Enter your last name: Franklin
EdwardFranklin

Output a string in C use the following two methods:

1. **using printf() with \n**

We saw the printf() function which we used to output a string in C as it is to the user. But, if we wanted to output two strings with a space in between them or conversely output two strings in different lines, we could do this also by using the printf() function.

To output two strings in two separate lines in C, we will have to include the newline character \n in the printf() function. The newline character will make sure that the next time anything is printed, it is printed in the next line. The complete syntax for this method is:
printf("%s \n", char *s);
printf("%s \n", char *s1)
Here, s and s1 are the pointers which point to the array of characters which contain the strings that we need to output. Because of the \n newline character, s1 string will be output to the user in the next line.

To output two strings with a space in between, but in the same line, we just need to include that space in the printf() function. The complete syntax for this method is:
printf("%s %s", char *s, char *s1);
Here, s and s1 are the pointers which point to the array of characters which contain the strings that we need to output.

Let us look at an example to understand this better.

```
#include <stdio.h>
int main()
{
// char array that stores first name of the student
   char f_name[20];

// char array that stores last name of the student
   char l_name[20];

// take user input
   printf("Enter your first name: \n");

// use scanf() to get input
   scanf("%s", &f_name);

// take user input
   printf("Enter your last name: \n");
```

```
// use scanf() to get input
   scanf("%s", &l_name);

// printing the input value in separate lines using printf()
   printf("%s \n", f_name);
   printf("%s \n", l_name);

// printing the input value in the same line with a space using printf()
   printf("%s %s", f_name, l_name);

   return 0;

}
```

Output

```
Enter your first name: Edward
Enter your last name: Franklin
Edward
Franklin
Edward Franklin
```

2. using puts()

Another function which we can use to output a string in C is the puts() function. The thing to note however is that after printing the given string in the output, the puts() function transfers the control to the next line. So any other string that we print after the execution of the puts()line will be printed in the next line by default and not in the same line.

The complete syntax for this method is:

```
int puts(const char* strptr);
```

Here, strptr is the pointer which points to the array of characters which contains the string which we need to output.

Let us look at an example to understand this better.

```
#include <stdio.h>

int main()
{
// array that stores first name of the student
   char f_name[20];

// array that stores last name of the student
   char l_name[20];

// take user input
   printf("Enter your first name: \n");
```

```
// use fgets to get input
   fgets(f_name,20,stdin);

// take user input
   printf("Enter your last name: \n");

// use fgets to get input
   fgets(l_name,20,stdin);

// printing the input value using puts()
   puts(f_name);
   puts(l_name);

   return 0;

}
```

Output

```
Enter your first name: Edward
Enter your last name: Franklin
Edward
Franklin
```

As you can see in the code, we never specified that we need to leave a line after printing the first name, but the code still did, because of using the puts() function, which by default transfers control to the next line after printing.

Comparision between puts() and fputs()

The puts() and fputs() functions both are used to output strings to the user. While the puts() method converts the null termination character \0 to the newline character \n, the fgets() method does not do so. Hence, in the case of puts(), the control is transferred to a new line in the output, but in case of fputs(), it remains in the same line.

What is an Array of Strings in C?

In C, an array represents a linear collection of data elements of a similar type, thus an array of strings means a collection of several strings. The string is a one-dimensional array of characters terminated with the null character. Since strings are arrays of characters, the array of strings will evolve into a two-dimensional array of characters.

Syntax

It is possible to define an array of strings in various ways, but generally, we can do so as follows:

```
char variableName[numberOfStrings][maxSizeOfString];
```

Initialization of Array of Strings

Initialization in C means to provide the data to the variable at the time of declaration. It is pretty straightforward to initialize an array of strings. We can just create an array of some size and then we can place comma-separated strings inside that. See the example below to get a good idea about it.

```
char car[5][20] = {"Mahindra", "Audi", "Lamborghini", "Hyundai", "Tata Motors"};
```

Explanation:

In this example, we have initialized a two-dimensional character array **car** which can have a maximum of 5 strings of size 20.

Strings as an Array of Characters

As we know, C strings are nothing but an array of characters. Therefore we can initialize the string as an array of characters instead of directly writing the string in double quotes. Both approaches are the same. In the latter scenario, C interprets the string in a similar manner and appends a null character at the end.

```
char str[5] = {'H', 'e', 'l', 'l', 'o', '\0'};
char str[5] = "Hello";
```

We can also create our array of strings with this approach of writing strings as an array of characters. Let's re-write the same initialization,

```
char car[5][20] = {
    {'M','a','h','i','n','d','r','a', '\0'},
    {'A','u','d','i','\0'},
    {'L','a','m','b','h','o','r','g','i','n','i','\0'},
    {'H','y','u','n','d','a','i','\0'},
    {'T','a','t','a', ' ', 'M', 'o', 't', 'o', 'r','s','\0'}
};
```

Example of Array of String in C

1. Reading and Displaying Array of Strings

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

int main()
{
   char car[5][20] = {"Mahindra", "Audi", "Lambhorgini", "Hyundai", "Tata
Motors"};
   for(int itemNumber = 0 ; itemNumber < 5 ; itemNumber++)
   {  //Iterate over selected string
      for(int characterPosition = 0 ; characterPosition < strlen(car[itemNumber]) ;
characterPosition++)
         printf("%c", car[itemNumber][characterPosition]);
      printf("\n");
   }
```

```
    return 0;
}
```

Output:

Mahindra
Audi
Lamborghini
Hyundai
Tata Motors

Explanation:

- We have created an array of strings, which has a fixed size of 5, and we have initialized that array at the time of declaration with the strings.
- Subsequently, we are looping in a nested manner, in which the outer loop is accessing the item of the array and the inner loop is printing the characters of that string till the end.

Functions of Strings

There are several functions that help us while manipulating strings in C. Some of them are described below.

strcat() Function in C

This function is used to concatenate two strings.
char *strcat(char *destination, const char *source)
It accepts two pointers one to the base address of the destination string and the other to the source string and then concatenates the source to the destination and returns the concatenated destination string.

strlen() Function in C

This function is used to find the length of the string.
size_t strlen (const char * str);
It accepts a string and returns the size.

strcmp() Function in C

This function is used to compare two strings.
int strcmp (const char* str1, const char* str2);
It accepts two pointers to the strings and compares them, returns 0 if they are identical otherwise returns a non-zero value.

strcpy() Function in C

This function is used to copy a string.
char* strcpy(char* destination, const char* source);
It accepts two pointers to the destination and source string and copies the characters of the source to the destination, subsequently, it returns the pointer to the destination string.

strrev() Function in C

This function is used to reverse a given string.
char* strrev (const char* str);
It accepts a string and performs a reverse operation finally returning the pointer to the same string.

Some Invalid Operation on an Array of String

1. If we access a memory block that doesn't exist or is out of limits it will contain some garbage, so we should always access the memory according to its limit. All those operations in which we access invalid memory will be referred to as invalid operations in an array of strings.

2. The strings pointed by a pointer(in array of string having type char*) are internally translated as **constant char array** or read-only, so you cannot change them that's why all operations which try to modify the string by its pointer will be invalid.
An important point to note, that might create confusion is, You cannot modify a constant string but still you can always change the pointer value to point to another string.

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

int main()
{
    char* arr[2] = {"abc", "def"};
    strcpy(arr[0], "ghi"); // valid operation is arr[0]="ghi"
    printf("%s", arr[0]);
    return 0;
}
```

Output:

```
Segmentation fault (core dumped)
```

Apart from the strcpy, the use of functions like strcat, gets, fgets, and scanf to modify these types of strings will also be invalid.

3. When we write an array of strings with 2D array notation, the pointer becomes constant. So we cannot change it further to point to some other string.

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

int main()
{
    char arr[2][20] = {"abc", "def"};
    arr[0] = "ghi";

    printf("%s", arr[0]);
    return 0;
}
```

Output:

```
code.c: In function 'main':
code.c:7:12: error: assignment to expression with array type
    7 |     arr[0] = "ghi";
      |            ^
```

The above code gives an error as the name of the array is a constant pointer and we try to change it. But we can change the content of that string.

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

int main()
{
   char arr[2][20] = {"abc", "def"};

// arr[0] = "ghi";  Invalid
   strcpy(arr[0], "ghi");
   printf("%s", arr[0]);
   return 0;
}
```

Output:

ghi

In the above code, we change the content of the string pointer by arr[0] which is valid as it is still pointing to the same location.

