SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

18CSS101J – Programming for Problem

Solving Unit V

SR
INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI

COURSE LEARNING RATIONALE (CLR)	*The purpose of learning this course is to:*
CLR -1:	Think and evolve a logically to construct an algorithm into a flowchart and a pseudocode that can be programmed
CLR -2:	Utilize the logical operators and expressions to solve problems in engineering and real-time
CLR -3:	Store and retrieve data in a single and multidimensional array
CLR -4:	Utilize custom designed functions that can be used to perform tasks and can be repeatedly used in any application
CLR -5:	Create storage constructs using structure and unions. Create and Utilize files to store and retrieve information
CLR -6:	Create a logical mindset to solve various engineering applications

SR

INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI

COURSE LEARNING OUTCOMES (CLO)	At the end of this course, learners will be able to:
CLO -1:	Identify methods to solve a problem through computer programming. List the basic data types and variables in C
CLO -2:	Apply the logic operators and expressions. Use loop constructs and recursion. Use array to store and retrieve data
CLO -3:	Analyze programs that need storage and form single and multi-dimensional arrays. Use preprocessor constructs in C
CLO -4:	Create user defined functions for mathematical and other logical operations. Use pointer to address memory and data
CLO	Create structures and unions to represent data constructs. Use files to

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

S. No	LEARNING RESOURCES
	TEXT BOOKS
1.	*Zed A Shaw, Learn C the HardWay: Practical Exercises on the Computational SubjectsYou Keep Avoiding (Like C), AddisonWesley, 2015*
2.	*W. Kernighan, Dennis M. Ritchie,The C Programming Language, 2nd ed. Prentice Hall, 1996*
3.	*Bharat Kinariwala,Tep Dobry, Programming in C, eBook*
4.	*http://www.c4learn.com/learn-c-programming-language/*

SR INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

UNIT V

INTRODUCTION

Initializing Structure, Declaring Structure variable- Structure using typedef, Accessing members – Nested structure Accessing elements in a structure array – Array of structure Accessing elements in a structure array –Passing Array of Structure to function- Array of Pointers to structures- Bit Manipulation of structure and pointer to structure – Union Basic and declaration – Accessing Union Members Pointers to union

SR INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

UNIT V

INTRODUCTION

Dynamic memory allocation, malloc, realloc , free – Allocating Dynamic Array- Multidimensional array using dynamic memory allocation- file: opening, defining, closing, File Modes, File Types- Writing contents into a file – Reading file contents – Appending an existing file- File permissions and rights- changing permissions and rights.

STRUCTURE

❑ A Structure is a collection of variables of different data types under a single name.

❑ The   variables are   called members   of the   structure.

❑ The structure is also called a user-defined data type.

Defining a Structure

- Syntax:

struct structure_name

    {
    data_type member_variable1;
    data_type member_variable2;
    …………………………………; data_type member_variableN;
    };

Once structure_name is declared asnew data type, then   variables of that typecan be declared as:

    struct structure_name structure_variable;

Note: The members of a structure do not occupy memory until they are associated with a variable.

9

- Example

  struct student

  {
  char
  name[20];
  int roll_no;
  float marks;
  char gender;
  long int phone_no;
  };
  struct student st;

- Multiple variables of *struct student* type can be declared as:

  struct student st1, st2, st3;

Defining a structure…

- Each variable of structure has its own copy of member variables.

- The member variables are accessed using the dot (.) operator or memberoperator.

- For example: *st1.name* is member variable *name* of *st1* structure variable while *st3.gender* is member variable *gender* of *st3* structure variable.
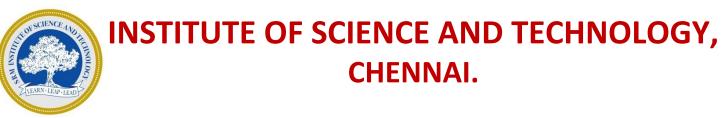
Defining a structure…

struct student

{

charname[20];

int roll_no;

float

marks;

char

gender;

long

intphone_no;

}st1, st2, st3;

struct

{

charname[20]; int

roll_no; float

marks;

char gender;

long intphone_no;

}st1, st2, st3;

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.

```
struct
student
    {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);
```

Initialization

Partial Initialization

- We can initialize the first few members and leave the remaining blank.

- However, the uninitialized members should be only at the end of the list.

