i) r → Opens a Text File into The Read File mode.
(ii) w → Opens a Text File into The Write File mode.
(iii) r+, w+, a+ → Opens a Text File into read and write mode.
(iv) rb → Opens a Binary File in read mode.
(v) wb → Opens a Binary File in Write mode.
(vi) ab → Opens a Binary File in append mode
(vii) rb+, wb+, ab+ → Opens a Binary File in read and Write mode.

⊙ Eg:- (Program of Fopen() Function)

IMP.

```
#include <stdio.h>
Void main(){
    FILE * Fp;
    Char ch;
    Fp = Fopen("yash.c", "r");
    while(1){
        ch = Fgetc(Fp);
        if(ch == EOF){
            break;
        }
    }
    printf("%c", ch);
    Fclose(Fp);
}
```

Good Write

DATE : __/__/____
PAGE : _____

* Eclose() → This Function is used to close a File.
→ SYNTAX ::: Eclose (File * Filename);

Q → Write Code For Working of Eprintf() fn ?

→
```
# include < Stdio.h>

int main (){
        FILE * Fp ;                          ——— (1)
        Fp = Eopen ("File.txt", "w");        ——— (2)
        Eprintf ( Fp, "Hello \n");           ——— (3)
        Eclose ( Fp);                        ——— (4)
        return 0;
}
```

Explination → (1) will Create an File pointer and in (2), we will link That pointer To The "File.txt" file in write mode, in (3), we will write "Hello" in The Text File with help of Eprintf() and (4) will disconnect our File pointer from The "File.txt" file. This is The Working of Eprintf() function.

— JAI LOGIC

1. Eputc() Function

→ Eputc() is The Simplest Function to write characters in a File.
→ SYNTAX ::: int fputc (character, File pointer);
→ This Fn returns the written character of The File and if The File is Empty, Then, This Fn will return EOF (End OF File).

→ In Eputc() function, Firstly The fn will delete all The characters and data written in The Text File, This function will first Empty The Text File and Then This function will add That characters To The Text file. Thus, firstly, The Text file will be made Empty, Then, character will be added To The Text file.

Good Write

DATE: ___/___/____
PAGE: _____

2. Egetc() fn in C

→ Egetc() fn helps to read the characters from a file.
→ SYNTAX ::: int fgetc(File * Fp);
→ Returns the Read character from the File otherwise Returns EOF (End of File) when the Given File is Empty.

Q → Write Code to understand the Code of Egetc() and Eputc() where you are Given a file, 'Yash.txt' which Contains "Yash Pandey"?

→
```
# include <stdio.h>
int main() {
    FILE * ptr = NULL;
    ptr = fopen("yash.txt", "r");          ——— (1)
    char c = fgetc(ptr);                   ——— (2)
    printf("The character 'c' is ::: %c", c);   ——— (3)
    fclose(ptr);                           ——— (6)
    File * ptr2 = fopen("yash.txt", "w");  ——— (4)
    fputc('O', ptr2);                      ——— (5)
    fclose(ptr2);                          ——— (7)
    return 0;
}
```

IMP.

∴ OUTPUT → The character 'c' is ::: Y

Explination → In (1), we linked our pointer ptr to the Text file in read mode where in (2), with fgetc() fn, the First character of the Text File; i.e. 'Y' will get Copied into the character variable of (2), In (3), we printed our character file and in (6) we disconnected our pointer ptr from the Text file using fclose() Fn. In (4), we created another file pointer and we pointed that pointer to the Text file in write mode, Due to (5), Firstly whole Text file will be cleared and char 'O' will Get written on the Text File. In (7), we disconnected the pointer ptr 2 from the File. This is the main Logic Behind File Handling.

— JAI LOGIC

Good Write

File_Handling.c

```c
#include<stdio.h>

int main(){
    FILE *Ptr=fopen("yash.txt","r");
    char c=fgetc(ptr);
    fclose(ptr);
    FILE *Ptr2=fopen("yash.txt","w");
    fputc('o',ptr2);
    fclose(ptr);
    return 0;
}
```

Snipped

UNIT-4

DATE: ___/___/_____
PAGE: _____

→ C-99 Extensions

→ The C99 Previously Known as 'C9X' is an Informal Name of ISO/IES or 9899:1999 of C-Programming Standard. It is The Enhanced and New Version of C90 with Added Features for The language and The Standard language. This leads To The Better Implementation of Computer Hardware (IEEE Arthimatic and Computer Tech.).