How Does Array of Strings Work in C?

Strings in C are a collection of characters, more formally arrays of characters that are terminated by a null character, which mean the last character will be \0 to determine that it is the end of the character array. So, the array of strings works in a similar way. It is just a two-dimensional character array.

Methods about How to Declare an Array of Strings in C

Use 2D Array Notation to Declare Array of Strings in C

As we know strings are nothing but an array of characters and hence the array of a string would be a two-dimensional array of characters. Suffice it to say, we can use a 2D character array to declare an array of strings in C.

```
char car[5][20] = {
   {'M','a','h','i','n','d','r','a', '\0'},
   {'A','u','d','i','\0'},
   {'L','a','m','b','h','o','r','g','i','n','i','\0'},
   {'H','y','u','n','d','a','i','\0'},
   {'T','a','t','a', ' ', 'M', 'o', 't', 'o', 'r','s','\0'}
};
```

Use the char* Array Notation to Declare Array of Strings in C

Before discussing this approach, let's understand what * means. When we write the * in front of the data type it becomes the pointer, which can point to the memory block of the same data type. Now if we assign the array to the pointer it points to the base address of the array. This means we can store the base address of an array in the pointer and then traverse the entire array.

So char* creates a pointer that points to the base address of the string. To declare

an array of strings we can create an array of these kinds of pointers. See the example below for a better illustration.
An important point to note is, that internally this **pointer to string** notation is translated as **const** character array, which means you cannot modify them.

```
#include <stdio.h>

int main()
{
   char *car[5] = {"Mahindra", "Audi", "Lambhorgini", "Hyundai", "Tata Motors"};

   for(int i = 0 ; i < 5 ; i++)
   {
      printf("%s\n", car[i]);
   }
   return 0;
}
```

Output:

```
Mahindra
Audi
Lamborghini
Hyundai
Tata Motors
```

Conclusion

- An array of strings in C is a one-dimensional array of strings and a two-dimensional array of characters.
- We can declare the array of strings by pointer method (char*) or by using 2d notations.
- The initialization of the array of strings is simple, we can either provide a string wrapped inside double quotes to create an array of pointers, or we can write a nested array of characters.
- When we create an array of strings by using char* notation/pointer approach, all strings are internally interpreted as a constant character array, so we cannot change it later but in the case of an array of strings by 2d notations the string pointers are constant, so we could change their content but not that address they pointing.
- While accessing strings we cannot access invalid or out-of-bound memory. This might either create undefined behavior or error.
- The following table provides most commonly used string handling function and their use...

Function	Syntax (or) Example	Description
strcpy()	strcpy(string1, string2)	Copies string2 value into string1
strncpy()	strncpy(string1, string2, 5)	Copies first 5 characters string2 into string1

Function	Syntax (or) Example	Description
strlen()	strlen(string1)	returns total number of characters in string1
strcat()	strcat(string1,string2)	Appends string2 to string1
strncat()	strncpy(string1, string2, 4)	Appends first 4 characters of string2 to string1
strcmp()	strcmp(string1, string2)	Returns 0 if string1 and string2 are the same; less than 0 if string1<string2; greater than 0 if string1>string2
strncmp()	strncmp(string1, string2, 4)	Compares first 4 characters of both string1 and string2
strcmpi()	strcmpi(string1,string2)	Compares two strings, string1 and string2 by ignoring case (upper or lower)
stricmp()	stricmp(string1, string2)	Compares two strings, string1 and string2 by ignoring case (similar to strcmpi())
strlwr()	strlwr(string1)	Converts all the characters of string1 to lower case.
strupr()	strupr(string1)	Converts all the characters of string1 to upper case.
strdup()	string1 = strdup(string2)	Duplicated value of string2 is assigned to string1
strchr()	strchr(string1, 'b')	Returns a pointer to the first occurrence of character 'b' in string1
strrchr()	'strrchr(string1, 'b')	Returns a pointer to the last occurrence of character 'b' in string1
strstr()	strstr(string1, string2)	Returns a pointer to the first occurrence of string2 in string1
strset()	strset(string1, 'B')	Sets all the characters of string1 to given character 'B'.
strnset()	strnset(string1, 'B', 5)	Sets first 5 characters of string1 to given character 'B'.
strrev()	strrev(string1)	It reverses the value of string1

UNIT - V

Structures: Definition and Initialization of Structures, Accessing Structures, Nested Structures,
Arrays of Structures, Structures and Functions, Pointers to Structures, Self Referential Structures, **Unions**, Type Definition (typedef), Enumerated Types.
Input and Output: Introduction to Files, Modes of Files, Streams, Standard Library Input/Output Functions, Character Input/Output Functions

Structures

We often have to store a collection of values. When the collection is of the same type then array is used to store values. When the collection of values is not of the same type then a structure is used to store the collection.
An Array is a group of related data items or collection of homogeneous data that share a common name i.e. it contains element s of same datatype. If we want to represent a collection of data items of different types using a single name, then we cannot use an array.
C supports a constructed data type known as structure which is a method of packing data of different types. A structure is a convenient tool for handling a group of logically related data items. It is a collection of heterogeneous data that share a common name.
A structure is a collection of items which may be of different types referenced commonly using one name. Each item in the structure is called as a member.

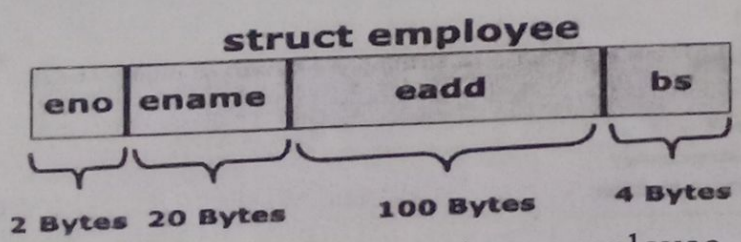
Declaration of Structure

A structure is also called user-defined data type. When a structure is declared a ne w data type is created. You declare a structure using keyword struct . The following is the syntax to declare a structure:

```
struct [structure-name]
{
members declaration
}[variables];
```

If we want to store details of an employee we have to create a structu re:

```
struct employee
{
int eno;
char ename[20]; char eadd[100]; float bs;
};
```



Memory Image for structure employee

The above is a structure declaration for employee structure. Employee structure contains members eno, ename, eadd, and bs. Each member may belong to a different type of data. When you declare a structure you just create a new data type - struct employee. Structure name is otherwise known as tag. The tag name may be used subsequently to declare variables that have the tag's structure.

Note that the above declaration has not declared any variables. It simply describes a format called template to represent information as shown below:

eno	Integer
ename	20 Characters
eadd	100 Characters
bs	Float

While declaring a structure you can also declare variables by giving list of variables after right brace. Members of a structure may be of standard data type or may be of another structure type.

```
struct point
{
int x,y;
}fp,sp; /*sp and fp are two structure variables*/
```

Declaring Structure variable You can declare structure variables using struct structname . You can use a variable of any structure anywhere in the program where the structure is visible. Normally structures are declared at the beginning of the program and outside all functions, so that they are global and available to the entire program. Following is the declaration of structure variable e1:

```
struct employee e1;
```

When the variables are created to the template then the memory is allocated. The number of bytes occupied by structure variable is equal to the total number of bytes required for all members of the structure. For example, a variable of s truct employee will occupy 126 bytes (2+20+100+4). So, 126 bytes are allocated for the employee structure variable e1.

Accessing a member

A structure is a collection of members. All members are to be accessed using a single structure variable. So to access each member of the structure you have to use member operator (.) .

We can assign values to members of a structure in a number of ways. The members themselves are not variables. They should be linked to structure variables in order to make them meaningful members.

structure-variable.member

The following is an example fo r accessing member of a structure variable.

```
struct employee x;
 x. eno=10; /*place 10 into eno number*/ scanf("%f",&x.bs); strcpy(x.ename,"Santosh");
printf("%d %s %f",x.eno,x.ename,x.bs);
```

Once you use member operator to access a member of a structure, it is equivalent to a variable of the member type. That means x.eno in the above example is equal to any integer variable as eno is of integer type.