- The uninitialized members are assigned default values as follows:
  - Zero for integer and floating point numbers.
  - '\0' for characters and strings.

```c
struct
student
    {char
    name[20];
    int roll;
    char remarks;
    float marks;
    };
void main()
{
struct student s1={"name", 4};
clrscr();
printf("Name=%s", s1.name);
printf("\n Roll=%d", s1.roll);
printf("\n Remarks=%c",
s1.remarks);  printf("\n
Marks=%f", s1.marks);  getch();
}
```

Accessing member of structure/ Processing a structure

- By using dot (.) operator or period operator or operator.

- Syntax:

  structure_variable.member

- Here, *structure_variable* refers to the name of a *struct* type variable and *member* refers to the name of a member within thestructure.

membe r

Question

- Create a structure named *student* that has *name*, *roll and mark* as members. Assume appropriate types and size of member. Write a program using structure to read and display the data entered by the user.

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

void main()
{
struct student s;
clrscr();
printf("Enter
name:\t");
gets(s.name);
printf("\n Enter roll:\t");
scanf("%d", &s.roll);
printf("\n Enter
marks:\t");  scanf("%f",
&s.mark);
printf("\n Name \t Roll \t Mark\n");
printf("\n.............................\n");
printf("\n%s\t%d\t%f", s.name, s.roll,
s.mark);
```

Copying and Comparing Structure Variables

- Two variables of the same structure type can be copied in   the same way as ordinary variables.

- If *student1* and *student2* belong to the same structure, then the following statements are valid:

  student1=student2; student2=student1;

- However, the statements such as:

  student1==student2 student1!=student2

are not permitted.

- If we need to compare the structure variables, we may do   so by comparing members individually.

SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI.

Here, structure has beendeclared global i.e. outside of main() function. Now, any function can access it and create a structure variable.

```c
structstudent
    {
    char name[20];
    int roll;
    };
void main()
{
struct student
student1={"ABC", 4, };
struct student
student2; clrscr();
student2=student1;

printf("\nStudent2.name=%s",
student2.name);
printf("\nStudent2.roll=%d",
student2.roll);
if(strcmp(student1.name,student2.na me)==0 &&
    (student1.roll==student2.roll))
    {
    printf("\n\n student1 and
    student2 are same.");
    }
getch();
}
```

How structure elements are stored?

- The elements of a structure are always contiguous stored in memory locations.

- A structure variable reserves number of bytes equal to sum of bytes needed to each of its members.

- Computer stores structures using the concept of "word boundary". In a computer with two bytes word boundary, the structure variables are stored left aligned and consecutively one after the other (with at most one byte unoccupied in between them cal ed slack byte).

How structure elements are stored?

- When we declare structure variables, each one of them may contain slack bytes and the values stored in such slack bytes are undefined.

- Due to this, even if the members of two variables are equal, their structures do not necessarily compare.

Array of structure

- Let us consider we have a structure as:

struct student
{
char name[20];
int roll;
char remarks;
float marks;
};

- If we want to keep record of 100 students, we have to make 100 structure variables like st1, st2, …,st100.

- In this situation we can use array of structure to store the records of 100 students which is easier and efficient to handle (because loops can be used).

24

Array of structure…

- Two ways to declare an array of structure:

struct student

{

char name[20];

int roll;
char remarks;
float marks;

}st[100];

struct student

{
char name[20];
int roll;
char remarks;
float marks;
};
struct student st[100];

- Write a program that takes roll_no, fname   lname of 5 students and prints the same records in ascending order on the basis of roll_no

Reading values

```c
for(i=0; i<5; i++)
    {
        printf("\n Enter roll number:");
        scanf("%d",  &s[i].roll_no);


        printf("\n Enter first name:");
        scanf("%s",  &s[i].f_name);


        printf("\n Enter Lastname:");
        scanf("%s",  &s[i].l_name);
    }
```

Sorting values

```
for(i=0; i<5; i++)
    {
        for(j=i+1; j<5; j++)
        {
                if(s[i].roll_no<s[j].roll_no)
                {
                    temp = s[i].roll_no;
                    s[i].roll_no=s[j].roll_n
                    o;  s[j].roll_no=temp;
                }
        }
    }
```

Question

- Define a structure of employee having data members name, address, age and salary. Take the data for n employees in an array and find the average salary.

- Write a program to read the *name*, *address*, and *salary* of 5 employees using array of structure. Display information of each employee in alphabetical order of theirname.

Array within Structure

- We can use single or multi dimensional arrays of type *int* or *float*.
  - E.g *struct*
  . *student*

       {

       char
       name[20];

       int roll;

       float
       marks[6];

  struct student
  s[10];

- Here, the member *marks* contains six elements, *marks[0], marks[1],* ...., *marks[5]* indicating marks obtained in six different subjects.

- These elements can be accessed using appropriate subscripts.

- For example, *s[25].marks[3]* refers to the marks obtained in the fourth subject by the 26th student.

Array within structure…

31

Reading Values

