UNIT – V

SLO – 1 :

Initializing Structure, Declaring structure variable

What is a Structure?

- Structure is a collection of logically related data items of different data types grouped together under a single name.
- Structure is a user defined data type.

Examples:

  - Student name, roll number, and marks
  - Real part and complex part of a complex number

- Helps in organizing complex data in a more meaningful way
- The individual structure elements are called members

Defining a Structure

Syntax of Structure:

```
struct structure_name
     {
           data_type member1;
          data_type member2;


     };
```

- struct is a keyword.
- structure_name is a tag name of a structure.
- member1, member2 are members of structure.

Contd.

- The individual members can be ordinary variables, pointers, arrays, or other structures (any data type)
  - The member names within a particular structure must be distinct from one another
  - A member name can be the same as the name of a variable defined outside of the structure

- Once a structure has been defined, the individual structure-type variables can be declared as:

```
        struct tag var_1, var_2, …, var_n;
```

Example

- A structure definition

```
struct student {
        char name[30];
        int  roll_number;
        int  total_marks;
        char dob[10];
      };
```

- Defining structure variables:

$\underbrace{\texttt{struct student}}_{\text{A new data-type}}\ \texttt{a1, a2, a3,name;}$

Structure initialization

- Structure initialization
- Syntax:

```
struct structure_name structure_variable={value1, value2, …, valueN};
```

- Note: C does not allow the initialization of individual structure members within the structure definition template.

Structure initialization

```
struct student
{                                                  // stru1.c
char name[20];
int roll_no;
float marks;
char gender;
long int phone_no;
};
void main()
{
struct student st1={“ABC", 4, 79.5, 'M', 5010670};
clrscr(); printf("Name\t\t\tRoll No.\tMarks\t\tGender\tPhone No.");
printf("\n.........................................................................\n");

printf("\n %s\t\t %d\t\t %f\t%c\t %ld", st1.name, st1.roll_no, st1.marks, st1.gender,
st1.phone_no);
getch();
}
```

A Compact Form

- It is possible to combine the declaration of the structure with that of the structure variables:

```
struct tag {
            member 1;
            member 2;
            :
            member m;
         }  var_1, var_2,…, var_n;
```

- Declares three variables of type struct tag
- In this form, tag is optional

Declare and access structure variables

Declaration of structure:

- A structure variable declaration is similar to the declaration of variables of any other data type. It includes the following elements:

1) The keyword struct.
2) The structure tag name.
3) List of variable names separated by commas.
4) A terminating semicolon.

Example:

```
struct book
{
    char title[100];
    char author[50];
    int pages;
    float price;
} book1;

struct book book2;
```

Declare structure variable in two ways

We can declare structure variable in two ways:

1) Just after the structure body like book1.

2) With struct keyword and structure tag name like book2.

Accessing structure members:

- The following syntax is used to access the member of structure.

structure_variable.member_name

- structure_variable is a variable of structure and member_name is the name of variable which is a member of a structure.
- The "."(dot) operator or 'period operator' connects the member name to structure name.

Example:

book1.price represents price of book1.

Accessing structure members:

We can assign values to the member of the structure variable book1 as below,

```
strcpy(book1.title,”ANSI C”);
strcpy(book1.author,”Balagurusamy”);
book1.pages=250; book1.price=120.50;
```

We can also use scanf function to assign value through a keyboard.

```
scanf(“%s”,book1.title);
scanf(“%d”,&book1.pages);
```

Difference between Structure and Array

Array	Structure
An array behave like a built-in datatype all we have to do is to declare an array variable and use it.	First we have to design and declare a data structure before the variables of that type are declared and use.
An array is a collection of related data element of same type.	Structure can have elements of different type.
An array is derived datatype.	Structure is programmer defined.

Accessing a Structure

- The members of a structure are processed individually, as separate entities
  - Each member is a separate variable
- A structure member can be accessed by writing

  variable.member

  where variable refers to the name of a structure-type variable, and member refers to the name of a member within the structure
- Examples:

  a1.name, a2.name, a1.roll_number, a3.dob

Example:
//structure.c

```
#include<stdio.h>
#include<conio.h>
struct book
{
 char title[100];
 char author[50];
 int pages;
 float price;
};
void main()
{
 struct book book1;
 clrscr();
 printf("enter title, author name, pages and price of book");
 scanf("%s",book1.title);
 scanf("%s",book1.author);
 scanf("%d",&book1.pages);
 scanf("%f",&book1.price);

 printf("\nDetails of the book\n");
 printf("##################\n");
 printf("Book Title   : %s\n",book1.title);
 printf("Author Name  : %s\n",book1.author);
 printf("No. of Pages : %d\n",book1.pages);
 printf("Price        : %6.2f",book1.price);
 getch();
}
```

- book is structure whose members are title, author, pages and price.
- book1 is a structure variable.

Example: Complex number addition

```
void main()
{
    struct  complex
    {
        float  real;
        float  cmplex;
    }  a, b, c;

    scanf (“%f %f”, &a.real, &a.cmplex);
    scanf (“%f %f”, &b.real, &b.cmplex);

    c.real = a.real + b.real;
    c.cmplex = a.cmplex + b.cmplex;
    printf (“\n %f + %f j”, c.real, c.cmplex);
}
```

SLO – 2 :
Structure using typedef, Accessing members

Typedef

- Typedef is a keyword in the C language, it is used to define own identifiers that can be used in place of type specifiers such as int, float, and double.
- The names you define using typedef are not new data types, but synonyms for the data types or combinations of data types they represent.
- The name space for a typedef name is the same as other identifier.
- A typedef can be used to simplify the declaration for a structure.