We can initializ e a structure variable by listing out values to be stored in members of the structure within braces after structure variable, while declaring the variable.

Structure Initialization

```
main()
{
struct st_record
{
int rno;
char name[20]; int weight; float height;
};
struct st_record st1={1, "Ashok",60,180.75}; struct st_record st2={2, "Balu",50,185.72};
...
...
}
```

C language does not permit the initialization of individual structure members within the template. The initialization must be done only in the declaration of the actua l variables.

Example: Program to demonstrate how to use structure

```
main()
{
struct employee
{
int eno;
char ename[20]; char eadd[100]; float bs;
};
struct employee x; int hra;
printf("Enter employee details: "); scanf("%d",&x.eno);
gets(x.ename); gets(x.eadd); scanf("%f",&x.bs);

if(x.bs>15000)
hra = 1500;
else
hra = 1000;

printf("Details of employee\n"); printf("Employee Number: %d\n",x.eno); printf("Employee Name: %s\n",x.ename); printf("Employee Address: %s\n",x.eadd); printf("Employee Basic Salary: %f\n",x.bs); printf("Employee Salary: %f\n",x.bs+hra);
}
```

Array of structures

Just like how we create an array of standard data types like int, float etc, you can create an array of structures also. An array of structures is an array where each element is a structure. You can declare an array of structures as follows:

struct employee edet[10];

Now edet is an array of 10 elements where each element can store data related to an employee. To access eno of 5 th element you would enter:

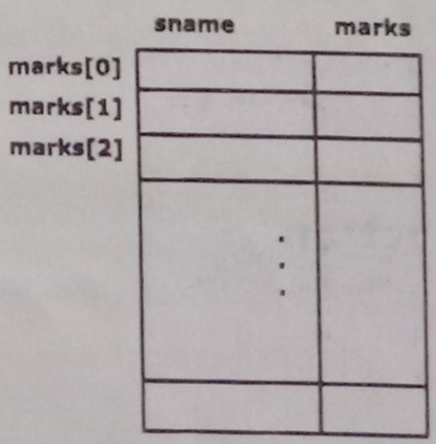
edet[5].eno=105;

Example: Program to take marks details of 10 students and display the name of the student with highest marks.

```
main()
{
struct smarks
{

char sname[20]; int marks;
};
struct smarks marks[10]; int i,pos;
for(i-0;i<10;i++)
{
printf("Enter student [%d] name: ",i); gets(marks[i].sname);
printf("Enter student [%d] marks: ",i); scanf("%d",&marks[i].marks);
}
pos=0; for(i=1;i<10;i++)
{
if(marks[i].marks > marks[pos].marks) pos=i;
}
printf("Student %s has got highest marks %d\n",
marks[pos].sname,marks[pos].marks);
}
```

An array of smarks structure

Structures and assignment



We can copy the value of one structure variable to another when both the structure variables belong to same structure type. For instance if you have two variables called x and y of structure employee then you can copy value of y to x using simple assignmen t operator.

x= y;

Copying one structure variable to another is equivalent to copying all members of one to another. Interesting fact about structure assignment is even members that are of string type are copied. Remember copying a string to another is not possible without using strcpy function. But when you copy one structure variable to another of the same type, the data (including strings) will be automatically copied.

Assume that x and y are variables of employee structure, copying y to x would copy even ename and eadd though they are strings. Because copying a structure variable to another structure variable is equivalent to copying the entire block (allocated to y) to another (block allocated to x) in memory.

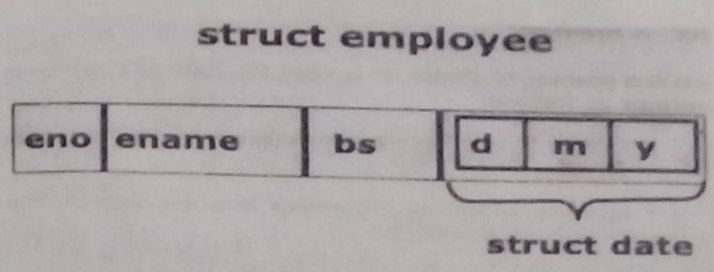
Pointer to Structure

A pointer can also point to structure just like how it can point to a standard data type. The process is similar to what we have already seen with standard data types.
struct employee *p; /*a pointer pointing to struct employee */
Now p can be made to point to e1 (a variable of struct employee) as follows: struct employee *p;
struct employee e1 = {1, "Srikanth","Vizag",65000 }; p=&e1; /* make p point to e1 */
To access a member of a structure variable using a pointer to structure, use arrow operators. Arrow operator is comb ination of hyphen and greater than symbol (->).
pointer-to-struct -> member
In order to use p, which is a pointer to struct, to access the data of e1, which is a struct employee variable, use arrow operator as follows:
printf("%d %f ", p-> eno, p -> bs);
In fact you can also use * operator to access a member through pointer. The following is the expression to access member x using pointer p.
x = (*p).x;
Remember parantheses around *p are required as without that C would take member operator (.) first and then indirection operator (*), as. has higher precedence than * . That means *p.x would be taken as *(p.x) by C and not as (*p).x.
Example: Program to i llustrate pointer to structure variables concept.

```
#include<stdio.h> main()
{
struct employee
{
int eno;
char ename[20]; float bs;
};

struct employee e,*emp; clrscr();
printf("Enter employee name: "); gets(e.ename);
printf("Enter employee number: "); scanf("%d",&e.eno);
printf("Enter employee salary: "); scanf("%f",&e.bs);
emp = &e;
printf("\nEmployee Details:\n"); printf("%d\t%s\t%0.2f",emp→eno,emp→ename,emp→
→bs); getch();
}
```

Output:

```
Enter employee name: BalaGuru Enter employee
number: 201 Enter employee salary: 27000
Employee Details:
201    BalaGuru    27000.00
```



Nested Structure

A structure may contain a member that is of another structure type. When a structure is used in another structure, it is called as nested structure.
Let us add a new member – date of joining (dj) to employee structure. DJ is consisting of members' day, month and year . So it is another structure. Here is the complete declaration of both the structures.

```
struct date
{
int  d; char m[15]; int year;
};
struct employee
{
int eno;
char ename[20]; float bs; struct date dj;
};
```

The figure above depicts the memory image of struct employee. It contains four members where member DJ again contains another three members.

Accessing nested structure

To access members of nested structure, first we need to access the member in the outer structure and then the member of nested structure. For example, the following example shows how to initialize and access nested members.

```
struct employee e1 = {1,"Ken",10000,{10,"October",2012}}; printf("Date : %d-%s-
%d",e1.dj.d,e1.dj.m,e1.dj.y);
```

Example: Program to illustrate structure within structure concept.

```
#include<stdio.h> main()
{
struct student
{
int rno;
char name[20]; int marks; struct dob
{
 int day;
}db;
}s[2];
int i; clrscr();
int d;
char mon[20]; int y;
for(i=0;i<2;i++)
{
printf("Enter Student [%d] Roll Number: ",i+1); scanf("%d",&s[i].rno);
fflush(stdin);
printf("Enter Student [%d] Name: ",i+1); gets(s[i].name);
printf("Enter Student [%d] Marks: ",i+1); scanf("%d",&s[i].marks);
printf("Enter Student [%d] Date of Birth: ",i+1); scanf("%d %s
%d",&s[i].db.d,s[i].db.mon,&s[i].db.y);
}
```

```
printf("\nStudents Details:\n"); for(i=0;i<2;i++)
{
printf("\n%d\t%s\t%d\t%d-%s-%d\n",s[i].rno,s[i].name,
s[i].marks,s[i].db.d,s[i].db.mon,s[i].db.y);

}
getch();
}
```

Unions

Unions are the concept borrowed from structures and follow the same syntax as structures. However there is a distinction between them in terms of storage. In structures each member has its own storage location, where as all the members of union use the same location i.e. all through a union may contain many members of different types; it can handle only one member at a time. In a union, mem ory is allocated to only the largest member of the union. All the members of the union will share the same area. That is why only one member can be active at a time.