→ Some Important Features of C-99 Extensions are:-

→ To Create a Macros With a Variable N.O.of Arguments.
→ C99 allows To Use Sofisticated Numbers and Designated Initilizers.
→ Restricted Pointers and New Comment Techniques (//).
→ Variable length Array and Flexible Array Members.
→ Inline Functions and New Header Files were Included, Addition of Compound Literals.
→ Some new Keywords as "BOOL", "COMPLEX", "IMAGINARY", "INLINE" & "RESTRICT".

→ ANSI ⟶ American National Standard Institute.
ISO ⟶ International Organization for Standardization.

★ The C99 Standard Incoporated The New Enhancements and Included Advanced features that are desirable for any modern Computer language. Some of The Features are been borrowed from C++ while others are a Modification of few Constructs.

→ Special features of C99 ⟶ IMP.

1. Declaration Of Variables ⟶ It is legal To declare Variables in any Point of The Program within The Curly Braces ({}) of The main() function.

2. Variable length Array ⟶ In ANSI, C-Array Dimension must have To be declared. But, C99 Permits declaration Of Array Dimensions Using Integer Variables of any Valid Integer Expressions. This is Called Variable Length Array.

3. Command Technique ⟶ C-99 Allows To Put Comment using Double Slash (//), as done in C++ and C-Compiler Willn't Show any Error which was not Possible in The ANSI-C language.

Good Write

IMP.

DATE: ___/___/_____
PAGE: _____

→ C Standard Libraries

→ The C Standard Library Refers To a Collection of Standardized implemented functions for a specific Compiler and Platform. These Static and dynamic Libraries are Typically Implemented by The Compiler Creators and are Included in Installation.

→ These Standard Libraries are Responsible for Interacting with Lower-level Systems and They are Supposed to offer further program portability to a Code Base.

→ Most C Compilers Come with a Set of Standard Libraries to Adhere The ANSI / ISO Standards and These Standard Libraries interact with Lower Level Systems and offer a Certain amount of Cross-Platform Compatibility.

1. stdio.h

→ #include <stdio.h> is Necessary Header file To Access it's Library Functions.

→ printf() and scanf() are defined in The Library pointed by stdio.h Header file. The Code of printf() and scanf() is not defined in The header file "stdio.h" but it is defined in a Library and "stdio.h" header file is pointing To That Library Section of C-Library.

2. math.h

→ #include <math.h> is Necessary Header file To Access it's Library Functions.

→ sqrt(), cbrt(), pow(,) are Library functions of "#include <math.h>".

→ These f^n Returns Float Value as Returntype ..... (FLOAT)

3. ctype.h

→ "#include <ctype.h>" is Necessary Header file To Access it's Library Functions.

→ toupper (char) ⟶ F^n will Convert lowercase char To Uppercase char.

tolower (char) ⟶ F^n will Convert Uppercase char To lowercase char.

4. String.h

→ "#include <string.h>" is Nessesary Header file to use it's library Functions.

→ Strlen() ⟶ Returns The length of The String.

Strcpy() ⟶ To Copy The 2nd String into 1st String.

Strcat() ⟶ Concatinate The 2nd string at End of 1st String.

Strcmp() ⟶ Returns 0 ($S_1 = S_2$), -Ve ($S_1 < S_2$) and +Ve ($S_1 > S_2$).

Strchr() ⟶ Returns a pointer to The first occurance of char ch.

Strstr() ⟶ Returns a pointer to The first occurance of S_2 on S_1.

5. stdlib.h

→ "#include <stdlib.h>" is Nessesary Header file to use it's library functions.

→ This Standard library is used for General Utility.

→ atof() ⟶ Fn to Convert String to float.

atoi() ⟶ Fn to Convert String to Int.

atol() ⟶ Fn to Convert String to long.

atoll() ⟶ Fn to Convert String to long long.

rand() ⟶ Fn to Get a Random Value.

abort() ⟶ Fn to Exit a Program

→ For Dynamic Memory Allocation, This header file is must.

6. assert.h

→ "#include <assert.h>" is Nessesary header file to use it's library Functions.

→ assert() fn is used in debug a program where if The Condn written in assert() fn is True, Then, below line Code will Exicute and If The Condn is False, Program will Stop There and Reason of Error will Get displayed on The output Screen.

7. time.h (Date and Time Fn)

→ "#include <time.h>" is Nessesary Header file to use it's library Functions.

→ Setdate() & getdate() ⟶ To Change or get The Time of The System.

time() and difftime() → To Get Actual Time or To Get diff-b/w 2 Time.

gmtime() ⟶ gmtime() To Get The Universal Time.

ctime() ⟶ Returns The Date and Time in form of String.

