CHAPTER - 9

"Structures"

A collection of values of different data types.

Example :-
For a student store the following :
- name (string)
- rollno (integer)
- cgpa (Float)

• - dot operator

syntax :-
```
struct student {
    char name[100];
    int roll;
    float cgpa;
};
```

Properties

Datatype declare/define (once) → before main()

In-built datatype	user-defined
int	structure
float	
Char	

```
int n;
struct student s1;
s1.cgpa = 7.5;
s1.name = "CHETAN";   //error (cannot be reinitialized)
s1.roll = 045;
```

because char str[100] can't be reinitialized

```
strcpy(s1.name, "CHETAN");
```

works if `char *str;`

• → dot operator

Basic structure Program

```
#include<stdio.h> #include<string.h>
struct student {
    int roll;
    float cgpa;
    char name[100];
};
void main() {
    struct student s1;
    s1.cgpa = 9.5;
    s1.roll = 45;
    strcpy (s1.name, "chetan");
    printf ("student name is %s\n", s1.name);
    printf ("student roll no is: %d\n", s1.roll);
    printf ("student cgpa is %f\n", s1.cgpa);
}
```

structure declaration

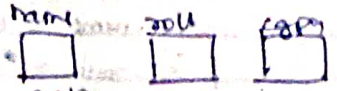
variable (structure)

initialization

printing

Structures in Memory

```
struct Student {
    char name[100];
    int roll;
    float cgpa;
};
```



Structures are stored in continuous memory location

Practice Qs 55.

write a pgm to store the data of 3 students.

```
#include<stdio.h>
#include<string.h>
struct student {
    int roll;
    float cgpa;
    char name[100];
};
int main(){
    struct student s1;
    s1.roll = 1;
    s1.cgpa = 9.0;
    strcpy (s1.name, "chetan");
    printf ("Student name: %s\n", s1.name);
    printf ("student roll no: %d\n", s1.roll);
    printf ("student cgpa: %f\n", s1.cgpa);

    s2 = s1;
    s3 = s1;
}
```

copy

paste → s2 instead of s1

paste → s3 instead of s1

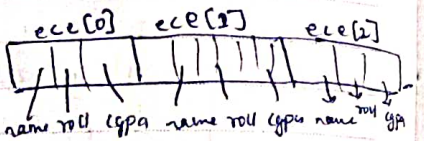
benefits of array :-
* Same elements. Storage with single variable name,
* sorting, searching concepts.

Ex:- to store 100 elements we don't required 100 variables.

benefits of structure :-
* Heterogeneous collection of data.
* code readability Enhanced.

Array of Structures

datatype variable count
struct Student ECE[100];
struct Student CSE[100];



CSE[0].roll = 200
CSE[0].cgpa = 7.6
} Access.

Ex:-
```
#headerfile
#structure declaration
void main(){
    int n=3;
    struct student s[n];
```

(contd)


```
    for(int i=0;i<n;i++){
        printf("Enter details of %dth student\n", i+1);
        printf("Enter roll :");
        scanf("%d", &s[i].roll);
        printf("Enter cgpa:");
        scanf("%f", &s[i].cgpa);
        printf("Enter name:");
        scanf("%s", &s[i].name);
    }
    for(int i=0;i<n;i++){
        printf("\n\n student %d details:\n", i+1);
        printf("student roll : %d\n", s[i].roll);
        printf("student cgpa : %.2f\n", s[i].cgpa);
        printf("student name : %s\n", s[i].name);
    }
}
```

Initializing Structures

format has to follow like structure declaration

struct student s1 = { 1, 9.9, "chetan" };
roll cgpa name

Pointers to structures

```
struct student s1;
struct student *ptr;        same as int
ptr = &s1;
```

Ex:-

```
void main(){
    struct student s1 = {1, 9.9, "Chetan"};
    struct student *p = &s1;
    printf("%d\n", (*p).roll);
}
```

access value using pointer

Arrow Operator

(*ptr).code ⟷ ptr → code

Ex:-

```
void main(){
    struct student s1={1, 9.9, "chetan"};
    struct student *p = &s1;
    printf("%d\n", p→roll);
    printf("%f\n", p→cgpa);
    printf("%s\n", p→name);
}
```


passing structure to function

```
//Function prototype
void printInfo (struct student s1);
```

Ex:- { structure dec^n & header files.

```
void printInfo (struct student s1){
    printf("Student Details\n");
    printf("roll:%d\n", s1.roll);
    printf("cgpa:%f\n", s1.cgpa);
    printf("name:%s\n", s1.name);
}
void main(){
    struct student s1 = {1, 9.9, "Chetan"};
    printInfo(s1);
}
```