Like structure, a union can be declared using keyword union as follows:

```
union item
{
int m; char c; float x;
}code;
```

This declares a variable code of type union item. The union contains 3 members, each with different data type. However we can use one of them at a time. This is due to the fact that only one location is allocated for a union variable, irrespective of its size.

1000	1001	1002	1003

The compiler allocated a piece of storage that is large enough to hold largest variable type in the union. In the declaration above, the member x requires 4 bytes which is largest among the members.

For example, if we use x in the above example, then we should not use m and c. If you try to use both at the same time, though C doesn't stop; you overwrite one da ta with another. If you access m after x and c, then the value available in union will be the value of m. If you display all the members the value of m will be the same and rest all will be garbage values.

A union is used for 2 purposes:

1) To conserve memor y by using the same area for two or more different variables.
2) To represent the same area of the memory in different ways.

Example: Program to illustrate concept of unions.

```
#include<stdio.h> #include<conio.h> main()
{
union number
{
```

```
int     n1;  float  n2; char name[20];
};
union number  x; clrscr();  printf("\nEnter Name: "); gets(x.name);
printf("Enter the value of n2: "); scanf("%f", &x.n2);
printf("Enter the value of n1: "); scanf("%d", &x.n1); printf("\n\nValue of n1 = %d",x.n1);
printf("\nValue of n2 = %f",x.n2); printf("\nName = %s",x.name); getch();
}
```

Output:

Enter Name: Zaheer
Enter the value of n2: 3.23 Enter the value of n1: 6
Value of n1 = 6
Value of n2 = 3.218751 Name = ♠

Enumeration

An enumeration is a user -defined data type consists of integral constants and each integral constant is give n a name. Keyword enum is used to defined enumerated data type.

enum type_name{ value1, value2,...,valueN }; Here, *type_name* is the name of enumerated data type or tag. And *value1*, *value2*,....,*valueN* are values of type *type_name* .

By default, *value1* will be equal to 0, *value2* will be 1 and so on but, the programmer can change the default value.

```
//Changingthedefaultvalueofenumelements enumsuit{
    club=0;
    diamonds= 10;
    hearts= 20;
    spades= 3;
};
```

Declaration of enumerated variable

Above code defines the type of the data but, no any variable is created. Variable of type enum can be created as:

```
enum boolean{ false;
    true;
};
enum boolean check;
```

Here, a variable check is declared which is of type enum boolean .

Example of enumerated type

```
#include <stdio.h>
enum { sunday, monday, tuesday, wednesday, thursday, friday, saturday};
int main(){
enum  today; today=wednesday; printf("%d day",today+1);
```

```
return 0;
}
```

Output

4 day

typedef

typedef is a keyword used in C language to assign alternative names to existing types. Its mostly used with user defined data types, when names of data types get slightly complicated. 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 type. Now this ulong identifier can be used to define unsigned long type variables.

```
ulong i, j ;
```

Application of typedef

typedef can be used to give a name to user defined data type as well. Lets see its use with structures.

```
typedef struct
{
type member1; type member2; type member3;
} type_name ;
```

Here type_name represents the stucture definition associated with it. Now this type_name can be used to declare a variable of this stucture type.

```
type_name t1, t2 ;
```

Example of structure definition using typede f

```
#include< stdio.h> #include< conio.h> #include< string.h> typedef struct employee
{
char   name[50]; intsalary;
} emp  ; void main( )
{
emp e1;

printf("\nEnter Employee record\n"); printf("\nEmployee name\t"); scanf("%s",e1.name);
printf("\nEnter Employee salary \t"); scanf("%d",&e1.salary); printf("\nstudent name is
%s",e1.name); printf("\nroll is %d",e1.salary); getch();
}
```

typedef and Pointers

typedef can be used to give an alias name to pointers also. Here we have a case in which use of typedef is beneficial during pointer declaration.

In Pointers * binds to the right and not the left.

```
int* x, y ;
```

By this declaration statement, we are actually declaring x as a pointer of type int, whereas y will be declared as a plain inte ger.

```
typedef int* IntPtr ;

IntPtr x, y, z;
```

But if we use typedef like in above example, we can declare any number of pointers in a single statement.

Binary Files, Random Accessing of Files
Enum, Typedef
Preprocessor Commands, Sample C Programs

* BINARY FILES

Input/output stream in file

Binary file is a file that is used to store any type of data. Binary files are typically used to store data records.

Text file is a file which contains readable characters and a few control characters like tab, new line etc. Example for text file is .txt , .c and etc. If the contents in a file are readable then it is a text file. Text file is terminated with a special character ^ Z or any other character depending on programmer's choice.

A Binary file is a file in which anything goes. Example for binary file is .exe, .obj, .dat and .com file. In these files, the contents are unreadable and show meaningless characters. Binary files are not terminated with any special character.

As Binary files don't have termination character, we have to specify whether you are dealing with text file or binary file. This is done using b (binary) qualifier in the modes of operation as (wb, rb, ab, wb+, rb+, ab+). The modes perform the same operation as like for text files.

Data file (.dat) is example of binary file. Data file is a collection of records. Each record is a structure. Whatever you have in a structure variable in memory that is copied to file as it is. So, if an integer is occupying two bytes t he same number of bytes even on the file and the exact memory image is copied to file.

Writing to binary file using fwrite()

The function fwrite can be used to write a block of memory into file. Whatever you have in that block of memory that is written in to file as it is.

```
int fwrite(void *address, int size, int noblocks, FILE *fp);
```

void *address	is starting address of the block that is to be written. fopen takes bytes from this address in memory.

int size	Size of each block in bytes. Use of sizeof operator to get exact number of bytes to be written while you are dealing with structures.
int noblocks	How many blocks are to be written? The size of each block is specified by int size. So, total number of bytes written will be size*noblocks
FILE *fp	Identifies the file into which the data is to be written.

fwrite returns the number of blocks written and NOT number of bytes written.
E.g.: fwrite(&s, sizeof(s), 1, fp);
In this, one structure is written to file. The starting address of the structure v ariable is taken using address operator &s. Size is taken using sizeof operator.
Reading binary files with fread()
fread() is same as fwrite() in syntax but does the opposite. It reads a block from file and places that block into memory (a structure varia ble).
int fread(void *address, int size, int noblocks, FILE *fp);
fread returns the number of blocks read. If fread() couldn't read record successfully it returns 0.
E.g.: fread(&s, sizeof(s), 1, fp);
This place's the data read from file into structur e.
Example: Program to illustrate fread() and fwrite() functions.

```
#include<stdio.h> struct student
{
int sno;
char sname[20]; char gender; int tfee;
int fpaid;
};
main()
{
FILE *fp;
struct student s; char c='y'; clrscr();
fp = fopen("student.dat","wb"); while(c=='y')
{
printf("Enter student details:\n"); scanf("%d",&s.sno);
fflush(stdin); gets(s.sname); fflush(stdin); s.gender=getchar();
scanf("%d %d",&s.tfee,&s.fpaid); fwrite(&s,sizeof(s),1,fp);
printf("Add student details...Press y or n: ");

fflush(stdin); scanf("%c",&c);
}
fclose(fp);
printf("\nStudent Details\n"); fp=fopen("student.dat","rb");
printf("SNO \t Name \t Gender \t TotalFees \t FeesPaid \t BalanceFee\n");
while((fread(&s,sizeof(s),1,fp))!=0)
```

```
{
printf("%d \t %s \t %4c \t\t %6d \t %5d \t\t %5d\n",
s.sno,s.sname,s.gender,s.tfee,s.fpaid,(s.tfee-s.fpaid));
}
fclose(fp); getch();
}
```

Hence if large amount of numerical data is to be stored in a disk file, using text mode may turn out to be inefficient. The solution is to open the file in binary mode and use those functions (fread( ) and fwrite()) which store the numbers in binary format. It means each number would occupy same number of bytes on disk as it occupies in memory.

RANDOM ACCESS TO FILES

So far we accessed the file sequentially, accessing from first byte to last byte. But it is also possible to access the file from a particular location.

When we are interested in accessing only a particular part of a file and not in reading other parts, it can be done with the concept of Random access. Random access allows you to move to a particular position in the file and perform input and output from that position.

When you access the file from beginning to end it is called Sequential Access. When you read/write file from a specific position, it is called Random Access. Random Access means moving the pointer of the file to the required location and read/write content at the location.