Example:-

```
typedef struct
{
char firstname[20];
char lastname[20];
int no;
} student;
```

Now we can use student directly to define variables of student type without using struct keyword. Following is the example:-

student student_a;

Example:-

- It is also possible to use type definitions with structures. The name of the type definition of a structure is usually in uppercase letters.

```
#include<stdio.h>
 typedef struct telephone                    //typedef.c
 {
 char *name;
 int number;
 }TELEPHONE;
 int main()
 {
 TELEPHONE index;
 index.name=”xyz”;
 index.number=12345;
 printf(“Name : %s\n”, index.name);
 printf(“Telephone number: %d\n”,index.number);
 return 0;
}
```

SLO – 1 :
Nested structure Accessing elements in a structure array

Nested structure

- A structure that contains another structure as a member variable is known as nested structure or structure within a structure.
- Structure which is part of other structure must be declared before the structure in which it is used.

Example

```
#include<stdio.h>
#include<conio.h>
struct address
{
 char add1[50];
 char add2[50];
 char city[25];
};
struct employee
{
 char name[100];
 struct address a;
 int salary;
};
```

Example

```
void main()
{
 struct employee e;                          //nested7.c
 clrscr();
 printf("enter name,address1 and address2,city,salary\n");
 scanf("%s",e.name);
 scanf("%s",e.a.add1);
 scanf("%s",e.a.add2);
 scanf("%s",e.a.city);
 scanf("%d",&e.salary);
 printf("detail of employaee:\n");
 printf("%s\n",e.name);
 printf("%s\n",e.a.add1);
 printf("%s\n",e.a.add2);
 printf("%s\n",e.a.city);
 printf("%d\n",e.salary);
 getch();
}
```

SLO – 2 :
Array of structures Accessing elements in a structure array

Array of structures

- Array of structure mean collection of structures.
- Array storing different type of structure of variables.
- As we have an array of basic data types, same way we can have an array variable of structure.

Following example shows how an array of structure can be used

```
#include<stdio.h>
#include<conio.h>
struct result                    //arrstu.c
{
 char name[10];
 int rollno;
 int sub1;
};
void main()
{
 struct result r[2];
 int i;
 clrscr();
 printf("enter detail of student :");
```

Following example shows how an array of structure can be used

```
for(i=0;i<2;i++)
 {
 printf("\nenter name,roll no,sub1 marks");
 scanf("%s",r[i].name);
 scanf("\n%d",&r[i].rollno);
 scanf("\n%d",&r[i].sub1);
 }
 printf("\n detail of student:\n");
 printf("\n####################################");
 printf("\nStudent name  roll no  sub1 marks");
 printf("\n####################################\n");
 for(i=0;i<2;i++)
 {
 printf("%s",r[i].name);
 printf("\t\t%d\t",r[i].rollno);
 printf("%d\n",r[i].sub1);
 }
 printf("####################################");
 getch();
}
```

SLO – 1 :
Passing Array of structure of function

Given an array of structure and we have to pass it to the function in C.

structure declaration is:

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

Here,

- exam is the structure name
- roll, marks and name are the members of the structure/variable of the structure

array of structure

- **Here is the function that we are using in the program,**

void structfun(struct exam obj1);

Here,

- void is the returns type of the function i.e. function will not return any value.
- structfun is the name of the function.
- struct exam obj1 is the object of exam structure - while declaring a function we must have to use struct keyword with the structure name.

array of structure

- **Structure object/variable declaration is:**

```
struct exam obj[2];
```

obj is an array of structure for 2 structures.

array of structure

- **Assigning values to the structure variables (array of structure variables) inside the main()**

```
// assign values using the object 1
   obj[0].marks = 35;
   obj[0].roll = 10;
   strcpy(obj[0].name , “RAMKUMAR");

   // assign values using the object 1
   obj[1].marks = 75;
   obj[1].roll = 11;
   strcpy(obj[1].name , "BALA");
```

array of structure

- **Function calling statement,**

```
structfun(obj);
```

Here,

obj is an array of structure.

Program to pass an array of structures to a function in C (STOF.C)

```
/*
C program to pass an arrays of structures
to a function
*/

#include <stdio.h>

// Declare a global structure since we need to pass
// it to a function
struct exam
{
   int roll;
   int marks;
   char name[20];
};
```

structures to a function

```
// array of structure object
struct exam obj[2];

 // declaration of the function
void structfun(struct exam *obj);

// function to print structure elements switch
// two different objects
void structfun(struct exam *obj)
{
   //Values using the object 1
   printf("\nName is : %s",obj[0].name);
   printf("\nRoll No. is : %d",obj[0].roll);
   printf("\nMarks are : %d",obj[0].marks);

   printf("\n");

   // Values using the object 2
   printf("\nName is : %s",obj[1].name);
   printf("\nRoll No. is : %d",obj[1].roll);
   printf("\nMarks are : %d",obj[1].marks);
}
```

structures to a function

```
// main function
int main()
{
   clrscr();
   // assign values using the object 1
   obj[0].marks = 35;
   obj[0].roll = 10;
   strcpy(obj[0].name , "RAMKUMAR");

   // assign values using the object 1
   obj[1].marks = 75;
   obj[1].roll = 11;
   strcpy(obj[1].name , "BALA");

   // Passing structure to Function
   structfun(obj);
   getch();
   return 0;
}
```

SLO – 2 :
Array of pointers to a structures

Array of pointers to a structures

```
struct tStuff
 { int x;
   int y;
   char z[100];
  }
struct tStuff *MyStuff[500];
```

- The above code declares MyStuff to be an array of pointers to tStuff structs, with enough room for 500 such pointers.
- Thus MyStuff[0] is a pointer to a tStruct struct, as is MyStuff[1], etc.

SLO – 1 :
Bit Manipulation to structure and Pointer to structure

Bit manipulation to structure

Why bit manipulation?