DATE: ___/___/_____
PAGE: _____

8. setjump.h ⟶ Neglected Part!!!

→ "#include <setjump.h>" is the Nessecary header file. For it's Library Functions.

→ Setjmp() and longjmp() are Library Fn of this Header File. The Working of these 2 fn is Very Similar To That of goto() fn.

9. Stdarg.h ⟶ Neglected Part!!!

→ "#include <stdarg.h>" is a Nessecary header File for it's Library Fn and it allow fn to Accept Indefinate N.o. of Arguments. It Provides facilities To Step Through a list of Fn arguments of Unknown Number and Time.

⊙ Eg:- Vprintf() i.e. Variadic Functions.

10. Unistd.h ⟶ Neglected Part!!!

→ "#include <Unistd.h>" is The Header file which is used To Provide The Access to The POSIX POSIX Operating System API. It also defines many symbols To Represent Configuration Variables and Implementation of provided features.

→ Sysconf(), Confstr(), pathconf() are Library Fn of Unistd.h.

Q → Write The Fn Code of: -

1. Factorial of a Number

```
→ int fact (int n) {
    if (n==1){
        return 1;
    }
    else {
        return n* fact(n-1);
    }
}
```

2. Fibbionoci of a Number Index

(1 1 2 3 5 8 13 21 34 ....)

```
→ int fib (int n) {
    if (n==0 || n==1){
        return 1;
    }
    else {
        return fib(n-1) + fib(n-2);
    }
}
```

Good Write

Exp 4

Q → Write Code for Linear and Binary Search on Array?

DATE: ___/___/____
PAGE: _____

→
```
#include <stdio.h>
int linear (int arr[], int n, int key){ // Time Complexity is O(n)----
    for (int i=0; i<n; i++){
        if(arr[i]==key){
            return i;
        }
    }
    return -1;
}
int binary (int arr[], int n,int key){ // Time Complexity is O(log n)-----
    int low=0, high=n-1;
    while (low<=high){
        int mid = (low+high)/2;
        if (arr[mid]==key){
            return mid;
        }
        else if (arr[mid]<key){ low=mid+1; }
        else { high=mid-1; }
    }
    return -1;
}
int main(){
    int arr[10]={1,2,3,4,5,6,7,8,9,10};
    int key=7;
    printf("The Index of %d Via Linear Search is %d", key, linear(arr,10,7));
    printf("The Index of %d Via Binary Search is %d", key, binary(arr,10,7));
    return 0;
}
```

(\n)

∴ OUTPUT → The Index of 7 Via Linear Search is 6.
The Index of 7 Via Binary Search is 6.

* For Binary Search, we have To pass an Sorted Array in Increasing Order of Value of Elements of The Array.

Good Write

Linear_And_Binary.c

```c
#include<stdio.h>

int linear(int arr[],int n,int key){ //Time Complexity Of This Code Is O(n).....
    for(int i=0;i<n;i++){
        if(arr[i]==key){
            return i;
        }
    }
    return -1;
}

int binary(int arr[],int n,int key){ //Time Complexity Of This Code Is O(logn).....
    int low=0,high=n-1;
    while(low<=high){
        int mid=(low+high)/2;
        if(arr[mid]==key){
            return mid;
        }
        else if(arr[mid]<key){
            low=mid+1;
        }
        else{
            high=mid-1;
        }
    }
    return -1;
}

int main(){
    int arr[10]={1,2,3,4,5,6,7,8,9,10};
    int key=7;
    printf("THE INDEX OF %d IN THE ARRAY VIA LINEAR SEARCH IS ::: %d\n",key,linear(arr,10,key));
    printf("THE INDEX OF %d IN THE ARRAY VIA BINARY SEARCH IS ::: %d\n",key,binary(arr,10,key));
}
```

Snipped

IMP.

DATE: ___/___/____
PAGE: _____

Q → Write The Code of Sorting Via Bubble Sort ?

→
```
#include <stdio.h>

Void Bubblesort (int arr[], int n){ // Time Complexity is O(n^2) .....
    for ( int i=0; i<n-1; i++){
        for (int j=0; j<n-1-i; j++){
            if ( arr[j] > arr[j+1]){
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
Void print (int arr[], int n){ // Fn To Print Sorted Array .....
    printf(" Sorted Array is ::: { ");
    for (int i = 0; i<n; i++){
        if ( i == (n-1) ){
            printf ("%d } ", arr[i]);
        }
        else {
            printf ("%d, ", arr[i]);
        }
    }
}
int main () {
    int arr[8] = { 4, 3, 1, 5, 2, 8, 7, 6};
    Bubblesort (arr, 8);
    print ( arr, 8);
}
```