```c
for(i=0;i<n;i++)
{   printf("\n Enter information about student%d:", i+1);
        printf("\n  Name:\t");
        scanf(" %s",  s[i].name);
        printf("\nClass:\t");
        scanf("%d", &s[i]._class);
        printf("\n Section:");
        scanf(" %c",  &s[i].section);
        printf("\n Input marks of 6 subjects:\t");
        for(j=0;j<6;j++)
        {   scanf("%f", &temp);
                s[i].marks[j]=temp;
}       }
```

Structure within anotherStructure (Nested Structure)

- Let us consider a structure *personal_record* to store the information of a personas:

struct   personal_record
             { char name[20];
                 int day_of_birth;
                 int month_of_birth;
                 int year_of_birth;
                 float salary;
             } person;

Structure within another Structure (Nested Structure)...

- In the structure above, we can group all the items related to birthday   together and declare
  them under a substructure as:

```
struct Date
        {
        int day_of_birth;
        int month_of_birth;
        int  year_of_birth;
        };
struct  personal_record
        {
        char name[20];   struct Date birthday;
        float salary;
```

Structure within anotherStructure (Nested Structure)...

- Here, the structure *personal_record* contains a member named *birthday* which itself is a structure with 3 members. This is called structure within structure.

- The members contained within the inner structure canbe accessed as:

  person.birthday.day_of_birth
  person.birthday.month_of_birth
  person.birthday. year_of_birth

- The other members within the structure   are accessed
  personal_record usual: as

  person.name   person.salary

3
5

```c
printf("Enter name:\t");
scanf("%s", person.name);
printf("\nEnter day of birth:\t");
scanf("%d", &person.birthday.day_of_birth);
printf("\nEnter month of birthday:\t");
scanf("%d", &person.birthday.month_of_birth);
printf("\nEnter year of birthday:\t");
scanf("%d", &person.birthday.year_of_birth);
printf("\nEnter salary:\t");
```

Structure within another Structure (Nested Structure)…

- ***Note:- More than one type of structures can be nested…***

```c
struct
 date
    {int day;
    int
    month;
    int year;
    };

struct name
    {
    char
    first_name[10];
    char
    middle_name[10];
    char last_name[10];
    };

struct personal_record
    {
    float salary;
    struct date
    birthday,deathday;
    struct name full_name;
    };
```

Assignment

- Create a structure named *date* that has *day*, *month* and *year* as its members. Include this structure as a member in another structure named *employee* which has *name*, *id* and *salary* as other members. Use this structure to read and display employee's name, id, date of birthday andsalary.

Pointer to Structure

- Astructure type pointer variable can be declared as:

    struct book

    {
    char name[20];
    int pages; float price;
    };
    *struct book *bptr;*

- However, this declaration for a pointer to structure does not allocate any memory for a structure but allocates only for a pointer, so that to access structure's members through pointer ***bptr,*** we must allocate the memory using ***malloc()*** or tion. OR

- Now, individual structure members are accessed as:

    bptr->name bptr->pages bptr->price

    *(*bptr).name (*bptr).pages   (*bptr).price*

- Here, **->** is called arrow operator and there must be a pointer to the structure on the left side of

    thisoperator.

```c
struct book
*bptr;

bptr=(struct book *)malloc(sizeof(struct book));