- Pieces of data shorter than a byte.
- C language is very efficient.
- Bit Fields allow the packing of data in a structure.
- Packing several objects into a machine word.
- E.g. 1 bit flags can be compacted.
- Flags can be used in order to store the Boolean values**(T / F)**.

Applications

- Error detection and correction algorithms,
- Data compression,
- Encryption algorithms,
- Optimization.

Bit Manipulation to structure

Syntax : Bit Manipulation

```
Struct databits
{
    int b1 : 1;
    int b2 : 1;
    int b3 : 1;
    int b4 : 4;
    int b5 : 9;
}data1;
```

How to access the Individual Bits in structure ?

data1.b1

data1.b2

data1.b3

data1.b4

data1.b5

SLO – 2 :
Union Basic and declaration

Union

- A union is a user-defined type similar to structs in C except for one key difference.
- Structs allocate enough space to store all its members wheres unions allocate the space to store only the largest member.

How to define a union?

- We use the union keyword to define unions. Here's an example:

```
union car
{
 char name[50];
 int price;
};
```

The above code defines a derived type union car.

Create union variables

- When a union is defined, it creates a user-defined type.
- However, no memory is allocated. To allocate memory for a given union type and work with it, we need to create variables.

Here's how we create union variables.

```
union car
{
 char name[50];
 int price;
};
int main()
{
 union car car1, car2, *car3;
 return 0;
}
```

Create union variables

- Another way of creating union variables is:

```
union car
{
  char name[50];
  int price;
} car1, car2, *car3;
```

In both cases, union variables car1, car2, and a union pointer car3 of union car type are created.

Access members of a union

- We use the . operator to access members of a union. To access pointer variables, we use also use the -> operator.
- In the above example,

- To access price for car1, car1.price is used.
- To access price using car3, either (*car3).price or car3->price can be used.

Difference between unions and structures (union2.C)

- Let's take an example to demonstrate the difference between unions and structures:

```
#include <stdio.h>
union unionJob
{
  //defining a union
  char name[32];
  float salary;
  int workerNo;
} uJob;

struct structJob
{
  char name[32];
  float salary;
  int workerNo;
} sJob;

int main()
{
  printf("size of union = %d bytes", sizeof(uJob));
  printf("\nsize of structure = %d bytes", sizeof(sJob));
  return 0;
}
```

Why this difference in the size of union and structure variables?

- Here, the size of sJob is 38 bytes because

- the size of name[32] is 32 bytes
- the size of salary is 4 bytes
- the size of workerNo is 2 bytes
- However, the size of uJob is 32 bytes. It's because the size of a union variable will always be the size of its largest element. In the above example, the size of its largest element, (name[32]), is 32 bytes.

- With a union, all members share the same memory.

Example: Accessing Union Members (union3.c)

```
#include <stdio.h>
union Job {
   float salary;
   int workerNo;
} j;

int main() {
   j.salary = 12.3;

   // when j.workerNo is assigned a value,
   // j.salary will no longer hold 12.3
   j.workerNo = 100;

   printf("Salary = %.1f\n", j.salary);
   printf("Number of workers = %d", j.workerNo);
   return 0;
}
```

SLO – 1 :

Accessing Union Members, Pointers to Union

Accessing Union Members

- Union in C is same like a structure except the allocation of memory for its members.
- Union is a special data type available in C that enables you to store different Data types in the same memory location
- Union occupies lower memory space as compared to structure.
- We can access single member of union at a time, You can define a union with many members, but only one member can contain a value at any given time
- In one word union is a collection of different data types which are grouped together.
- Each element in union is called member.

Example

Example:-

```
 Union emp
{
           int id;
           char name[10];
           float sal;
 }x;
```

Syntax of Using Normal Variable in Union:-

```
union tag_name
{
  data type var_name1;
  data type var_name2;
  data type var_name3;
 }
```

Example:-

```
 union student
{
 int mark;
 char name[11];
 float average;
 }
```

Accessing Union members

We can access only single union member at a time and to access we use two operators:

1. DOT operator

Syntax: union _variable. member

```
x.id;
x.name;
x.sal;
```

2. ARROW operator

Syntax: union variable ->member

```
x ->id;
x->name;
x->sal;
```

Pointers to Union

- Here, we are going to learn about creating union pointer, changing, and accessing union members using the pointer.
- Pointer to union can be created just like other pointers to primitive data types.

Consider the below union declaration:

```
union number{
   int a;
   int b;
};
```

Here, a and b are the members of the union number.

Pointers to Union

Union variable declaration:

union number n;
Here, n is the variable to number.

Pointer to union declaration and initialization:

union number *ptr = &n;
Here, ptr is the pointer to the union and it is assigning with the address of the union variable n.

Accessing union member using union to pointer:

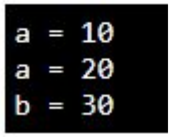
Syntax:
```
pointer_name->member;
```

Example:
```
// to access members a and b using ptr
ptr->a;
ptr->b;
```

Example : union to pointer

```
#include <stdio.h>
int main()
{
   // union declaration
   union number {
      int a;
      int b;
   };
   // union variable declaration
   union number n = { 10 }; // a will be assigned with 10
   // pointer to union
   union number* ptr = &n;
   printf("a = %d\n", ptr->a);
   // changing the value of a
   ptr->a = 20;
   printf("a = %d\n", ptr->a);
// changing the value of b
   ptr->b = 30;
   printf("b = %d\n", ptr->b);
   return 0;
}
```

Output:



```
a = 10
a = 20
b = 30
```

SLO – 2 :
Dynamic Memory Allocation: malloc(), realloc(),calloc(),free Functions

Dynamic Memory Allocation

- Dynamic Memory Allocation is manual allocation and freeing of memory according to your programming needs.
- Dynamic memory is managed and served with pointers that point to the newly allocated memory space in an area which we call the heap.
- Now you can create and destroy an array of elements dynamically at runtime without any problems.
- To sum up, the automatic memory management uses the stack, and the C Dynamic Memory Allocation uses the heap.
- The <stdlib.h> library has functions responsible for Dynamic Memory Management.