∴ OUTPUT → Sorted Array is ::: {1, 2, 3, 4, 5, 6, 7, 8}

Good Write

Bubble_Sort.c

```c
#include<stdio.h>

void BubbleSort(int arr[],int n){ //Time Complexity Of Bubble Sort Is O(n^2)....
    for(int i=0;i<n-1;i++){
        for(int j=0;j<n-1-i;j++){
            if(arr[j]>arr[j+1]){
                int temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
}

void print(int arr[],int n){ //Function To Print The Sorted Array.....
    printf("SORTED ARRAY IS ::: {");
    for(int i=0;i<n;i++){
        if(i==(n-1)){
            printf("%d}",arr[i]);
        }
        else{
            printf("%d,",arr[i]);
        }
    }
}

int main(){
        int arr[7]={1,4,6,8,2,3,7};
        BubbleSort(arr,7);
        print(arr,7);
}
```

Snipped

IMP.

DATE: ___/___/_____
PAGE: _____

Q→ Write The Code Of Sorting Via Insertion Sort?

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

→

```
Void InsertionSort(int arr[], int n){ // Time Complexity is O(n^2)....
    for(int i=1; i<n; i++){
        int curr = arr[i];
        int prev = i-1;
        while(prev>=0 && arr[prev]>curr){
            arr[prev+1] = arr[prev];
            prev--;
        }
        arr[prev+1] = curr;
    }
}
Void print(int arr[], int n){ // Fn To Print Sorted Array.....
    printf("Sorted Array is ::: {");
    for(int i=0; i<n; i++){
        if(i==(n-1)){
            printf("%d}", arr[i]);
        }
        else{
            printf("%d,", arr[i]);
        }
    }
}
Void main(){
    int arr[8]={4,3,1,5,2,8,7,6};
    InsertionSort(arr, 8);
    print(arr, 8);
}
```

∴ OUTPUT → Sorted Array is ::: {1,2,3,4,5,6,7,8}

Good Write

Insersion_Sort.c

```c
#include<stdio.h>

void InsersionSort(int arr[],int n){ //Time Complexity Of Insersion Sort Is O(n^2)....
    for(int i=1;i<n;i++){
        int curr=arr[i];
        int prev=i-1;
        while(prev>=0 && arr[prev]>curr){
            arr[prev+1]=arr[prev];
            prev--;
        }
        arr[prev+1]=curr;
    }
}

void print(int arr[],int n){ //Function To Print The Sorted Array.....
    printf("SORTED ARRAY IS ::: {");
    for(int i=0;i<n;i++){
        if(i==(n-1)){
            printf("%d}",arr[i]);
        }
        else{
            printf("%d,",arr[i]);
        }
    }
}

int main(){
        int arr[7]={1,4,6,8,2,3,7};
        InsersionSort(arr,7);
        print(arr,7);
}
```

Snipped

IMP.

DATE: __/__/____
PAGE: _____

Q → Write The Code of Sorting Via Selection Sort?

```
#include <stdio.h>
Void Selectionsort (int arr[], int n){ // Time Complexity of Selection Sort is O(n^2)....
    for( int i=0; i<(n-1); i++){
        int min = i;
        for (int j=i+1; j<n; j++){
            if( arr[min] > arr[j]){ min = j; }
        }
        int temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }
}
Void print ( int arr[], int n){ // Fn To Print Sorted Array.....
    printf (" Sorted Array is ::: {");
    for (int i = 0; i<n; i++ ){
        if (i==(n-1)) {
            printf (" %d }", arr[i]);
        }
        else{
            printf ("%d, ", arr[i]);
        }
    }
}
Void main () {
    int arr[8] = { 4,3,1,5,2,8,7,6};
    Selectionsort (arr, 8);
    print (arr, 8);
}
```

∴ OUTPUT ⟶ Sorted Array is ::: {1,2,3,4,5,6,7,8}

Selection_Sort.c

```c
#include<stdio.h>

void SelectionSort(int arr[],int n){ //Time Complexity Of Selection Sort Is O(n^2)....
    for(int i=0;i<(n-1);i++){
        int min=i;
        for(int j=i+1;j<n;j++){
            if(arr[min]>arr[j]){
                min=j;
            }
        }
        if(i!=min){
            int temp=arr[i];
            arr[i]=arr[min];
            arr[min]=temp;
        }
    }
}

void print(int arr[],int n){ //Function To Print The Sorted Array.....
    printf("SORTED ARRAY IS ::: {");
    for(int i=0;i<n;i++){
        if(i==(n-1)){
            printf("%d}",arr[i]);
        }
        else{
            printf("%d,",arr[i]);
        }
    }
}

int main(){
        int arr[7]={1,4,6,8,2,3,7};
        SelectionSort(arr,7);
        print(arr,7);
}
```