printf("\n Enter name:\t");
scanf("%s", bptr->name);
printf("\n Enter no. of pages:\t");
scanf("%d", &bptr->pages);
printf("\n Enter price:\t");
scanf("%f", &
```

Pointer to Structure…

- Also, the address of a structure type variable can be stored in a structure type pointer variableas follows:

struct book

    {

    char name[20]; int

    pages; float price;

    };

*struct book b,*bptr; bptr=&b;*

- Here, the base address of *b* is assigned to*bptr* pointer.

Pointer to Structure…

- Now the members of the structure book can be accessed in 3 ways as:

b.name	bptr->name	(*bptr).name
b.pages	bptr->pages	(*bptr).pages
b. price	bptr-> price	(*bptr).price

Pointer to array of structure

- Let we have a structure asfollows:

struct book
    {
char name[20]; int
pages; float price;
};
*struct book b[10], *bptr;*

- Then the assignment statement *bptr=b;* assigns the address of the zeroth element of *b* to *bptr*.

Pointer to array of structure…

- The members of *b[0]* can be accessedas:

bptr->name   bptr->pages   bptr->price

- Similarly members of *b[1]* can be accessedas:

(bptr+1)->name (bptr+1)->pages (bptr+1)->price

- The following *for* statement can be used to print all the values of array of structure *b* as:

printf("%s %d %f", bptr->name, bptr->pages,

for(bptr=b;bptr<b+10;bptr++)

4
5

Problem

- Define a structure of employee having data members name, address, age and salary. Take data for n employee in an array **dynamically** and find the averagesalary.

- Define a structure of student having data members name, address, marks in C language, and marks in information system. Take data for n students in an array dynamically and find the total marksobtained.

Function and Structure

- We will consider four caseshere:

  - *Passing the individual members to functions*

  - *Passing whole structure to functions*

  - *Passing structure pointer to functions*

  - *Passing array of structure to functions*

Passing structure member to functions

- Structure members can be passed to functions as actual arguments in function call like ordinary variables.

- Problem: Huge number of structure members

- Example: Let us consider a structure *employee* having members *name*, *id* and *salary* and pass these members to afunction:

display(emp.name,emp.id,emp.salary);

```
Void display(char e[],int id ,float sal)
{
printf("\ nName\ t\ tID\ t\ tSalary\ n);
printf("%s\t%d\t%.2f",e,id,sal);
}
```

Passing whole structure to functions

- Whole structure can be passed to afunction by the syntax:

- ***function_name(structure_variable_name);***

- Thecalled function has the form:

- *return_type function_name(struct tag_name structure_variable_name)*
  - *{*
  - *... ... ...;*

4
5

display(emp);

```c
void display(struct employee e)
{
printf("\nName\tID\tSalary\n");
printf("%s\t%d\t%.2f",e.name,e.id,e.sa
lar);
}
```

Passing structure pointer tofunctions

- In this case, address of structure variable is passed as an actual argument to a function.

- The corresponding formal argument must be a   structure type pointer variable.

- Note: Any changes made to the members in   the called function are directly reflected in the calling function.

display(&emp);

```c
void display(struct employee *e)
{
printf("\nName\tID\tSalary\n");
printf("%s\t%d\t%.2f",e->name,e->id,e->salary);
}
```

Passing array of structures to function

- Passing an array of structure type to a function is similar to passing an array of any type to afunction.

- That is, the name of the array of structure is passed by the calling function which is the base address of the array of structure.

- Note: The function prototype comes after the structure definition.

display(emp);   //emp is array name of size 2

```c
void display(struct employee ee[])
{
int i;
printf("\n Name\t\t ID\t\t
Salary\n");  for(i=0;i<2;i++)
   {
   printf("%s\t\t%d\t\t%.2f\n",ee[i].name,ee[i].id,ee[i].salary);
   }
}
```

Structure using typedef

- It allows us to introduce synonyms for data types which could have been declared some other way.

- It is used to give New name to the Structure.

- New name is used for Creating instances, Passing values to function,declaration etc…

Example:

```c
#include<stdio.h>
 int main()
 {
typedef int Number;
 Number num1 = 40,num2 = 20;
 Number answer;

answer = num1 + num2;

printf("Answer : %d",answer);
return(0);
}
```

Output :

Answer : 60

- In the above program we have used typedef to create alias name to data type. We have created alias name to 'int' data type. We have given new name to integer data type i.e 'Number'.

- In the second example, **Record** is **tag-name**. **'employee'** is nothing but **New Data Type**. We can now create the variables of type **'employee'** Tag name is optional.

Different Ways of Declaring Structure using Typedef :

```
typedef struct
{
char ename[30];
int ssn;
int deptno;
}
employee;
```

```
typedef struct
Record
{
char ename[30];
int ssn;
int deptno;
}
employee;
```

Live Example : Using Typedef For Declaring Structure