Function Purpose :
malloc(), realloc(),calloc(),free()

malloc() - Allocates the memory of requested size and returns the pointer to the first byte of allocated space.

calloc() - Allocates the space for elements of an array. Initializes the elements to zero and returns a pointer to the memory.

realloc() - It is used to modify the size of previously allocated memory space.

free() - Frees or empties the previously allocated memory space.

malloc() Function

- The C malloc() function stands for memory allocation. It is a function which is used to allocate a block of memory dynamically.
- It reserves memory space of specified size and returns the null pointer pointing to the memory location.
- The pointer returned is usually of type void. It means that we can assign C malloc() function to any pointer.

Syntax of malloc() Function:

```
ptr = (cast_type *) malloc (byte_size);
```

Here,

- ptr is a pointer of cast_type.
- The C malloc() function returns a pointer to the allocated memory of byte_size.

Example of malloc():

Example: ptr = (int *) malloc (50)

- When this statement is successfully executed, a memory space of 50 bytes is reserved.
- The address of the first byte of reserved space is assigned to the pointer ptr of type int.

free() Function

- The memory for variables is automatically deallocated at compile time.
- In dynamic memory allocation, you have to deallocate memory explicitly. If not done, you may encounter out of memory error.
- The free() function is called to release/deallocate memory in C.
- By freeing memory in your program, you make more available for use later.

Example : free()

```
#include <stdio.h>
int main() {
int* ptr = malloc(10 * sizeof(*ptr));
if (ptr != NULL){
  *(ptr + 2) = 50;
  printf("Value of the 2nd integer is %d",*(ptr + 2));
}
free(ptr);
}
```

Output

```
 Value of the 2nd integer is 50
```

calloc() Function

- The C calloc() function stands for contiguous allocation. This function is used to allocate multiple blocks of memory.
- It is a dynamic memory allocation function which is used to allocate the memory to complex data structures such as arrays and structures.
- Malloc() function is used to allocate a single block of memory space while the calloc() in C is used to allocate multiple blocks of memory space.
- Each block allocated by the calloc() function is of the same size.

calloc() Function

Syntax of calloc() Function:

```
ptr = (cast_type *) calloc (n, size);
```

- The above statement is used to allocate n memory blocks of the same size.
- After the memory space is allocated, then all the bytes are initialized to zero.
- The pointer which is currently at the first byte of the allocated memory space is returned.
- Whenever there is an error allocating memory space such as the shortage of memory, then a null pointer is returned.

realloc() Function

- Using the C realloc() function, you can add more memory size to already allocated memory. It expands the current block while leaving the original content as it is.
- realloc() in C stands for reallocation of memory.
- realloc() can also be used to reduce the size of the previously allocated memory.

Syntax of realloc() Function:

```
ptr = realloc (ptr,newsize);
```

- The above statement allocates a new memory space with a specified size in the **variable newsize.**
- After executing the function, the pointer will be returned to the first byte of the memory block.
- The new size can be **larger or smaller** than the previous memory.

SLO – 1 & 2 :
Allocating dynamic array, multidimensional array using Dynamic memory allocation

Allocating dynamic array

- We can create both static and dynamic array in C. These arrays can be one dimensional or multiple dimensional.
- In statically allocated array problem is that we have to specify the size of the array before the compilation.
- So the problem is generated when we don’t know how much size of the array required ahead of time.
- We can resolve these issues using dynamic memory allocation.

Allocating dynamic array

- The advantage of a dynamically allocated array is that it is allocated on the heap at runtime.

- In the below program, I am using malloc to allocate the dynamic memory for the 1D and 2D array.

Syntax of malloc in C

```
void * malloc (size_t size);
```

Parameters

- size ==> This is the size of the memory block, in bytes.

Return Value:

- Returns a **pointer** to the allocated memory, if enough memory is not available then it returns NULL.

2D array using the dynamic memory allocation

In C language like the 1D array, we can also create the 2D array using the dynamic memory allocation at runtime. In below, listing some generic steps to create the 2D array using the pointers.

Steps to creating a 2D dynamic array in C using pointer to pointer

- Create a pointer to pointer and allocate the memory for the row using malloc().

```
int ** piBuffer = NULL;
piBuffer = malloc( nrows * sizeof(int *));
```

- Allocate memory for each row-column using the malloc().