NOTE (*) Structures are pass by value

typedef KeyWord (used to create nicknames)

Mostly Used in Structures

used to create alias for data types

```
typedef struct ComputerEngineeringStudent {
        int roll;
        float cgpa;
        char name[100];
} coe;
```

struct computerEngineeringStudent s1; (X)

coe s1; (✓)

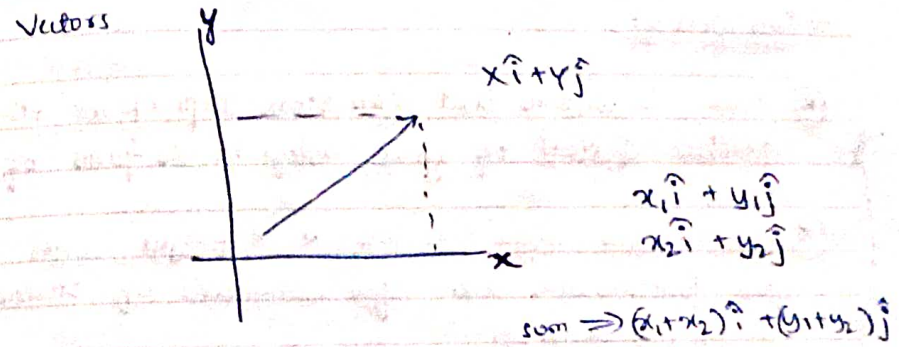
Ex:-
```
#include<stdio.h>
typedef struct student {
        int roll;
        float cgpa;
        char name[100];
} s;
// alias name
void main(){
        s s1 = {1, 9.9, "Chetan"};
        printf("%d\n%f\n%s\n", s1.roll, s1.cgpa, s1.name);
}
```

NOTE:// typedef can be used for int, float, char also

Qs:- Take address (house no, block, city, state) of 5 people

Sol:- File.No:-97 addressStructure.c

Qs. create a structure to store vectors. Then make a function to return sum of 2 vectors.



Sol:- File.No-98 vectorStructure.c

Qs. create a structure to store complex number. (use Arrow operator)

Sol:- complexStructure.c (File.No-99)

Qs. you have to store the marks of 30 students in class. what you use?

(a) array of 30 floats.

(b) structure

Ans:- (a) array of 30 floats

Structure is also solution but used when there are diff types of data has to store.

Q. Make a structure to store Bank Account Information of customers of ABC Bank. Also, make an alias for it.

(name, Acc No)

Sol°: bankAccountStructure.c (file.No:-100)

Home Work Set

(x) Make a System that can store Info of all students, teachers & staff of your college in the form of structures.

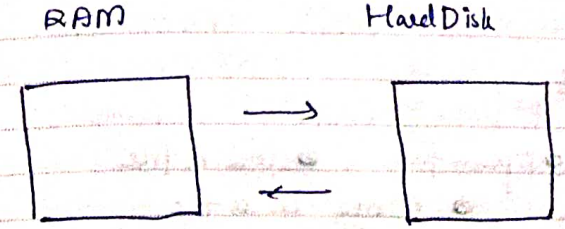
(x) you can also make this into a C project with other functionalities like Cgpa calculations of Students, storing attendence etc.

projectStructure.c (File.No:-102)

CHAPTER - 10

FILE input / output

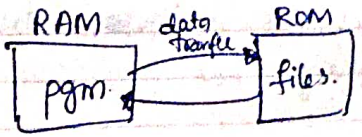
FILE IO



volatile
as soon as the power off the computer the data vanished.

non-volatile
are not affected by computer's power off

Ex:- Microsoft world file's editings



FILE :- Contains in a storage device to store data.

- RAM is volatile
- contents are lost when program terminates.
- Files are used to persists the data.

Operations on file

- Create a file
- Open a file
- close a file
- read from a file
- write in a file.

Types of files

Text files

- textual data
  - .txt
  - .c

Binary files

- binary data
  - .exe
  - .mp3
  - .jpg.

FILE Pointer :-

FILE is a (hidden) structure that needs to be created for opening a file

A FILE ptr that points to this structure & is used to access the file.

ptr used to access the file


```
FILE *fptr;
```

Opening a file

```
FILE *fptr;
fptr = fopen("filename", "mode");
```

Ex:- test.txt (or) hello.c

"mode" → read, write ---

"r".