It can be achieved with the help of functions fseek(), ftell() and rewind() available in the I/O library.

ftell(): ftell takes file pointer and returns a number type long that corresponds to the current position. This function is useful in saving the current position of a file, which can be used later in the program. It takes the following form:

variable = ftell(file_ pointer);

E.g.: n = ftell(fp);

n would give the relative effect (in bytes) of current position. This means that n bytes have already been read or written.

fseek(): It is used to move the file pointer position to a desired location within the file. It takes the following form: fseek(file_pointer, offset, position);

file_pointer is a pointer to the file concerned, in which the pointer is to be moved.

The offset is a number or variable of type long. It is the number of bytes by which pointer should move f rom the position. Positive value moves forward and negative value moves backward.

The position indicates from where the movement should take place. It can take one of the following:

0 – Beginning of file 1 – Current position 2 – End of the file

Instead of numbers the symbolic constants – SEEK_SET, SEET_CUR, SEEK_END, which are declared in stdio.h can also be used.

When the operation is successful, fseek returns a zero. If we attempt to move the file pointer beyond file boundary an error occurs and fseek returns -1.

E.g.:

- fseek(fp, 100, 0) – Moves pointer fp forward to 100 th byte from the beginning of the file. The first byte is at index 0. fseek() skips first 100 bytes (0 -99) and places pointer at byte with index 100.
- fseek(fp, -25L, 1) - Moves p ointer fp backward by 25 byte from current position of the file.
- fseek(fp, -10L, 2) - Moves pointer fp backward by 100 bytes from end of the file.

/*Following snippet reads fifth record from student.dat*/
struct student s; FILE *fp;
/*open file in read binary mode*/ fp = fopen("student.dat","rb");
/*skip 4 record and move to 5th record*/ fseek(fp, sizeof(struct student)*4,0);
/*read a record*/ fread(&s,sizeof(struct student),1,fp);

EXAMPLES OF FSEEK & FTELL

FI.TXT contains the following content

H	**e**	**l**	**l**	**o**		**h**	**i**		**h**	**o**	**w**		**a**	**r**	**e**		**y**	**o**	**u**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	d	**e**	**a**	**r**		**s**	**t**	**u**	**d**	**e**	**n**	**t**	**s**	**EOF**					
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34					

Example: Program to print every 5th character from beginning in a given file.

```
#include<stdio.h> main()
{
int n=0; char c; FILE *fp; clrscr();
fp = fopen("f1.txt","r"); while((c=getc(fp))!=EOF)
{
printf("%c",c); n=n+5; fseek(fp,n,0);
}
getch();
}
```

Output: h oe e

Example: Program to print every 5th character from current position in a given file.

```
#include<stdio.h> main()
{
int n=0; char c; FILE *fp;
fp = fopen("f1.txt","r"); while((c=getc(fp))!=EOF)
{
printf("%c",c);
fseek(fp,5,1);
}
getch();
}
```

Output: hh ore

Example: Program to print conte nts of a given file in reverse

```
#include<stdio.h> main()
{
int n=1,i; char c; FILE *fp; clrscr();
fp = fopen("f1.txt","r");
```

```
while((c=getc(fp))!=EOF);

i=ftell(fp); fseek(fp,-n,2); while(i>=0)
{
c=getc(fp); printf("%c",c); n++;
i--;
fseek(fp,-n,2);
}
getch();
}
```

Output: stneduts raed uoy era woh ih olleh

Example: Program to p rint contents of file from end

```
#include<stdio.h> main()
{
int n=1,i; char c; FILE *fp; clrscr();
fp = fopen("f1.txt","r");
while((c=getc(fp))!=EOF); i=ftell(fp);
n=i;
fseek(fp,-(n),2); while(n!=0)
{
c=getc(fp); printf("%c",c); n--;
fseek(fp,-n,2);
}
getch();
}
```

Output: hello hi how are you dear students

Example: Program to print every 5th character from end

```
#include<stdio.h> main()
{
int n,i; char c; FILE *fp; clrscr();
fp = fopen("f1.txt","r");
while((c=getc(fp))!=EOF); i=ftell(fp);
n=1;
fseek(fp,-n,2); while(n<i)
{
c=getc(fp); printf("%c",c); n=n+5; fseek(fp,-n,2);
}
getch();
}
```

Output: suaoa l

rewind():

It takes a file pointer and resets the position to the start of the file.

Syntax: rewind(file_pointer);

For example, the statement

rewind(fp); n= ftell(fp);
would assign 0 to n because the file position has been set to the start of the file by rewind. The first byte in the file is numbered as 0, second as 1 and so on.
This function helps us in reading a file more than once, without having to close and open the file. Whenever a file is opened for reading or writing a rewind is alone implicitly.
This function can be used for the modes of operation with + as postfix to the modes (w, a, r, wb, rb, and ab).

Example: Program to illustrate rewind() function.

```
#include<stdio.h>
main()
{
FILE *f1;
char c; clrscr();
printf("Write data to file: \n");  printf("Specify @ to end the file reading \n\n"); f1 =
fopen("sample.txt","w+");
while((c=getchar())!='@') putc(c,f1);
rewind(f1);
printf("\nRead data from file: \n"); while((c=getc(f1))!=EOF)
printf("%c",c);

fclose(f1); getch();
}
```

With effect from the academic year 2022-2023
BCA151LC

Programming in C - Lab

1. Write programs using
   i. arithmetic,
   ii. logical,
   iii. bitwise
   iv. ternary operators.
2. Write programs simple control statements :
   i. Roots of a Quadratic Equation,
   ii. extracting digits of integers,
   iii. reversing digits ,
   iv. finding sum of digit ,
   v. printing multiplication tables,
   vi. Armstrong numbers,
   vii. checking for prime,
   viii. magic number,
3. Sin x and Cos x values using series expansion
4. Conversion of Binary to Decimal, Octal, Hexa and Vice versa
5. Generating a Pascal triangle and Pyramid of numbers
6. Recursion:
   i. Factorial,
   ii. Fibonacci,
   iii. GCD
7. Finding the maximum, minimum, average and standard deviation of given set of numbers using arrays
8. Reversing an array ,removal of duplicates from array
9. Matrix addition , multiplication and transpose of a square matrix .using functions
10. Functions of string manipulation: inputting and outputting string , using string functions such as strlen( ),strcat( ),strcpy( ).....etc
11. Writing simple programs for strings without using string functions.
12. Finding the No. of characters, words and lines of given text file
13. File handling programs : student memo printing

1. Write programs using

i.arithmetic

```
// C program to demonstrate syntax of binary arithmetic
// operators
#include <stdio.h>

int main()
{
        int a = 10, b = 4, res;

        // printing a and b
        printf("a is %d and b is %d\n", a, b);

        res = a + b; // addition
        printf("a + b is %d\n", res);

        res = a - b; // subtraction
        printf("a - b is %d\n", res);

        res = a * b; // multiplication
        printf("a * b is %d\n", res);

        res = a / b; // division
        printf("a / b is %d\n", res);

        res = a % b; // modulus
        printf("a %% b is %d\n", res);

        return 0;
}
```

Output

```
a is 10 and b is 4
a + b is 14
a - b is 6
a * b is 40
a / b is 2
a % b is 2
```

ii.logical,

```
// C program for Logical
// OR Operator
#include <stdio.h>

// Driver code
int main()
{
        int a = -1, b = 20;

        if (a > 0 || b > 0) {
                printf("Any one of the given value is "
                        "greater than 0\n");
        }
        else {
                printf("Both values are less than 0\n");
        }
        return 0;
}
```

Output

```
Any one of the given value is greater than 0
```

iii.bitwise

```
// C Program to demonstrate use of bitwise operators
#include <stdio.h>
int main()
{
        // a = 5(00000101), b = 9(00001001)
        unsigned char a = 5, b = 9;

        // The result is 00000001
        printf("a = %d, b = %d\n", a, b);
        printf("a&b = %d\n", a & b);

        // The result is 00001101
        printf("a|b = %d\n", a | b);

        // The result is 00001100
        printf("a^b = %d\n", a ^ b);

        // The result is 11111010
        printf("~a = %d\n", a = ~a);

        // The result is 00010010
        printf("b<<1 = %d\n", b << 1);

        // The result is 00000100
        printf("b>>1 = %d\n", b >> 1);

        return 0;

}
```