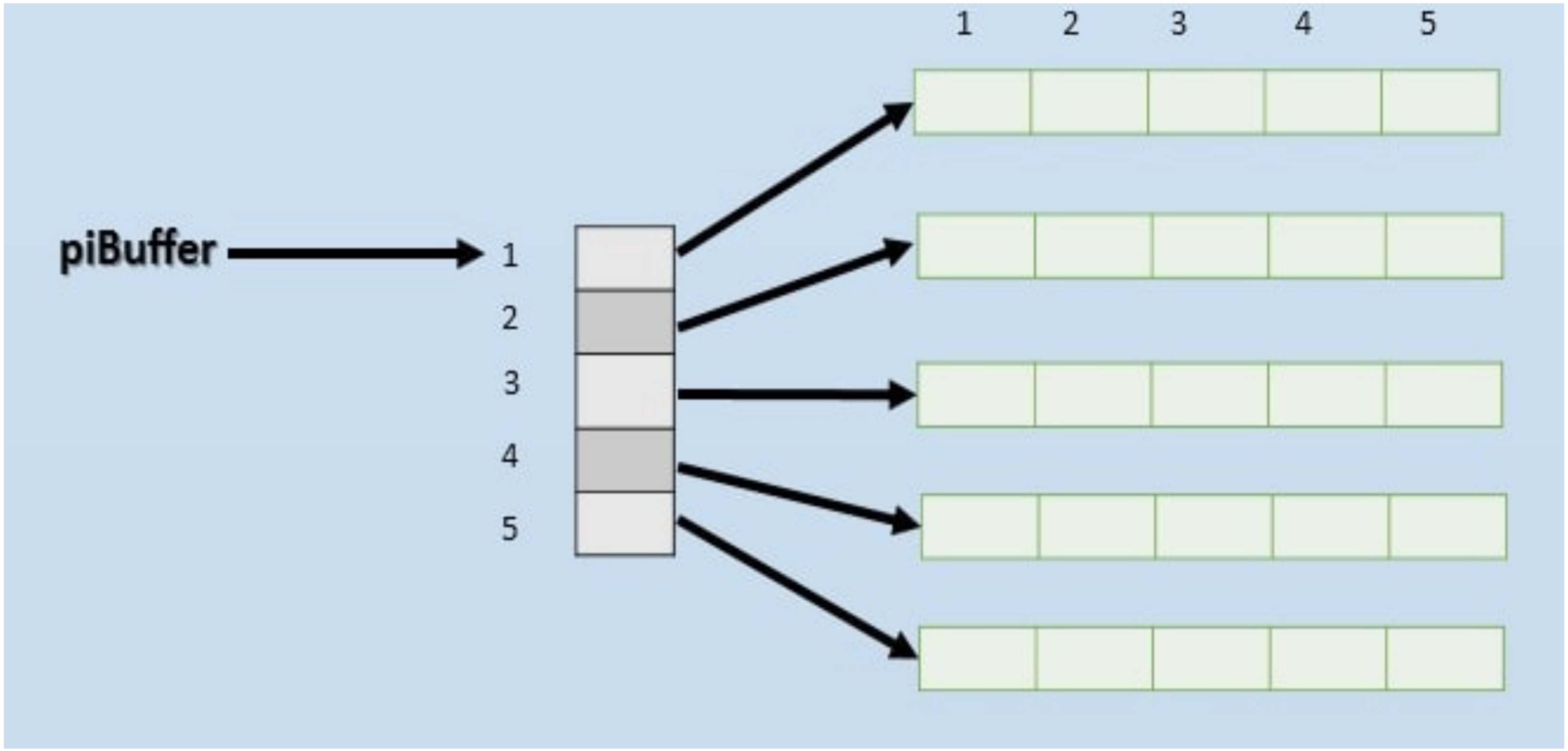
```
for(i = 0; i < nrows; i++)
{
piBuffer[i] = malloc( ncolumns * sizeof(int));
}
```

- If each row does not have the same number of columns then allocate memory for each row individually.

```
piBuffer[0] = malloc( ncolumns * sizeof(int));
piBuffer[1] = malloc( ncolumns * sizeof(int));
piBuffer[n] = malloc( ncolumns * sizeof(int));
```

2D array using the dynamic memory allocation

Let’s see the below picture where I am creating a 5×5 2D array using the dynamic memory allocation.



SLO – 1 :
file : opening, defining, closing, file Modes, File Types

WHAT IS FILE?

File is a collection of bytes that is stored on secondary storage devices like disk. There are two kinds of files in a system. They are,

1. Text files (ASCII)
2. Binary files

- Text files contain ASCII codes of digits, alphabetic and symbols.
- Binary file contains collection of bytes (0's and 1's). Binary files are compiled version of text files.

BASIC FILE OPERATIONS IN C PROGRAMMING:

There are 4 basic operations that can be performed on any files in C programming language. They are,

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

MODE OF OPERATIONS PERFORMED ON A FILE IN C LANGUAGE

- There are many modes in opening a file. Based on the mode of file, it can be opened for reading or writing or appending the texts. They are listed below.
- **r** – Opens a file in **read mode** and sets pointer to the first character in the file. It returns null if file does not exist.
- **w** – Opens a file in **write mode**. It returns null if file could not be opened. If file exists, data are overwritten.
- **a** – Opens a file in **append mode**. It returns null if file couldn't be opened.
- **r+** – Opens a file for **read and write mode** and sets pointer to the first character in the file.
- **w+** – opens a file for **read and write mode** and sets pointer to the first character in the file.
- **a+** – Opens a file for **read and write mode** and sets pointer to the first character in the file. But, it **can't modify existing contents**.

Let us see the syntax for each of the file operations in a table:

File operation	Declaration & Description
fopen() – To open a file	**Declaration:** **FILE *fopen (const char *filename, const char *mode)** fopen() function is used to open a file to perform operations such as reading, writing etc. In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist. **FILE *fp;** **fp=fopen (“filename”, ”‘mode”);** **Where,** fp – file pointer to the data type “FILE”. filename – the actual file name with full path of the file. mode – refers to the operation that will be performed on the file. **Example: r, w, a, r+, w+ and a+.**

File operations

fclose() – To close a file	Declaration:
	int **fclose**(FILE *fp);
	fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as below.
	fclose (fp);

fgets() – To read a file	Declaration: char ***fgets**(char *string, int n, FILE *fp) fgets function is used to read a file line by line. In a C program, we use fgets function as below. **fgets** (buffer, size, fp); where, buffer – buffer to put the data in. size – size of the buffer fp – file pointer
fprintf() – To write into a file	**Declaration:** **int fprintf(FILE *fp, const char *format, …);fprintf()** function writes string into a file pointed by fp. In a C program, we write string into a file as below. fprintf (fp, “some data”); or fprintf (fp, “text %d”, variable_name);

How to Create a File?