"test.txt"

Closing a file

```
fclose(fptr);
```

book open → read → book write close

FILE opening modes

- "r" - open to read
  - if file does not exist
    NULL will store to pointer
  - if exist pointer to points to it.
- "rb" - Open to read in binary
- "w" - open to write
  - if file does not exist
    file will automatically created.
  - if file already exist
    it will lost its content.
- "wb" - open to write in binary
- "a" - open to append
  - if file exist,
    it will continue with it

BEST Practice

Check if a file exists before reading from it.

```
FILE *fptr;
fptr = fopen("NewTest.txt", "r");
if (fptr == NULL) {
    printf("file doesn't exist");
} else {
    // reading
}
fclose(fptr);
```


Reading from a file

```
char ch;
fscanf(fptr, "%c", &ch);
```

Reading char

```
FILE *fptr
fptr = fopen("102-test.txt", "r");
if(fptr == NULL){
    printf("FILE Doesnot Exist\n");
} else {
    char ch;
    fscanf(fptr, "%c", &ch);
    printf("%c", ch);      // H
    fscanf(fptr, "%c", &ch);
    printf("%c", ch);      // E
    :                      // L
    :                      // L
    :                      // O
}
fclose(fptr);
```

102-test.txt
HELLO

Reading Integers

```
FILE *fptr;
fptr = fopen("102test.txt", "r");
if (fptr == NULL){
    printf("file does not exist\n");
} else {
    int ch;
    fscanf (fptr, "%d", &ch);
    printf ("%d\n", ch);        //45
    fscanf (fptr, "%d", &ch);
    printf ("%d\n", ch);        //68
    fscanf (fptr, "%d", &ch);
    printf ("%d\n", ch);        //49
    fclose (fptr);
}
```

102-test.txt

45 68 49

Writing to a file

```
char ch = 'A';
fprintf (fptr, "%c", ch);
```

102-test.txt

~~CHETAN~~

A

```
FILE *fptr;
fptr = fopen ("102-test.txt", "w");
if(fptr == NULL){
    printf("File doesn't exist\n");
} else{
    fprintf (fptr, "%c", 'A');
    fclose (fptr);
}
```


Read & Write a char from FILE

```
fgetc (fptr)            //Read

fputc ('A', fptr)       //write
```

```
FILE *fptr;
fptr = fopen ("102-test.txt", "r");
if (fptr == NULL){
    printf ("file doesnot exist\n");
} else{
    printf ("%c\n", fgetc (fptr)); //C
    printf ("%c\n", fgetc (fptr)); //H
                                   //E
                                   //T
                                   //A
                                   //N
    fclose (fptr);
}
```

102-test.txt

CHETAN

```
FILE *fptr;
fptr = fopen ("102-test.txt", "w");
if (fptr == NULL){
    printf ("file doesnot exist\n");
} else{
    fputc ('G', fptr);
    fputc ('O', fptr);
    fputc ('U', fptr);
    fputc ('R', fptr);
    fputc ('I', fptr);
    fclose (fptr);
}
```

102-test.txt

~~CHETAN~~

GOURI

EOF (End of FILE)

* fgetc return EOF to show that the file has ended.

```
char ch = fgetc (fptr);
while (ch != EOF) {
    printf ("%c", ch);
    ch = fgetc (fptr);
}
```

102-test.txt

CHETAN IS AN ENGINEER.

NOTE : if Character then fgetc fputc

int, string } fscanf, fprintf

Practice Qs 61.

make a pgm to read 5 integers from a file.

103-test.txt

1 2 3 4 5

```
FILE *fptr;
fptr = fopen ("103-test.txt", "r");
if (fptr == NULL) {
    printf ("File does not exist\n");
} else{
    int a, ret;
    ret = fscanf (fptr, "%d", &a);
    while (ret != EOF){
        printf ("%d\t", a);
        ret = fscanf (fptr, "%d", &a);
    }
    fclose (fptr);
}
```

OR

```
FILE *fptr;
fptr = fopen ("103-test.txt", "r");
int n;
fscanf (fptr, "%d", &n); {
printf ("%d\n", n);
```

no 2
no 3
no 4
no 5

Practice Qs 62.

Make a pgm to input student info from user & enter into a file.

name
age
marks/cgpa

```
FILE *fptr;
fptr = fopen("104-test.txt", "w");
char name[100];
int age;
float cgpa;
printf("Enter your name: ");          //Chetan
scanf("%s", &name);
printf("Enter your age: ");           //21
scanf("%d", &age);
printf("Enter your cgpa: ");          //9.77
scanf("%f", &cgpa);
fprintf(fptr, "student name: %s\n", name);
fprintf(fptr, "student age: %d\n", age);
fprintf(fptr, "student cgpa: %f\n", cgpa);
fclose(fptr);
```

Auto creation of 104-test.txt

Student name : Chetan
Student age : 21
Student cgpa : 9.770000.