Output

```
a = 5, b = 9
a & b = 1
a | b = 13
a ^ b = 12
~a = -6
b << 1 = 18
b >> 1 = 4
```

iv.ternary operators.

```
// C program to find largest among two
// numbers using ternary operator

#include <stdio.h>

int main()
{
        int m = 5, n = 4;

        (m > n) ? printf("m is greater than n that is %d
> %d",
                                                m, n)
```

```
                    : printf("n is greater than m
that is %d > %d",
                                    n, m);

        return 0;
}
```

Output
m is greater than n that is 5 > 4

2 Write programs simple control statements :
i.Roots of a Quadratic Equation,

```
#include <math.h>
#include <stdio.h>
int main() {
   double a, b, c, discriminant, root1, root2, realPart,
imagPart;
   printf("Enter coefficients a, b and c: ");
   scanf("%lf %lf %lf", &a, &b, &c);

   discriminant = b * b - 4 * a * c;

   // condition for real and different roots
   if (discriminant > 0) {
      root1 = (-b + sqrt(discriminant)) / (2 * a);
      root2 = (-b - sqrt(discriminant)) / (2 * a);
      printf("root1 = %.2lf and root2 = %.2lf", root1,
root2);
   }

   // condition for real and equal roots
   else if (discriminant == 0) {
      root1 = root2 = -b / (2 * a);
      printf("root1 = root2 = %.2lf;", root1);
   }

   // if roots are not real
   else {
      realPart = -b / (2 * a);
      imagPart = sqrt(-discriminant) / (2 * a);
      printf("root1 = %.2lf+%.2lfi and root2 = %.2f-
%.2fi", realPart, imagPart, realPart, imagPart);
   }

   return 0;
}
```

Output
Enter coefficients a, b and c: 2.3
4
5.6
root1 = -0.87+1.30i and root2 = -0.87-1.30i

ii.extracting digits of integers

```
// C program of the above approach
#include <stdio.h>
#define MAX 100

// Function to print the digit of
// number N
void printDigit(int N)
{
        // To store the digit
        // of the number N
        int arr[MAX];
        int i = 0;
        int j, r;

        // Till N becomes 0
        while (N != 0) {

                // Extract the last digit of N
                r = N % 10;

                // Put the digit in arr[]
                arr[i] = r;
                i++;

                // Update N to N/10 to extract
                // next last digit
                N = N / 10;
        }

        // Print the digit of N by traversing
        // arr[] reverse
        for (j = i - 1; j > -1; j--) {
                printf("%d ", arr[j]);
        }
}

// Driver Code
int main()
{
        int N = 3452897;

        printDigit(N);
        return 0;
}
```

Output:
3 4 5 2 8 9 7

iii reversing digits

```
#include <stdio.h>

/* Iterative function to reverse digits of num*/
int reverseDigits(int num)
{
        int rev_num = 0;
        while (num > 0) {
                rev_num = rev_num * 10 + num % 10;
```

```
                num = num / 10;
        }
        return rev_num;
}

/*Driver program to test reverseDigits*/
int main()
{
        int num = 4562;
        printf("Reverse of no. is %d",
reverseDigits(num));

        getchar();
        return 0;
}
```

Output
Reverse of no. is 2654

iv.finding sum of digit ,

```
// C program to compute sum of digits in
// number.
#include <stdio.h>

/* Function to get sum of digits */
int getSum(int n)
{
        int sum = 0;
        while (n != 0) {
                sum = sum + n % 10;
                n = n / 10;
        }
        return sum;
}

// Driver code
int main()
{
        int n = 687;

        // Function call
        printf(" %d ", getSum(n));
        return 0;
}
```

Output
21

v.printing multiplication tables

```
// C program to Demonstrate the
// Multiplication table of a number
#include <stdio.h>
void print_table(int range, int num)
{
        // Declaring a variable mul to store the product.
        int mul;

        // For loop to calculate the Multiplication table.
        for (int i = 1; i <= range; i++) {
                // To store the product.
                mul = num * i;

                // Printing the Multiplication Table.
                printf("%d * %d = %d", num, i, mul);

                // Proceeding to the next line.
                printf("\n");
        }
}
// Driver code
int main()
{

        // The range of the
        // Multiplication table
        int range = 10;

        // The number to calculate the
        // Multiplication table
        int num = 5;

        // Calling the Function.
        print_table(range, num);

        return 0;
}
```

Output

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

vi.Armstrong numbers,

```
// C Program to Display Armstrong
// numbers between 1 to 1000
#include <math.h>
#include <stdio.h>

int main()
{
        int i, sum, num, count = 0;
        printf(
                "All Armstrong number between 1 and
1000 are:\n");
```

```
        // This loop will run for 1 to 1000
        for (i = 1; i <= 1000; i++) {
                num = i;
                // Count number of digits.
                while (num != 0) {
                        num /= 10;
                        count++;
                }
                num = i;
                sum = pow(num % 10, count)
                        + pow((num % 100 - num %
10) / 10, count)
                        + pow((num % 1000 - num %
100) / 100, count);
                // Check for Armstrong Number
                if (sum == i) {
                        printf("%d ", i);
                }
                count = 0;
        }
}
```

Output

All Armstrong number between 1 and 1000 are:

1 2 3 4 5 6 7 8 9 153 370 371 407

vii.checking for prime,

```
// C Program to check for prime number using Simple
Approach
#include <stdio.h>

// Function to check prime number
void checkPrime(int N)
{
        // initially, flag is set to true or 1
        int flag = 1;

        // loop to iterate through 2 to N/2
        for (int i = 2; i <= N / 2; i++) {

                // if N is perfectly divisible by i
                // flag is set to 0 i.e false
                if (N % i == 0) {
                        flag = 0;
                        break;
                }
        }

        if (flag) {
                printf("The number %d is a Prime
Number\n", N);
        }
        else {
                printf("The number %d is not a Prime
Number\n", N);
        }
        return;
}

// driver code
int main()
{
        int N = 546;

        checkPrime(N);

        return 0;
}
```

Output

The number 546 is not a Prime Number

viii.magic number

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

int main ()
{
   // declare integer variables
   int n, temp, rev = 0, digit, sum_of_digits = 0;

   printf (" Enter a Number: \n");
   scanf (" %d", &n); // get the number

   temp = n; // assign the number to temp variable

   // use while loop to calculate the sum of digits
   while ( temp > 0)
   {
      // extract digit one by one and store into the
sum_of_digits
      sum_of_digits = sum_of_digits + temp % 10; /* use
modulus symbol to get the remainder of each iteration by
temp % 10 */
      temp = temp / 10;
   }

   temp = sum_of_digits; // assign the sum_of_digits to
temp variable
   printf (" \n The sum of the digits = %d", temp);

   // get the reverse sum of given digits
   while ( temp > 0)
   {
      rev = rev * 10 + temp % 10;
      temp = temp / 10;
   }

   printf (" \n The reverse of the digits = %d", rev);


   printf (" \n The product of %d * %d = %d",
sum_of_digits, rev, rev * sum_of_digits);
   // use if else statement to check the magic number
```

```
    if ( rev * sum_of_digits == n)
    {
       printf (" \n %d is a Magic Number. ", n);
    }
    else
    {
       printf (" \n %d is not a Magic Number. ", n);
    }
    return 0;

}
```

Output

```
Enter a number
1729
The sum of the digits = 19
The reverse of the digits = 91
The product of 19 * 91 = 1729
1729 is a Magic Number.
```

3. Sin x and Cos x values using series expansion

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

int fac(int x)
{
        int i,fac=1;
        for(i=1;i<=x;i++)
                fac=fac*i;
        return fac;
}

int main()
{
        float x,Q,sum=0;
        int i,j,limit;

        printf("Enter the value of x of sinx series: ");
        scanf("%f",&x);

        printf("Enter the limit upto which you want to
expand the series: ");
        scanf("%d",&limit);

        Q=x;
        x = x*(3.1415/180);

        for(i=1,j=1;i<=limit;i++,j=j+2)
        {
                if(i%2!=0)
                {
                        sum=sum+pow(x,j)/fac(j);
                }
                else
                        sum=sum-pow(x,j)/fac(j);
        }
        printf("Sin(%0.1f): %f",Q,sum);
        return 0;
}
```

output

```
Enter the value of x of sinx series: 40
Enter the limit upto which you want to expand the series:
5
Sin(40.0): 0.642772
```