Snipped

Q → Write Code To Calculate Square Root of a Number?

```
#include <stdio.h>
#include <math.h>
int main(){
        printf("Square Root of 100 is %f", Sqrt(100));
        printf("Cube Root of 216 is %f \n", Cbrt(216));
        printf("The Value of 8 Raised To Power 3 is %f \n", pow(8,3));
        return 0;
}
```

\n

* Return Type of Sqrt(), Cbrt(), pow() are Float Value Not INT Value.....

∴ OUTPUT →
```
Square Root of 1000 is 10.00000000
Cube Root of 216 is 6.00000000
The Value of 8 Raised To Power 3 is 512.00000000
```

Good Write

Square_Root.c

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

int main(){
    int a=100;
    printf("SQUARE ROOT OF %d IS %f\n",a,sqrt(a));
    int b=216;
    printf("CUBE ROOT OF %d IS %f\n",b,cbrt(b));
    int c=8,d=3;
    printf("THE VALUE OF %d RAISED TO POWER %d IS %f\n",c,d,pow(c,d));
}
```

Snipped

Easy

DATE: __/__/____
PAGE: _____

Q → Write Code To Reverse an Array?

→

```
#include <stdio.h>
Void reversearray (int arr[], int start, int end){ // Fn To Reverse Array....
        While ( start < end){
                int temp = arr [start];
                arr [start] = arr [end];
                arr [end] = temp;
                start ++;
                end --;
        }
}
Void print (int arr[], int n){ // Fn To Print Array.....
        printf (" Array is ::: { ");
        for (int i=0 ; i< n ; i++){
                if (i == (n-1) ){
                        printf (" %d } ", arr [i]);
                }
                else{ printf (" %d, ", arr [i]); }
        }
}
void main () {
        int arr [9] = { 1, 2, 3, 4, 5, 6, 7, 8, 9};
        print ( arr , 9);
        reversearray ( arr, 0, 8);
        printf ("\n After reversearray () function ::: " );
        print ( arr, 9);
}
```

∴ OUTPUT ⟶ Array is ::: { 1, 2, 3, 4, 5, 6, 7, 8, 9 }

After reversearray () function ::: Array is ::: { 9, 8, 7, 6, 5, 4, 3, 2, 1 }

Array_Order_Reversal.c

```c
#include<stdio.h>

void ReverseArray(int arr[],int start,int end){ //Function To Reverse The Array.....
    while(start < end){
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}

void print(int arr[],int n){ //Function To Print The Reversed Array.....
    printf("ARRAY IS ::: {");
    for(int i=0;i<n;i++){
        if(i==(n-1)){
            printf("%d}",arr[i]);
        }
        else{
            printf("%d,",arr[i]);
        }
    }
}

int main(){
    int arr[9]={1,2,3,4,5,6,7,8,9};
    print(arr,9);
    ReverseArray(arr,0,8);
    printf("\nAFTER ReverseArray() Function ::: ");
    print(arr,9);
    return 0;
}

```

Snipped

Easy

DATE: ___/___/_____
PAGE: _____

Q → Write Code To Reverse a String?

→
```
#include <Stdio.h>
#include <String.h>

Void ReverseString (Char * str1){ // Fn To Reverse The String.....
        int temp;
        int len = Strlen (str1);
        for (int i = 0; i < (len/2); i++){
                temp = str1[i]
                str1[i] = str1[len-i-1];
                str1[len-i-1] = temp;
        }
}
Void main (){
        Char str[10] = "YashPandey";
        ReverseString (str);
        printf ("Reversed String is ::: %s", str1);
}
```

∴ OUTPUT → Reversed String is ::: yednaPhsaY

Reverse_String.c

```c
#include <stdio.h>
#include <string.h>

void ReverseString(char*str1){  //Function To Reverse The String.....
    int temp;
    int len=strlen(str1);
    for(int i=0;i<len/2;i++){
        temp=str1[i];
        str1[i]=str1[len-i-1];
        str1[len-i-1]=temp;
    }
}

int main()  {
    char str[50];
    printf ("ENTER THE STRING ::: ");
    scanf("%s",&str);
    printf("BEFORE REVERSING THE STRING , STRING IS ::: %s \n", str);
    ReverseString(str);
    printf("AFTER REVERSING THE STRING , STRING IS ::: %s \n", str);
}
```

Snipped