CHAPTER 1

STRUCTURES

1. INTRODUCTION

Structure: It is a collection of heterogeneous elements of different data type.

Arrays	Structures
Array is a collection of homogeneous elements of same data type.	Structures is a collection of heterogeneous elements of different data type.
Array is a derived data type.	Structure is a programmed defined type.
Arrays must be declared.	Structures must be designed and declared.

2. DEFINING A STUCTURE

The general format of structure is given below:

Synatx

```
struct tagname
{
        datatype member 1;
        datatype member 2;
        -  -  -  -  -  -  -
        -   -  -  -  -  -  -
        datatype member n;
};
```

- ✓ The keyword struct defines a structure.
- ✓ tagname/ structure tag indicates name of structure.
- ✓ The structure is enclosed within a pair of flower brackets and terminated with a semicolon.
- ✓ The entire definition is considered as statements.
- ✓ Each member is declared independently for its name and type.

Example 1	Example 2

struct bookbank
{
char author[50];
char title[50];
int year;
float price;
};
	struct student
	{
	char name[50];
	int age;
	float marks;
	};

✓ In the above example 1 bookbank is structure name. ✓ The title, author, price, year are structure members.	✓ In the above example 2 student is structure name. ✓ The name, age, marks are structure members

3. DECLARING A STUCTURE VARIABLE

- ✓ After defining a structure format we can declare variable for that type.
- ✓ It includes the following elements:

i. Keyword struct
ii. A structure or tagname
iii. List of variables separated by comma
iv. Terminating semicolon

Syntax: struct tagname list_of_variables;

Example: struct bookbank book1, book2;

struct student s1, s2;

NOTE:

The complete declaration looks like either of below:

	Example	Syntax

1	struct bookbank
	{
	char author[50];
	char title[50];
	int year;
	float price;
	};
	struct bookbank book1, book2;
		struct tagname
		{
		datatype member 1;
		datatype member 2;
		- - - - - - - -
		- - - - - - - -
		datatype member n;
		};
		struct tagname list_of_variables;

2	struct bookbank { char author[50]; char title[50]; int year; float price; } book1, book2;	struct tagname { datatype member 1; datatype member 2; - - - - - - - - - - - - - - - - datatype member n; }list_of_variables;

4. TYPE DEFINED STRUCTURES

We can use the keyword typedef to define the structures as shown below:

Syntax	Example
typedef struct { datatype member 1; datatype member 2; - - - - - - - - - - - - - - - - datatype member n; }tagname; tagname list_of_variables;	typedef struct { char title[50]; char author[50]; int year; float price; }bookbank; bookbank book1, book2;

5. ACCESSING STRUCTURE MEMBERS

- ✓ The structure members should be linked to the structure variables to make it meaningful.
- ✓ It is done with the help of dot (.) operator.

Ex: i) book1.price → Indicates the price of book1.

ii) book2.author → Indicates the author of book2.

In Programs,

Ex: i) printf("Enter bokk1 price");

scanf("%f", &book1.price);

ii) strcpy(book1.author, "Balaguruswamy");

6. STRUCTURE INITIALIZATION

- ✓ The compile time initialization of a structure variable must have the following elements:
  - ➢ The keyword struct
  - ➢ The structure name or tagname
  - ➢ Name of the variable
  - ➢ Assignment operator (=)
  - ➢ The set of values for the members of the structure variables separated by comma and enclosed in flower brackets
  - ➢ Terminating semicolon

Example

```
struct bookbank
{
        char author[50];
        char title[50];
        int year;
        float price;
};
struct bookbank book1 = {"Balaguruswamy", "CPPS", 2021, 200.0};      //Complete
struct bookbank book2 = {"Kulshreshta", "BE", 2020};                 //Partial
```

The structure can be initialized inside a function or outside a function:

Inside the Function

```
main( )
{
        struct bookbank
        {
                char author[50];
                char title[50];
                int year;
                float price;
        };
        struct bookbank book1 = {"Balaguruswamy", "CPPS", 2021, 200.0};      //Complete
        struct bookbank book2 = {"Kulshreshta", "BE", 2020};                 //Partial
}
```

Outside the Function

```
struct bookbank
{
        char author[50];
        char title[50];
        int year;
        float price;
};
main( )
{
        struct bookbank book1 = {"Balaguruswamy", "CPPS", 2021, 200.0};      //Complete
        struct bookbank book2 = {"Kulshreshta", "BE", 2020};                 //Partial
}
```

Rules for initializing structure

1. We cannot initialize individual member inside the structure template.
2. The order of values must match order of members.
3. It is permitted to have partial initialization.
4. The uninitialized members will be assigned to default values as follows:
   0 → int 0.0 → float '\0' → char

7. COPYING AND COMPARING STRUCTURE VARIABLES

- ✓ The variables of the same structure type can be copied the same way as ordinary variable.
- ✓ If book1 and book2 belong to same structure then,

  book2 = book1 or book1 = book2 → valid.
- ✓ However, C does not permit any logical operation on structure variables like,

  book2 == book1 or book1 = = book2 → Invalid.
- ✓ In case we need to compare them we may do so by comparing members individual.

  book1.price = = book2.price → Valid

8. OPERATIONS ON INDIVIDUAL MEMBERS

The individual members are identified using the member operator (dot .).

A member with the dot operator along with its structure variable can be treated like any other variable name.