```c
#include<stdio.h>
 typedef struct b1
{
 char bname[30];
 int ssn;
 int pages;
}
book;
book b1 = {"Let Us C",1000,90};
 int main()
{
printf("\nName of Book :s",b1.bname);
printf("\nSSN of Book : %d",b1.ssn);
printf("\nPages in Book :
%d",b1.pages);  return(0);
}
```

OUTPUT
:

Name of Book : Let Us C

 SSN of Book : 1000

 Pages in Book : 90

Pointer to Structure Array

- Like we have array of integers, array of pointers etc, we can also have array of structure variables. And to use the array of structure variables efficiently,

- we use **pointers of structure type**. We can also have pointer to a single structure variable, but it is mostly used when we are dealing with array of structure variables.

Accessing Structure Members with Pointer

```c
#include <stdio.h>
struct Book
{ char name[10];
 int price; }
 int main()
{
 struct Book a; //Single structure variable
 struct Book *ptr; //Pointer of Structure type
 ptr = &a;
struct Book b[10]; //Array of structure variables
 struct Book *p; //Pointer of Structure type
 p = b;
 return 0;
 }
```

Accessing Structure Members with Pointer

- To access members of structure using the structure variable, we used the dot . operator.

- But when we have a pointer of structure type, we use arrow -> to access structure members.

Accessing Structure Members with Pointer

```c
#include <stdio.h>
 struct my_structure
 {  char name[20];
 int umber;
int rank;

};
 int main()
{
 struct my_structure variable = {"StudyTonight", 35, 1}; struct my_structure *ptr; ptr =
&variable;  printf("NAME: %s\n", ptr->name);

printf("NUMBER: %d\n", ptr->number);
printf("RANK: %d", ptr->rank);
return 0;
}
```

Bit Manipulation

- Suppose we want to store the gender of the person , then instead of wasting the complete byte we can manipulate single bit. We can consider Single bit as a flag. Gender of Person can be stored as **[M/F]** . By Setting the **flag bit**

- **1** we can set Gender as **"Male"** and "**Female**" by setting bit as 0

- 1.	To pack **Several data objects** in the single memory word , Bits are used.

- 2. Flags can be used in order to store the Boolean values **(T / F)**.

- 3. A method to define a structure of packed information is known as **bit fields**.

Syntax : Bit Manipulation

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

Explanation :

- In the above example, we can say that we have **allocated specific number of bits to each structure member**.

- Integer can store 2 bytes (*) , so 2 bytes are **manipulated in bits** and 1 bit is reserved for b1. Similarly b2,b3 will get single bit.

- Similarly we can structure efficiently to **store boolean values or smaller values** that requires little membory.

How to access the Individual Bits ?

- data1.b1
- data1.b2
- data1.b3
- data1.b4
- data1.b5

How to initialize Structure ?

struct databits
{
- -

- - -

}data1 = { 1,1,0,10,234 };
Initialized Result –

data1.b1 = 1

data1.b2 = 1

data1.b3 = 0

data1.b4 = 10

 data1.b5 = 234

Unions

- Unions are quite similar to the structures in C.

- Union is also a derived type as structure.

- Union can be defined in same manner as structures just the keyword used in defining union in union where keyword used in defining structure was struct.

```c
#include <stdio.h>
#include <string.h>
union Data
{
int i;
float f;
char str[20];
};
int main( )
{
union Data data;
printf( "Memory size occupied by data : %d\n", sizeof(data));
return 0;
}
```

OUTPUT

Memory size occupied by data : 20

Instead of union if we put struct keyword the output will be
Memory size occupied by data : 26

The memory occupied by a union will be large enough to hold the largest member of the union

Accessing Union Members

```c
#include <stdio.h>
 #include <string.h>
union Data
{
int i;
float f;
char str[20];
};
int main( )
{
    union Data data;
    data.i = 10;
    data.f = 220.5;
    strcpy( data.str, "C Programming");
    printf( "data.i : %d\n", data.i);
    printf( "data.f : %f\n", data.f);
    printf( "data.str : %s\n", data.str);
return 0;
}
```

OUTPUT
data.i : 1917853763
data.f : 4122360580327794860452759994368.000000
data.str : C Programming

The values of **i** and **f** members of union got corrupted because the final value assigned to the variable has occupied the memory location and this is the reason that the value of **str** member is getting printed very well.

Pointer to Union