- Whenever you want to work with a file, the first step is to create a file. A file is nothing but space in a memory where data is stored.
- To create a file in a 'C' program following syntax is used,

```
FILE *fp;
fp = fopen ("file_name", "mode");
```

- In the above syntax, the file is a data structure which is defined in the standard library.
- fopen is a standard function which is used to open a file.
- If the file is not present on the system, then it is created and then opened.
- If a file is already present on the system, then it is directly opened using this function.
- fp is a file pointer which points to the type file.

Example:

```
#include <stdio.h>
int main()
{
FILE *fp;
fp  = fopen ("data.txt", "w");
}
```

Output:

File is created in the same folder where you have saved your code.

Example:

You can specify the path where you want to create your file

```
#include <stdio.h>
int main() {
FILE *fp;
fp  = fopen ("D://data.txt", "w");
}
```

How to Close a file

- One should always close a file whenever the operations on file are over. It means the contents and links to the file are terminated. This prevents accidental damage to the file.

- 'C' provides the fclose function to perform file closing operation. The syntax of fclose is as follows,

fclose (file_pointer);

Example:

```
FILE *fp;
fp  = fopen ("data.txt", "r");
fclose (fp);
```

How to Close a file

- The fclose function takes a file pointer as an argument. The file associated with the file pointer is then closed with the help of fclose function. It returns 0 if close was successful and EOF (end of file) if there is an error has occurred while file closing.

- After closing the file, the same file pointer can also be used with other files.

- In 'C' programming, files are automatically close when the program is terminated. Closing a file manually by writing fclose function is a good programming practice

SLO – 2 :
Writing contents into a file

Writing to a File

- In C, when you write to a file, newline characters '\n' must be explicitly added.
- The stdio library offers the necessary functions to write to a file:
- **fputc(char, file_pointer)**: It writes a character to the file pointed to by file_pointer.
- **fputs(str, file_pointer)**: It writes a string to the file pointed to by file_pointer.
- **fprintf(file_pointer, str variable_lists)**: It prints a string to the file pointed to by file_pointer. The string can optionally include format specifiers and a list of variables variable_lists.

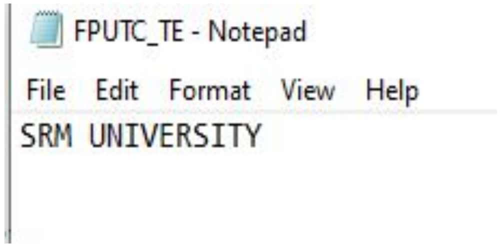
Example :fprintf(fptr, “SRM UNIVERSITY\n");

Example: fputc() Function

(fputfile.c)

```
#include <stdio.h>
int main() {
    int i;
    FILE * fptr;
    char fn[50];
    char str[] = "SRM UNIVERSITY\n";
    fptr = fopen("fputc_test.txt", "w"); // "w" defines "writing mode"
    for (i = 0; str[i] != '\n'; i++)
        {
        /* write to file using fputc() function */
        fputc(str[i], fptr);
        }
      fclose(fptr);
    return 0;
   }
```

Output:



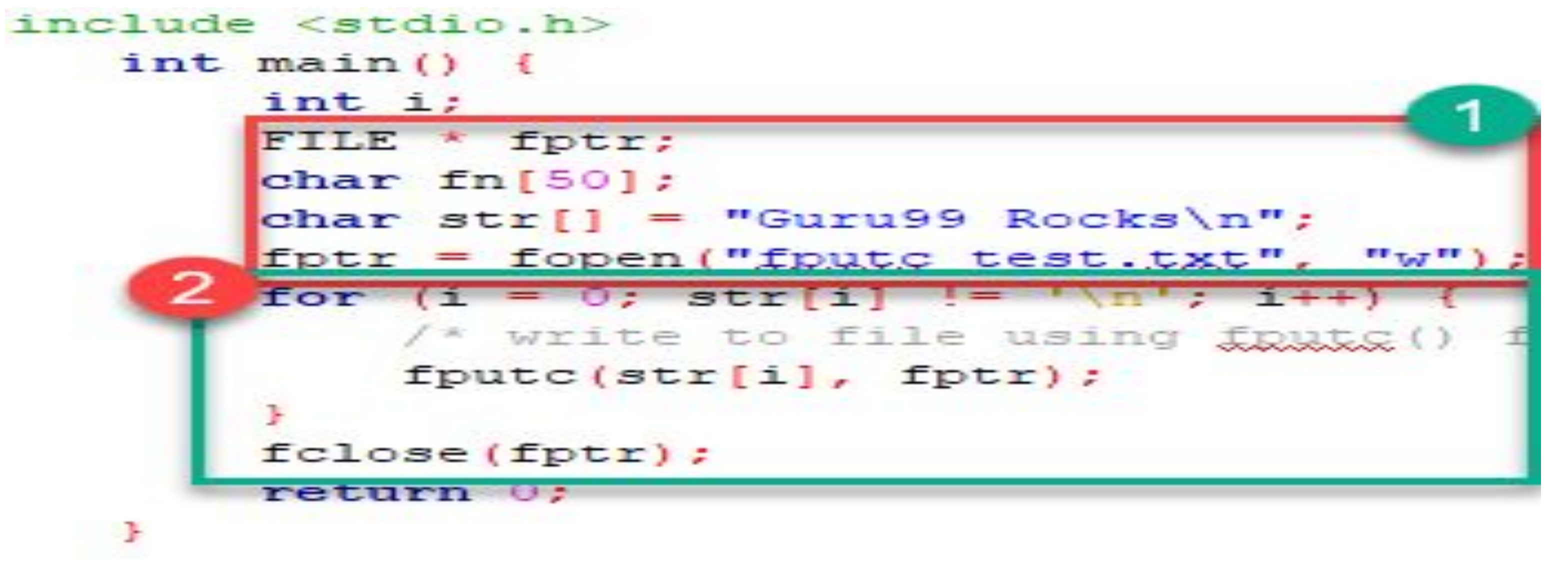
```
SRM UNIVERSITY
```

Example: fputc() Function

- Program writes a single character into the **fputc_test.txt** file until it reaches the next line symbol "\n" which indicates that the sentence was successfully written.
- The process is to take each character of the array and write it into the file.