4. Conversion of Binary to Decimal, Octal, Hexa and Vice versa

```
#include <stdio.h>

/*
* Function to convert a given decimal number to its
* hexadecimal equivalent.
*/
int decimal_to_hex(long number, char *hexstr, int pos) {
 long i;
 int new_position, n;
 char hexchar;

 new_position = pos;

 if(number > 0) {
 i = number % 16;
 n = number / 16;
 new_position = decimal_to_hex(n, hexstr, pos);
 if(i >= 10) {
 switch(i) {
 case 10: hexchar = 'A'; break;
 case 11: hexchar = 'B'; break;
 case 12: hexchar = 'C'; break;
 case 13: hexchar = 'D'; break;
 case 14: hexchar = 'E'; break;
 case 15: hexchar = 'F'; break;
 }
 } else {
 hexchar = '0' + i;
 }
 hexstr[new_position++] = hexchar;
 hexstr[new_position] = '\0';
 }
 return new_position;
 }

 /*
 * Function to convert a given binary number to
 * its decimal equivalent.
 */
 int binary_to_decimal(int num) {
  int rem, base = 1, decimal_number = 0;
  while( num > 0) {
  rem = num % 10;
  if((rem == 0) || (rem == 1)) {
```

```
decimal_number = decimal_number + rem * base;
num = num / 10 ;
base = base * 2;
} else {
return -1; // Invalid binary number
}
}
return decimal_number;
}

/*
* Function to convert a given decimal number in to
* its octal equivalent.
*/
int decimal_to_octal(int number) {
 int octal_number = 0, rem, base = 1;

 while(number > 0) {
 rem = number % 8;
 octal_number = octal_number + rem * base;
 base = base * 10;
 number = number / 8;
 }
 return octal_number;
 }

 void main() {
 int binary_numer, decimal_number, octal_number;
 char hexstr[25] = "0";

 printf("Enter a binary number: ");
 scanf("%d", &binary_numer);

 printf("Given Binary number is: %d\n", binary_numer);

 decimal_number = binary_to_decimal(binary_numer);
 if(decimal_number == -1) {
 printf("\nInvalid binary number. Try again.");
 return;
 }
 printf("Its Decimal equivalent is: %d",
decimal_number);

 octal_number = decimal_to_octal(decimal_number);
 printf("\nIts octal equivalent is: %d", octal_number);

 decimal_to_hex(decimal_number, hexstr, 0);
 printf("\nIts Hexa Decimal equivalent is: %s", hexstr);
 }
```

Output

```
Enter a positive decimal number : 15
Binary number :: 1111
Octal number :: 17
Hexadecimal number :: F
```

5. Generating a Pascal triangle and Pyramid of numbers

```
// C program to print Pascal's Triangle
// using combinations in O(n^2) time
// and O(1) extra space function
#include <stdio.h>
void printPascal(int n)
{
        for (int line = 1; line <= n; line++) {
                for (int space = 1; space <= n - line;
space++)
                        printf("");
                // used to represent C(line, i)
                int coef = 1;
                for (int i = 1; i <= line; i++) {
                        // The first value in a line
                        // is always 1
                        printf("%4d", coef);
                        coef = coef * (line - i) / i;
                }
                printf("\n");
        }
}

// Driver code
int main()
{
        int n = 5;
        printPascal(n);
        return 0;
}
```

Output

```
    1
  1   1
1   2   1
  1   3   3   1
1   4   6   4   1
```

6. Recursion:

i.Factorial

```
// C program to find factorial of given number
#include <stdio.h>

// function to find factorial of given number
unsigned int factorial(unsigned int n)
{
        if (n == 0)
                return 1;
        return n * factorial(n - 1);
}

int main()
{
        int num = 5;
        printf("Factorial of %d is %d", num,
factorial(num));
        return 0;
```

```
}
```

Output
Factorial of 5 is 120

ii.Fibonacci

```
// Fibonacci Series using Space Optimized Method
#include <stdio.h>
int fib(int n)
{
        int a = 0, b = 1, c, i;
        if (n == 0)
                return a;
        for (i = 2; i <= n; i++) {
                c = a + b;
                a = b;
                b = c;
        }
        return b;
}

int main()
{
        int n = 9;
        printf("%d", fib(n));
        getchar();
        return 0;
}
```

Output
34

iii.GCD

```
#include <stdio.h>
int hcf(int n1, int n2);
int main() {
  int n1, n2;
  printf("Enter two positive integers: ");
  scanf("%d %d", &n1, &n2);
  printf("G.C.D of %d and %d is %d.", n1, n2, hcf(n1,
n2));
  return 0;
}

int hcf(int n1, int n2) {
  if (n2 != 0)
    return hcf(n2, n1 % n2);
  else
    return n1;
}
```

Output

Enter two positive integers: 366
60
G.C.D of 366 and 60 is 6.

7. Finding the maximum, minimum, average and standard deviation of given set of numbers using arrays

```
// C Program to Find Mean, Variance and Standard
Deviation of Array
#include <stdio.h>
#include <math.h>
int main()
{
   int A[] = {21, 23, 41, 62, 32};
   int n = sizeof(A)/sizeof(A[0]), i;
   float sum = 0, Var = 0, Mean, SD;
   printf("Array A :-->");
   for(i=0; i<n; i++)
     printf("%d  ", A[i]);
   for(i = 0;i < n; i++)      //to find the mean
   {
      sum = A[i] + sum;
   }
   Mean = sum/n;
   printf("\n\nSum :--> %0.2f", sum);
   printf("\nMean :--> %0.2f", Mean);
   for(i=0;i<n;i++)     //to find the variance
   {
      Var = Var + ((A[i] - Mean) * (A[i]- Mean)) / (n-1);
   }
   printf("\nVariance :--> %0.2f", Var);
   SD = sqrt(Var);
   printf("\nStandard Deviation :--> %0.2f", SD);
   return 0;
}
```

Output:

```
Array A :--> 21  23  41  62  32
Sum :--> 179.00
Mean :--> 35.80
Variance :--> 277.70
Standard Deviation :--> 16.66
```

8. Reversing an array ,removal of duplicates from array

```
// Iterative C program to reverse an array
#include<stdio.h>

/* Function to reverse arr[] from start to end*/
void rvereseArray(int arr[], int start, int end)
{
        int temp;
        while (start < end)
        {
                temp = arr[start];
                arr[start] = arr[end];
                arr[end] = temp;
                start++;
                end--;
        }
}
```

```
/* Utility that prints out an array on a line */
void printArray(int arr[], int size)
{
int i;
for (i=0; i < size; i++)
        printf("%d ", arr[i]);

printf("\n");
}

/* Driver function to test above functions */
int main()
{
        int arr[] = {1, 2, 3, 4, 5, 6};
        int n = sizeof(arr) / sizeof(arr[0]);
        printArray(arr, n);
        rvereseArray(arr, 0, n-1);
        printf("Reversed array is \n");
        printArray(arr, n);
        return 0;
}
```

Output :

1 2 3 4 5 6
Reversed array is
6 5 4 3 2 1

```
/*
 * C Program to remove duplicates from array using
nested for loop
 */

#include <stdio.h>
int main()
{
   int n, count = 0;
   printf("Enter number of elements in the array: ");
   scanf("%d", &n);
   int arr[n], temp[n];
   if(n==0)
   {
      printf("No element inside the array.");
      exit(0);
   }
   printf("Enter elements in the array: ");
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &arr[i]);
   }

   printf("\nArray Before Removing Duplicates: ");
   for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);

   // To store unique elements in temp after removing the
duplicate elements
   for (int i = 0; i < n; i++)
   {
      int j;
      for (j = 0; j < count; j++)
      {
       if (arr[i] == temp[j])
         break;
      }
      if (j == count)
      {
       temp[count] = arr[i];
       count++;
      }
   }

   printf("\nArray After  Removing Duplicates: ");
   for (int i = 0; i < count; i++)
      printf("%d ", temp[i]);

   return 0;
}
```