```c
#include<stdio.h>
union team
{
char *name;
int members;
char captain[20]; };
int main()
{
union team t1,*sptr = &t1;
t1.name = "India";
printf("\nTeam : %s",(*sptr).name);
printf("\nTeam : %s",sptr->name);
return 0;
}
```

Output :
Team = India
Team = India

Dynamic Arrays

- Dynamic arrays are created using pointer variables and memory management functions malloc,calloc and realloc.

- The concept of dynamic arrays is used in creating and manipulating data structures such as linked list, stacks and queues.

Limits of Static Allocation

What if we don't know how much space we will need ahead of time?
Example:
✔ask user how many numbers to read in
✔read set of numbers in to array (of appropriate size)
✔calculate the average (look at all numbers)
✔calculate the variance (based on the average)
Problem: how big do we make the array??
    using static allocation, have to make the array as big as the user might specify
    (might not be big enough)

Dynamic Memory Allocation

1. Allow the program to allocate some variables (notably arrays), during the program, based on variables in program (dynamically)

2. Previous example: ask the user how many numbers to read, then allocate array of appropriate size

3. Idea: user has routines to request some amount of memory, the user then uses this memory, and returns it when they are done
memory allocated in the *Data Heap*

Dynamic Memory Allocation (cont.)

Memory Management Functions

calloc - routine used to allocate arrays of memory

malloc - routine used to allocate a single block of memory

realloc - routine used to extend the amount of space allocated previously

free - routine used to tell program a piece of memory no longer needed

    <u>note</u>:

✔memory allocated dynamically does not go away at the end of functions, you MUST explicitly free it up

malloc Function

It is a process by which the required memory address is obtained without an explicit declaration.

The required memory space is obtained by using the memory allocation functions like malloc() and calloc().

malloc() function: Used to allocate a single block of memory to store values of specific data types.

malloc Function (cont.)

Syntax:

ptr=(type *)malloc(size);

ptr : Pointer variable

type : Data type

size : Number of bytes to be alloted

Ex: int *ptr;

ptr= (int *)malloc(20);

The alloted space can be used to store 10 int type variables

Example program for malloc() function:
```
#include<stdio.h>
#include<malloc.h>
#include<conio.h>
void main()
{
    float *fp;
    fp=(float *)malloc(10);
    printf("Enter a float value : ");
    scanf("%f", &fp);
    printf("The address of pointer in memory is : %u", fp);
    printf("The value stored in memory is : %f", *fp);
    getch();
}
```

calloc Function

It is used to allocate memory in multiple blocks of same size during program execution.

Syntax:

ptr = (type *)calloc(n,m);

ptr = Pointer variable

type = Data type

n = Number of blocks to be allotted

m = Number of bytes in each block of memory

Ex:

float *ptr;

ptr=(float *)calloc(20,4)

Example program 1 for calloc() function:
#include<stdio.h>
#include<calloc.h>
#include<conio.h>
void main()
{
    float *fp;
    fp=(float *)calloc(10,4);
    printf("Enter a float value : ");
    scanf("%f", &fp);
    printf("The address of pointer in memory is : %u", fp);
    printf("The value stored in memory is : %f", *fp);
    getch();
}

Example program 2 for calloc() function:

```
float *nums;
int N;
int I;

printf("Read how many numbers:");
scanf("%d",&N);
nums = (float *) calloc(N, sizeof(float));
/* nums is now an array of floats of size N */
for (I = 0; I < N; I++) {
  printf("Please enter number %d: ",I+1);
  scanf("%f",&(nums[I]));
}
/* Calculate average, etc. */
```

Releasing Memory (free)

prototype: void free(void *ptr)

      memory at location pointed to by ptr is released (so we could use it again in the future)
      program keeps track of each piece of memory allocated by where that memory starts
      if we free a piece of memory allocated with calloc, the entire array is freed (released)
      results are problematic if we pass as address to free an address of something that was
      not allocated dynamically (or has already been freed)

free Example

```
float *nums;
int N;
printf("Read how many numbers:");
scanf("%d",&N);
nums = (float *) calloc(N, sizeof(float));
/* use array nums */
/* when done with nums: */
free(nums);
/* would be an error to say it again - free(nums) */
```

The Importance of free

```
void problem() {
   float *nums;
   int N = 5;
   nums = (float *) calloc(N, sizeof(float));
   /* But no call to free with nums */
} /* problem ends */
```

When function problem called, space for array of size N allocated, when function ends, variable nums goes away, but the space nums points at (the array of size N) does not (allocated on the heap) - furthermore, we have no way to figure out where it is)
Problem called *memory leakage*

realloc() function: It is used to modify or reallocate the memory space which is previously allocated.

Syntax: ptr=realloc(ptr,size);

Ex: int *p;

  p=(int *)malloc(50);

   ----

  p=realloc(p,100); //50 bytes is modified as 100 bytes.

free() function: It is used to release the memory space which is allocated using malloc() or calloc() function

Syntax: free(ptr);

realloc Example