Example: fputc() Function

1. Program, we have created and opened a file called fputc_test.txt in a write mode and declare our string which will be written into the file.
2. We do a character by character write operation using for loop and put each character in our file until the "\n" character is encountered then the file is closed using the fclose function.



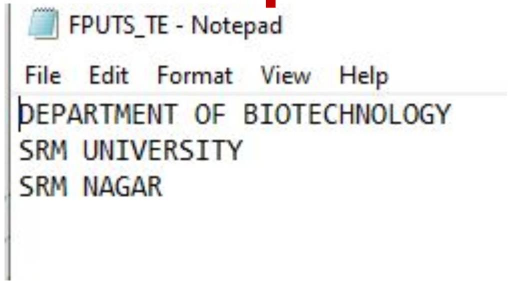
```
include <stdio.h>
    int main() {
        int i;
        FILE * fptr;
        char fn[50];
        char str[] = "Guru99 Rocks\n";
        fptr = fopen("fputc_test.txt", "w");
        for (i = 0; str[i] != '\n'; i++) {
            /* write to file using fputc() f
            fputc(str[i], fptr);
        }
        fclose(fptr);
        return 0;
    }
```

fputs () Function

FPUTPTR.C

```
#include <stdio.h>
int main() {
    FILE * fp;
    fp = fopen("fputs_test.txt", "w+");
    fputs("DEPARTMENT OF BIOTECHNOLOGY\n",fp);
    fputs("SRM UNIVERSITY", fp);
    fputs("\nSRM NAGAR\n", fp);
    fclose(fp);
    return (0);
  }
```

Output:



```
DEPARTMENT OF BIOTECHNOLOGY
SRM UNIVERSITY
SRM NAGAR
```

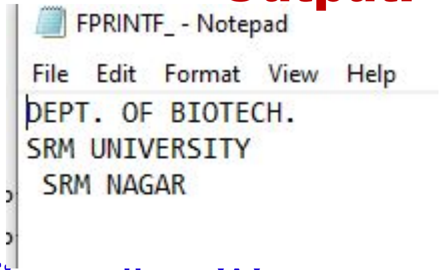
1. In the above program, we have created and opened a file called fputs_test.txt in a write mode.
2. After we do a write operation using fputs() function by writing three different strings
3. Then the file is closed using the fclose function.

fprintf()Function

fprintfptr.c

```
#include <stdio.h>
   int main() {
   FILE *fptr;
 fptr = fopen("fprintf_test.txt", "w"); // "w" defines "writing
mode"
  /* write to file */
  fprintf(fptr, "DEPT. OF BIOTECH.\nSRM UNIVERSITY\n SRM
NAGAR\n");
      fclose(fptr);
      return 0;
   }
```

Output:



```
DEPT. OF BIOTECH.
SRM UNIVERSITY
 SRM NAGAR
```

1. In the above program we have created and opened a file called fprintf_test.txt in a write mode.
2. After a write operation is performed using fprintf() function by writing a string, then the file is closed using the fclose function.

SLO – 1 :
Reading file contents

Reading data from a File

- There are three different functions dedicated to reading data from a file
- **fgetc(file_pointer):** It returns the next character from the file pointed to by the file pointer. When the end of the file has been reached, the EOF is sent back.
- **fgets(buffer, n, file_pointer):** It reads n-1 characters from the file and stores the string in a buffer in which the NULL character '\0' is appended as the last character.

Reading data from a File

fscanf(file_pointer,conversion_specifiers, variable_adresses):

- It is used to parse and analyze data.
- It reads characters from the file and assigns the input to a list of variable pointers variable_adresses using conversion specifiers.
- Keep in mind that as with scanf, fscanf stops reading a string when space or newline is encountered.

Example : fgetc

fgetcfile.c

```
#include <stdio.h>
int main()
{
//file pointer
int ch = 0;
FILE *fp = NULL;
//open the file in read
fp = fopen(“program.txt", "r");
if(fp == NULL)
{
printf("Error in creating the file\n");
exit(1);
}
```

Example : fgetc

```
while( (ch=fgetc(fp)) != EOF )
{
//Display read character
printf("%c", ch);
}
//close the file
fclose(fp);
printf("\n\n\nRead file successfully\n");
return 0;
}
```

Code Analysis

- first, we opened the already created text (“program.txt”) file in reading mode and get the file pointer. Using the if condition verifying that the file has been opened successfully or not.
- After opening the file successfully, using while loop to traverse each and every character of the file (“program.txt”) . The control comes out from the while loop when fgetc get the EOF.

Example :fgets

fgetsfile.c

```
#include <stdio.h>
int main() {
      FILE * file_pointer;
      char str[30],buffer[30];
      int i;

      file_pointer = fopen("program.txt", "r");
      printf("----read a line----\n");
      fgets(buffer, 50, file_pointer);
      printf("%s\n", buffer);
  //  for(i=0;str[i]!=EOF;i++)
   //{
    //       fgets(buffer,3, file_pointer);
   //printf("%s", buffer);
    //       }
      fclose(file_pointer);
      return 0;
  }
```

Example : fscanf

fscanffile.c

```
#include <stdio.h>
int main() {
      FILE * file_pointer;
      char buffer[30], c;
      char str1[10], str2[2], str3[20], str4[2];
      printf("----read and parse data----\n");
      file_pointer = fopen("program.txt", "r"); //reset the pointer

      fscanf(file_pointer, "%s %s %s %s", str1, str2, str3, str4);
      printf("Read String1 |%s|\n", str1);
      printf("Read String2 |%s|\n", str2);
      printf("Read String3 |%s|\n", str3);
      printf("Read String4 |%s|\n", str4);

      printf("----read the entire file----\n");

      file_pointer = fopen("program.txt", "r"); //reset the pointer
      while ((c = getc(file_pointer)) != EOF)
      printf("%c", c);
      fclose(file_pointer);
      return 0;
  }
```

SLO – 2 :

Appending an exiting file

Append mode (a)

- Append mode is used to append or add data to the existing data of file(if any).
- Hence, when you open a file in Append(a) mode, the cursor is positioned at the end of the present data in the file.