Practice Qs 63

Write a program to write all the odd no's from 1 to n in a file.

```
FILE *fptr;
fptr = fopen("105-oddNos.txt", "w");
int n;
printf("Enter n upto which you need odd nos: ");  //15
scanf("%d", &n);
for(int i=1; i<=n; i++){
    if(i%2 != 0){
        fprintf(fptr, "%d\t", i);
    }
}
fclose(fptr);
```

auto creation of 105-oddNos.txt

1 3 5 7 9 11 13 15

Sunday 28

Practice Qs 69

2 Numbers - a & b, are written in a file, write a program to replace them with their sum.

206-test.txt

~~2 5~~

5

```
FILE *fptr;
fptr = fopen ("206-test.txt","r");
if (fptr == NULL) {
    printf("file doesn't exist\n");
} else {
    int a,b;
    fscanf (fptr, "%d", &a);
    fscanf (fptr, "%d", &b);
    fclose (fptr);
    fptr = fopen ("206-test.txt","w");
    int sum = a+b;
    printf("Sum : %d\n", sum);
    fprintf (fptr, "%d", sum);
}
fclose (fptr);
```


Homework Set

(a) write a pgm to read a string from a file & output to the user.

Sol^n :- 207-printStringFile.c
207-test.txt

where i handled single word string

(b) replace the data in a file of Q(a) with no. of vowels in str.

```
char str[100];
fgets (str, 200, fptr);
--o--
fputs("Hello there\n", fptr);
```

multiword string

Sol^n :- 208-vowelInStrFile.c
208-test.txt.

CHAPTER – 11

Dynamic Memory Allocation

Static MA	Dynamic MA
variables allocate ☐	

(*) but we have that requirements can change at the middle of the program

Ex:- `int a[30];` // [store 30 student marks]
but in between again 3 students enrolled

How you store (i) again 33 student array

`int a[33]`

`int a2[30]` ← `int a[30]` (copy)

31, 32, 33 again at last.

(ii) DMA

increase the array size } at run time
decrease the array size }

DMA is a way to allocate memory to a data structure during run time.

compile-time	run-time
↓	↓
at C-complier assigning a memory for variables, arrays Etc	at user inputting while pgm is running at terminal

We need some functions to allocate & free memory dynamically.

Function for DMA

(i) malloc () – memory Allocation
(ii) calloc () – continuous allocation
(iii) free ()
(iv) realloc () – re-allocation

`#include <stdlib.h>`

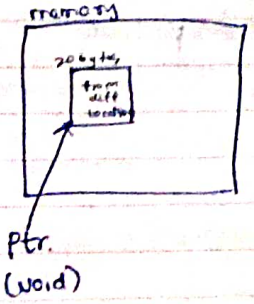
(1) malloc () – memory allocation

takes no. of bytes to be allocated & return a pointer of type void

```
ptr = (*int) malloc(5 * sizeof(int));
```

(5 * 4) bytes

typecast from void ptr ⇒ int ptr



sizeof(int) – int size
sizeof(float) – float size
sizeof(char) – char size
} according machine

4, 4, 1

When malloc asks memory more than computer's memory then malloc will return NULL.

Ex:- ptr = (int *) malloc (500000000000000000 * sizeof(int));

accessing elements is same like array ← indices

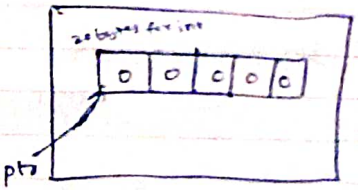
```
#include<stdio.h>
#include<stdlib.h>
void main(){
    int *ptr;
    ptr = (int*) malloc ( 5* sizeof(int));
    ptr[0]=1;
    ptr[1]=3;
    ptr[2]=5;
    ptr[3]=7;
    ptr[4]=9;
    for (int i=0;i<5;i++){
        printf("%d\n", ptr[i]);    // 1 3 5 7 9
    }
}
```


Pg 85 ss WAP to allocate memory to store 5 prices. (float)

Sol:- 110 - floatMalloc.c

Calloc() - continuous allocation

(*) Initialize with 0




```
ptr = (int*) calloc (5, sizeof(int));
```

typecast ↓ void ptr ↓ no. of location ↑ Per location size ↑

```
#include<stdio.h>
#include<stdlib.h>
void main(){
    float *ptr;
    ptr = (float*) calloc ( 5, sizeof(float));
    for(int i=0;i<5;i++){
        printf("%f\n", ptr[i]);
    }
}
```

//0.000000
0.000000
0.000000
0.000000
0.000000

Pg. 85 66 WAP to allocate memory of size n, where n is entered by user.