```
float *nums;
int I;
nums = (float *) calloc(5, sizeof(float));
/* nums is an array of 5 floating point values */
for (I = 0; I < 5; I++)
  nums[I] = 2.0 * I;
/* nums[0]=0.0, nums[1]=2.0, nums[2]=4.0, etc. */
nums = (float *) realloc(nums,10 * sizeof(float));
/* An array of 10 floating point values is allocated, the first 5
floats from the old nums are copied as the first 5 floats of the
new nums, then the old nums is released */
```

File Handling in C

What is a File?

- A *file* is a collection of related data that a computers treats as a single unit.

- Computers store files to secondary storage so that the contents of files remain intact when a computer shuts down.

- When a computer reads a file, it copies the file from the storage device to memory; when it writes to a file, it transfers data from memory to the storage device.

- C uses a structure called **FILE** (defined in **stdio.h**) to store the attributes of a file.

Steps in Processing a File

1. Create the stream via a pointer variable using the **FILE** structure:

   FILE *p;

2. Open the file, associating the stream name with the file name.

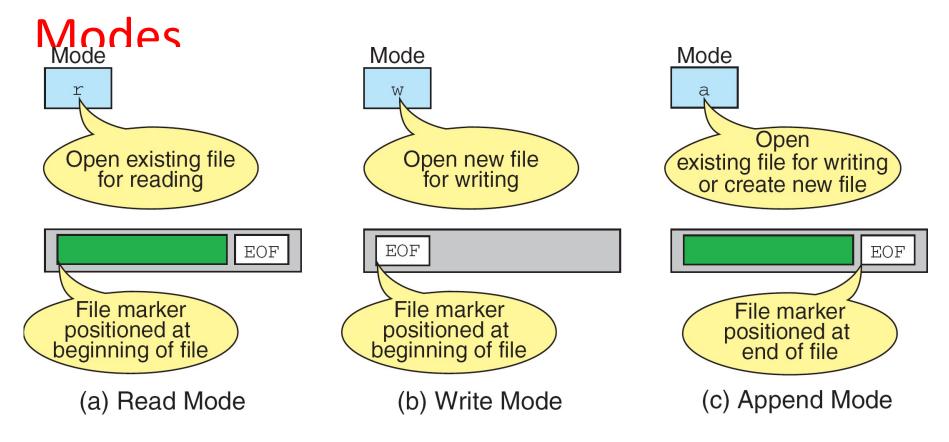3. Read or write the data.

4. Close the file.

The basic file operations are

- fopen - open a file- specify how its opened (read/write) and type (binary/text)
- fclose - close an opened file
- fread - read from a file
- fwrite - write to a file
- fseek/fsetpos - move a file pointer to somewhere in a file.
- ftell/fgetpos - tell you where the file pointer is located.

File Open

Mode	Meaning
r	Open text file in read mode • If file exists, the marker is positioned at beginning. • If file doesn't exist, error returned.
w	Open text file in write mode • If file exists, it is erased. • If file doesn't exist, it is created.
a	Open text file in append mode • If file exists, the marker is positioned at end. • If file doesn't exist, it is created.

More on File Open Modes

(a) Read Mode

(b) Write Mode

(c) Append Mode

Additionally,

- r+ - open for reading and writing, start at beginning

- w+ - open for reading and writing (overwrite file)

- a+ - open for reading and writing (append if file exists)

File Open

- The file open function (**fopen**) serves two purposes:
  - It makes the connection between the physical file and the stream.
  - It creates "a program file structure to store the information" C needs to
    process the file.

- Syntax:

  filepointer=fopen("filename", "mode");

More On **fopen**

- The file mode tells C how the program will use the file.

- The filename indicates the system name and location for the file.

- We assign the return value of **fopen** to our pointer variable:

  spData = fopen("MYFILE.TXT", "w");
  spData = fopen("A:\\MYFILE.TXT", "w");

More On
fopen

Closing a File

- When we finish with a mode, we need to close the file before ending the program or beginning another mode with that same file.

- To close a file, we use **fclose** and the pointer variable:
  fclose(spData);

fprintf()

Syntax:

fprintf (fp,"string",variables);

Example:

int i = 12;

float x = 2.356;

char ch = 's';

FILE *fp;

fp=fopen("out.txt","w");

fprintf (fp, "%d %f %c", i, x, ch);

fscanf()

Syntax:

fscanf

(fp,"string",identifiers); Example:

FILE *fp;

Fp=fopen("input.txt","r")

; int i;

fscanf (fp,"%d",i);

getc()

Syntax:

Syntax:

identifier = getc (file

pointer); Example:

FILE *fp;

fp=fopen("input.txt","r")

; char ch;

ch = getc (fp);

putc()

write a single character to the output file, pointed to by fp.

Example

:   FILE

*fp;

char ch;

putc (ch,fp);

End of File

- There are a number of ways to test for the end-of-file condition.   Another way   is to use the value returned by the *fscanf* function:

    ```
    FILE *fptr1;

    int istatus ;

    istatus = fscanf (fptr1, "%d", &var) ;

    if ( istatus == feof(fptr1) )

    {

            printf ("End-of-file encountered.\n") ;
    ```

Reading and Writing Files