Example : append

append.c

```
/*
Program to Read and Write from File using fscanf() and fprintf()
-----------------------------------------------------------*/
#include <stdio.h>
struct emp
{
 char name[21];
 int age;
};
main()
{
 FILE *fp1, *fp2;
 struct emp e;
 fp1 = fopen("program.txt", "a");
 printf("Enter Name and Age:\n");
 scanf("%s %d", e.name, &e.age);
```

Example : append

```
  fprintf(fp1, "%s %d", e.name, e.age);

  fclose(fp1);

  printf("\n\nPrinting contents of file:\n");
  fp2 = fopen("program.txt", "r");
  do
  {
    fscanf(fp2, "%s %d", e.name, &e.age);
    printf("%s %d\n", e.name, e.age);
  } while(!feof(fp2));
  fclose(fp2);
}
```

Difference between Append and Write Mode

- Write (w) mode and Append (a) mode, while opening a file are almost the same.
- Both are used to write in a file.
- In both the modes, new file is created if it does not already exist.
- The only difference they have is, when you open a file in the write mode, the file is reset, resulting in deletion of any data already present in the file.
- While in append mode this will not happen. Append mode is used to append or add data to the existing data of file, if any.
- Hence, when you open a file in Append (a) mode, the cursor is positioned at the end of the present data in the file

SLO – 1 :
File permissions and rights

File permissions and rights

Access to a file has three levels:

- **Read permission –** If authorized, the user can read the contents of the file.
- **Write permission –** If authorized, the user can modify the file.
- **Execute permission –** If authorized, the user can execute the file as a program.

Each file is associated with a set of identifiers that are used to determine who can access the file:

- **User ID (UID)** – Specifies the user that owns the file. By default, this is the creator of the file.
- **Group ID (GID)** – Specifies the user-group that the file belongs to.

Access permissions

Finally, there are three sets of access permissions associated with each file:

- **User permission –** Specifies the level of access given to the user matching the file’s UID.
- **Group permission –** Specifies the level of access given to users in groups matching the file’s GID.
- **Others permission –** Specifies the level of access given to users without a matching UID or GID.

ls -l command

The ls -l command can be used to view the permissions associated with each of the files in the current folder.

Example output of this command is given below.

Example:

```
flags links owner group size modified-date name

total of 24

drwxr-xr-x 7 user staff 224 Jun 21 15:26 .
drwxrwxrwx 8 user staff 576 Jun 21 15:02.
-rw-r--r-- 1 user staff 6 Jun 21 15:04 .hfile
drwxr-xr-x 3 user staff 96 Jun 21 15:17 dir1
drwxr-xr-x 2 user staff 64 Jun 21 15:04 dir2
--w-r--r-- 1 user staff 39 Jun 21 15:37 file1
-rw-r--r-- 1 user staff 35 Jun 21 15:32 file2
```

File mode and the different sets of permissions:

The flags in the first column specify the file mode and the different sets of permissions:

#1) The first character indicates the type of file:

– : represents an ordinary file

d: represents a directory

c: represents a character device file

b: represents a block device file

#2) The next three characters indicate user permissions:

The first of these three indicates whether the user has read permission:

– : indicates that the user does not have read permission.

r: indicates that the user has read permission.

The second character indicates whether the user has to write permission:

– : indicates the user does not have write permission.

w: indicates the user has to write permission.

The last character indicates whether the user has executed permission:

– : indicates that the user does not have to execute permission.

x: indicates that the user has executed permission.

File mode and the different sets of permissions:

#3) The next three characters indicate group permissions, similar to the user permissions above.

#4) The final three characters indicate public permissions, similar to the user permissions above.

- In case the file is an ordinary file, read permission allows the user to open the file and examine its contents. Write permission allows the user to modify the contents of the file. and execute permission allows the user to run the file as a program.
- In case the file is a directory, read permission allows the user to list the contents of the directory. Write permission allows the users to create a new file in the directory, and to remove a file or directory from it. Execute permission allows the user to run a search on the directory.

SLO – 2 :
Changing permissions and rights

Unix command-line tools to change the access permissions

Unix provides a number of command-line tools to change the access permissions:

- Note that only the owner of the file can change the access permissions.

chmod: change file access permissions

description: This command is used to change the file permissions. These permissions are read, write and execute permission for the owner, group, and others.

syntax (symbolic mode):

```
chmod [ugoa][[+-=][mode]] file
```

- The first optional parameter indicates who – this can be (u)ser, (g)roup, (o)thers or (a)ll
- The second optional parameter indicates opcode – this can be for adding (+), removing (-) or assigning (=) permission.
- The third optional parameter indicates the mode – this can be (r)ead, (w)rite, or e(x)ecute.

Changing permissions and rights

Example: Add write permission for user, group and others for file1

```
$ ls -l
-rw-r–r– 1 user staff 39 Jun 21 15:37 file1
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2

$ chmod ugo+w file1

$ ls -l
-rw-rw-rw- 1 user staff 39 Jun 21 15:37 file1
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2

$ chmod o-w file1
$ ls -l
-rw-rw-r– 1 user staff 39 Jun 21 15:37 file1
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2
```