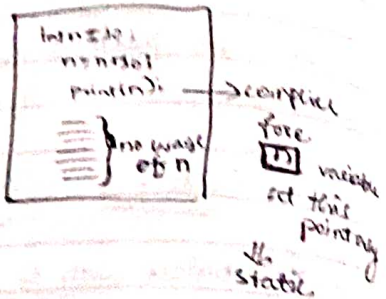
Sol:- 112-callocInput.c

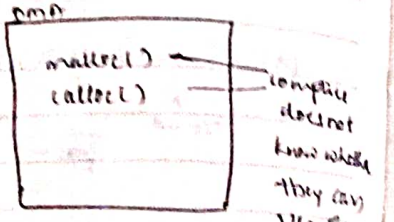
free ()

We use it to free memory that is allocated using malloc & calloc.

free(ptr);

Ex: 113-freeDMA.c





so pgmer has to free the Dynamic memory

Pg 64 & 65: In Q. 65, free the memory allocated with calloc

Soln: 114-freeDMA2.c

realloc () Re allocation

Reallocate (increase or decrease) memory using the same pointer & size.

Ex: 30 Student marks (int*) malloc(30*sizeof(int));

but 3 Extra cans for admission.

here comes into picture realloc

ptr = realloc (ptr, newSize);

Practice Qs 66 :-

Allocate memory for 5 numbers then dynamically increase it to 8 numbers.

```
int *ptr;
ptr = (int*) calloc(5, sizeof(int));
printf("Enter 5 numbers");
for(int i=0; i<5; i++){
    scanf("%d", &ptr[i]);
}
ptr = realloc(ptr, 8);
printf("Enter 8 numbers");
for(int i=0; i<8; i++){
    scanf("%d", &ptr[i]);
}
for(int i=0; i<8; i++){
    printf("%d\n", ptr[i]);
}
free(ptr);
```

PQ.Q.69:-
Create an array of size 5 (using calloc) & enter its values from the user.

Sol^n :- 116-reallocExample2.c

PQ.Q. 70:-
Allocate memory to store first 5 odd numbers, then reallocate it to store first 6 even number.

Sol^n :- 117-reallocOddEven.c

Learn a new language
like Java, C++ etc
because
C does not have Object-oriented

Homework Set

(a) dangling pointer

A dangling pointer is a pointer which points to some non-existing memory location.

Ex-
```
int *ptr;
ptr = (int*) malloc (sizeof(int));
==
--
free(ptr);   // memory is now released
```

but the pointer is still pointing to the deallocated memory because we haven't reinitialized to any other location.

∴ ptr is dangling pointer: (after deallocating memory)

Sol^n :- Reinitialize the pointer.

Ex:-
```
int *ptr;
ptr = (int*) malloc(sizeof(int));
==
--
free(ptr);
ptr = NULL;   // No more dangling-ptr
```

(b) allocate memory for 500 int using calloc & then store first 500 natural numbers in that space.

Sol^n :- 118-callocNaturalNos.c

(c) Memory leak in C

Memory leak in C occurs when a program allocates memory on heap using functions like malloc, calloc, realloc but fails to release it using free(). Over time this cause the program to consume more & more memory, which can lead to performance degradation or even crashes.

```
int *ptr =          (int*) malloc (5* sizeof(int));
ptr = NULL;         // memory leak occurs.
```

Sol^{n} :- use free()

```
int *ptr = (int*) malloc (5* sizeof(int));
---
---
free(ptr);
ptr = NULL;
```

Good practice

(d) better? malloc() (or) Calloc() → both universally better. Choice depends upon your requirements.

malloc()	Calloc()
• Does not initialize memory contain garbage values	• Initializes all allocated memory to zero.
• Takes 1 argument : total size in bytes	• Takes 2 args. no. of elements of size of each.
• Generally faster due to lack of initialization	• Slightly slower because zero-initialization
• Suitable when initialization isn't needed.	• Ideal when zero-initialization required.
• Allocates a single (contiguous) block of memory	• Allocates multiple contiguous block of memory.