```c
#include <stdio.h>
int main ( )
{
    FILE *outfile, *infile ;
    int b = 5, f ;
    float a = 13.72, c = 6.68, e, g ;
    outfile = fopen ("testdata", "w") ;
    fprintf (outfile, " %f %d %f ", a, b, c) ;

    fclose (outfile) ;
    infile = fopen ("testdata", "r") ;
    fscanf (infile,"%f %d %f", &e, &f, &g) ;
    printf (" %f %d %f \n ", a, b, c) ;
    printf (" %f %d %f \n ", e, f, g) ;
```

SRM
INSTITUTE OF SCIENCE AND TECHNOLOGY,
CHENNAI.

Example

```
#include <stdio.h>
#include<conio.h>
void main()
{
    char ch;
    FILE *fp;
    fp=fopen("out.txt","r");
    while(!feof(fp))
        {
        ch=getc(fp);
        printf("\n%c",ch);
        }
    getch();
}
```

fread ()

Declaration:

size_t fread(void *ptr, size_t size, size_t n, FILE *stream);

Remarks:

fread reads a specified number of equal-sized
data items from an input stream into a block.

ptr   = Points to a block into which data is
read size = Length of each item read, in bytes
n = Number of items read
stream = file pointer

Example

Example:

```c
#include <stdio.h>
int main()
{
    FILE *f;
    char buffer[11];
    if (f = fopen("fred.txt", "r"))
    {
    fread(buffer, 1, 10, f);
    buffer[10] = 0;
    fclose(f);
    printf("first 10 characters of the file:\n%s\n",
    buffer);
    }
 return 0;
}
```

fwrite()

Declaration:

size_t fwrite(const void *ptr, size_t size, size_t n, FILE*stream);

Remarks:

fwrite appends a specified number of equal-sized data items to an output file.

ptr   = Pointer to any object; the data written begins at

ptr   size   = Length of each item of data

n   =Number of data items to be appended

stream = file pointer

Example

Example:

```c
#include <stdio.h>
int main()
{
    char a[10]={'1','2','3','4','5','6','7','8','9','a'};
    FILE *fs;
    fs=fopen("Project.txt","w")
    ;  fwrite(a,1,10,fs);
    fclose(fs);
    return 0;
}
```

fseek()

This function sets the file position indicator for the stream pointed to by stream or you can say it seeks a specified place within a file and modify it.

SEEK_SET	**Seeks from beginning of file**
SEEK_CUR	**Seeks from current position**
SEEK_END	**Seeks from end of file**

Example:

```c
#include <stdio.h>

int main()
{
    FILE * f;
    f = fopen("myfile.txt",
    "w");  fputs("Hello World"   SEEK_CUR,   SEEK_END
    fseek(f, 0,
    SEEK_SET);  fputs("
    India", f);  fclose(f);
    return 0;
}
```

ftell()

offset = ftell(file pointer);

"ftell" returns the current position for input or output on the
file #include <stdio.h>

```c
int main(void)
{
  FILE *stream;
  stream = fopen("MYFILE.TXT",
  "w");  fprintf(stream, "This is a
  test");
  printf("The file pointer is at byte %ld\n", ftell(stream));
  fclose(stream);
  return 0;
```

SRM
INSTITUTE OF SCIENCE AND TECHNOLOGY,
CHENNAI.

THANK YOU………