Output:
Enter number of elements in the array: 8
Enter elements in the array: 9 3 6 9 5 4 0 5

Array Before Removing Duplicates: 9 3 6 9 5 4 0 5
Array After Removing Duplicates: 9 3 6 5 4 0

9. Matrix addition , multiplication and transpose of a square matrix .using functions

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

// function to add two 3x3 matrix
void add(int m[3][3], int n[3][3], int sum[3][3])
{
 for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
    sum[i][j] = m[i][j] + n[i][j];
}

// function to subtract two 3x3 matrix
void subtract(int m[3][3], int n[3][3], int result[3][3])
{
 for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
    result[i][j] = m[i][j] - n[i][j];
}

// function to multiply two 3x3 matrix
void multiply(int m[3][3], int n[3][3], int result[3][3])
{
 for(int i=0; i < 3; i++)
  {
```

```
  for(int j=0; j < 3; j++)
  {
   result[i][j] = 0; // assign 0
   // find product
   for (int k = 0; k < 3; k++)
   result[i][j] += m[i][k] * n[k][j];
  }
 }
}

// function to find transpose of a 3x3 matrix
void transpose(int matrix[3][3], int trans[3][3])
{
 for (int i = 0; i < 3; i++)
  for (int j = 0; j < 3; j++)
   trans[i][j] = matrix[j][i];
}

// function to display 3x3 matrix
void display(int matrix[3][3])
{
 for(int i=0; i<3; i++)
 {
  for(int j=0; j<3; j++)
   printf("%d\t",matrix[i][j]);

  printf("\n"); // new line
 }
}

// main function
int main()
{
 // matrix
 int a[][3] = { {5,6,7}, {8,9,10}, {3,1,2} };
 int b[][3] = { {1,2,3}, {4,5,6}, {7,8,9} };
 int c[3][3];

 // print both matrix
 printf("First Matrix:\n");
 display(a);
 printf("Second Matrix:\n");
 display(b);

 // variable to take choice
 int choice;

 // menu-driven
 do
 {
  // menu to choose the operation
  printf("\nChoose the matrix operation,\n");
  printf("---------------------------\n");
  printf("1. Addition\n");
  printf("2. Subtraction\n");
  printf("3. Multiplication\n");
  printf("4. Transpose\n");
  printf("5. Exit\n");
  printf("---------------------------\n");
  printf("Enter your choice: ");
  scanf("%d", &choice);

  switch (choice) {
   case 1:
    add(a, b, c);
    printf("Sum of matrix: \n");
    display(c);
    break;
   case 2:
    subtract(a, b, c);
    printf("Subtraction of matrix: \n");
    display(c);
    break;
   case 3:
    multiply(a, b, c);
    printf("Multiplication of matrix: \n");
    display(c);
    break;
   case 4:
    printf("Transpose of the first matrix: \n");
    transpose(a, c);
    display(c);
    printf("Transpose of the second matrix: \n");
    transpose(b, c);
    display(c);
    break;
   case 5:
    printf("Thank You.\n");
    exit(0);
   default:
    printf("Invalid input.\n");
    printf("Please enter the correct input.\n");
  }
 }while(1);

 return 0;
}
```

Output:-

```
First Matrix:
5 6 7
8 9 10
3 1 2
Second Matrix:
1 2 3
4 5 6
7 8 9

Choose the matrix operation,
---------------------
1. Addition
2. Subtraction
```

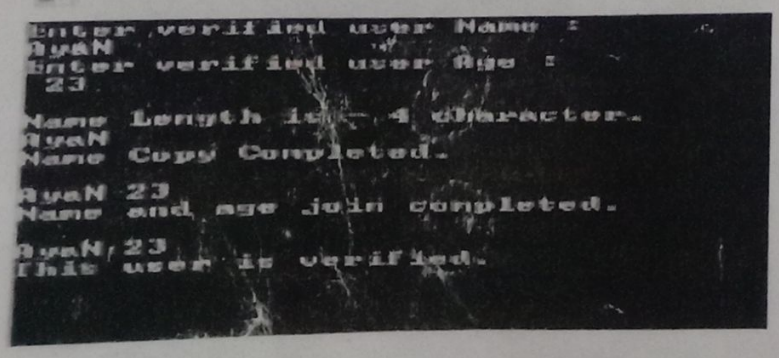
3. Multiplication
4. Transpose
5. Exit

Enter your choice: 1
Sum of matrix:
6 8 10
12 14 16
10 9 11

10. Functions of string manipulation: inputting and outputting string , using string functions such as strlen( ),strcat( ),strcpy( ).....etc

```
#include<stdio.h>
#include<strings.h>
main()
{
   int n;
   char name[20],age[20],temp[20],verify[5]="AyaN";

   puts("Enter verified user Name : ");
   gets(name);
   puts("Enter verified user Age : ");
   gets(age);

   n = strlen(name);
   printf("\nName Length is - %d character.\n",n);

   strcpy(temp,name);
   puts(temp);
   puts("Name Copy Completed.\n");

   strcat(name,age);
   puts(name);
   puts("Name and age join completed.\n");

   if(strcmp(verify,temp)=='\0')
   {
   puts(name);
   puts("This user is verified.");
   }
   else
   {
   puts(name);
   puts("This user is not verified.");
   }

getch();
}
```

Output :



11. Writing simple programs for strings without using string functions.

```
//C Program to Compare Two Strings Without Using
Library Function
#include<stdio.h>
int main() {
  char str1[30], str2[30];
  int i;

  printf("\nEnter two strings :");
  gets(str1);
  gets(str2);

  i = 0;
  while (str1[i] == str2[i] && str1[i] != '\0')
    i++;
  if (str1[i] > str2[i])
    printf("str1 > str2");
  else if (str1[i] < str2[i])
    printf("str1 < str2");
  else
    printf("str1 = str2");

  return (0);
}
```

Ouput
Enter two strings :hello
world
str1 < str2

12. Finding the No. of characters, words and lines of given text file

```
#include <stdio.h>
#include <stdlib.h>
int main(){
  FILE * file;
  char path[100];
  char ch;
  int characters, words, lines;
  file=fopen("counting.txt","w");
  printf("enter the text.press cntrl Z:
");
  while((ch = getchar())!=EOF){
    putc(ch,file);
  }
  fclose(file);
  printf("Enter source file path: ");
  scanf("%s", path);
  file = fopen(path, "r");
  if (file == NULL){
    printf("
Unable to open file.
");
    exit(EXIT_FAILURE);
  }
  characters = words = lines = 0;
```

```
  while ((ch = fgetc(file)) != EOF){
    characters++;
  if (ch == '
' || ch == '\0')
    lines++;
  if (ch == ' ' || ch == '\t' || ch == '
' || ch == '\0')
    words++;
  }
  if (characters > 0){
    words++;
    lines++;
  }
  printf("
");
  printf("Total characters = %d
", characters);
  printf("Total words = %d
", words);
  printf("Total lines = %d
", lines);
  fclose(file);
  return 0;
}
```

Output

When the above program is executed, it produces the following result −

```
enter the text.press cntrl Z:
Hi welcome to Tutorials Point
C programming Articles
Best tutorial In the world
Try to have look on it
All The Best
^Z
Enter source file path: counting.txt

Total characters = 116
Total words = 23
Total lines = 6
```

13. File handling programs : student memo printing

```
#include <stdio.h>
int main() {
        char name[50];
        int marks,i,n;
        printf("Enter number of students: ");
        scanf("%d",&n);
        FILE *fptr;
        fptr=(fopen("C:\\student.txt","w"));
        if(fptr==NULL) {
                printf("Error!");
                exit(1);
        }
        for (i=0;i<n;++i) {
                printf("For student%d\nEnter name:
",i+1);
                scanf("%s",name);
                printf("Enter marks: ");
                scanf("%d",&marks);
                fprintf(fptr,"\nName: %s \nMarks=%d
\n",name,marks);
        }
        fclose(fptr);
        return 0;
}
```

Nov 21/11 - 22, 23, 24

22/11 - 26, 25, 27

23/11 - 28, 29, 30

24/11 - [illegible], [illegible] 31, 32, 33

25/11 - 34, 35, 36

26/11 - 37, 38, 39