Ex:

```
i)    if(book2.year = = 2020)
          book2.price = 500;
ii)   if(book1.price = = 200.0)
          book1.price + = 100;
iii)  sum = book1.price + book2.price;
```

Note:

There are 3 ways to access members. They are:

1. Using dot notation [book1.price]
2. Indirect notation -- Using pointers [*ptr.price]
3. Selection notation – This operator [ptr → price]

9. ARRAY OF STRUCTURES

- ✓ We can declare an array of structures, each element of the array representing a structure variable.

Example

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

- ✓ In the above example an array called book is defined, that consist of 100 elements. Each element is defined to be that of type struct bookbank.
- ✓ It can be initialized as follows:

```
struct bookbank
{
        char author[50];
        char title[50];
        int year;
        float price;
};
struct bookbank book[2] = { {"Balaguruswamy", "CPPS", 2021, 200.0}
                            {"Kulshreshta", "BE", 2020, 500.0} };
```

- ✓ In the above example it declares the book as an array of 2 elements that is book[0] and book[1].
- ✓ Each element of book array is a structure variable with 4 members.
- ✓ An array of structures is stored inside the memory in the same way as a multi-dimensional array as shown in the figure below:

book[0].author	Balaguruswamy
book[0].title	CPPS
book[0].year	2021
book[0].price	200.0
book[1].author	Kulshreshta
book[1].title	BE
book[1].year	2020
book[1].price	500.0

10. ARRAYS WITHIN STRUCTURE

✓ Here, the array is present within structure.

Example:

```
struct marks
{
        int rollno;
        float subject[2];
};
struct marks student[3];
```

✓ In the above example the member subject contains two elements subject[0], subject[1].
✓ The elements can be accessed using appropriate subscript like:
student[0].rollno;
student[0].subject[0];
student[0].subject[1]; → Refers to marks obtained in the 2nd Subject by the 1st Student.

11. STRUCTURES WITHIN STRUCTURE

✓ A structure within a structure is called Nested Structure.

Example

```
struct bookbank
{
        char author[50];
        char title[50];
        struct bookbank1
        {
                int year;
                float price;
        }details;
}book;
```

✓ It can be accessed as:

book.author
book.title
book.details.year
book.details.price

12. STRUCTURES AND FUNCTIONS

✓ There are three methods by which the values of a structure can be transferred from one function to another:
i) The first method is to pass each member of structure as an actual argument of the function.
ii) The second method involves passing of a copy of entire structure to called function.
iii) The third method employs a concept called pointers to pass the structure as an argument.

Function Call Syntax:
function_name(structure_variable_name);
Function Definition Syntax:

data_type function_name(struct_type struct_name)
{
- - - - - -
- - - - - -
- - - - - -
return(expression);
};

CHAPTER 2

POINTERS

1. INTRODUCTION

- ✓ A pointer is a variable which stores the address of another variable.
- ✓ A pointer is a derive data type in 'C'.
- ✓ Pointers can be used to access and manipulate data stored in memory.

2. ADVANTAGES OF POINTERS

- ✓ Pointers are more efficient in handling arrays and data tables.
- ✓ Pointers can be used to return multiple values from a function.
- ✓ Pointers allow 'C' to support dynamic memory management.
- ✓ Pointers provide when efficient tool for manipulating dynamic data structures such as stack, queue etc.
- ✓ Pointers reduce length and complexity of programs.
- ✓ They increase execution speed and this reduces program execution time.

3. UNDERSTANDING POINTERS

Memory Cell	Address
	0
	1
	2
	-
	-
	-
	-
	65535

- ✓ The computer memory is a sequential collection of storage cells as shown in the figure above.
- ✓ The address is associated with a number starting of '0'.
- ✓ The last address depends on memory size.
- ✓ If computer system as has 64KB memory then, its last address is 655635.

4. REPRESENATTION OF A VARIABLE

Ex: int quantity = 179;

- ✓ In the above example quantity is integer variable and puts the value 179 in a specific location during the execution of a program.
- ✓ The system always associate the name "quantity" within the address chosen by system. (Ex: 5000)

Pointer Variables

Variable	Value	Address
quantity	179	5000
P	5000	5048

- ✓ Here, the variable P contains the address of the variable quantity. Hence, we can say that variable 'P' points to the variable quantity. Thus 'P' gets the name Pointer.

NOTE 1:

Pointer Constant: Memory addresses within a computer are referred to as pointer constants. We can't change them but, we can store values in it.

Pointer Values: Pointer values is the value obtained using address operator.

Pointer Variables: The variable that contains pointer value.

NOTE 2:

- ✓ Pointer uses two operators:
  1. The address operator (&)
     It gives the address of an object.
  2. The indirection operator (*)
     It is used to access object the pointer points to.

5. ACCESSING THE ADDRESS OF A VARIABLE

- ✓ The actual location of a variable in the memory is system dependent and therefore the address of a variable is not known to us immediately.
- ✓ Therefore the operator (&) and immediately preceding variable returns the address of the variable associated with it.
- ✓ **Example:**

```
int *quantity;
p = &quantity;
```

6. DECLARING POINTER VARIABLE

Example:

```
int *p                  //Declares a pointer variable p of integer type.
float *sum
```

Syntax: data_type *ptrname;

Where,

data_type → It specifies the type of pointer variable that you want to declare int, float, char and double etc.

*(Asterisk) → Tells the compiler that you are creating a pointer variable.

ptrname → Specifies the name of the pointer variable.

7. INITIALIZATION OF POINTER VARIABLE

✓ The process of assigning address of a variable to a pointer variable is known as Initialization.

Syntax: data_type *ptrname &expression

Where,

data_type → It can be any basic datatype.

ptrname → It is pointer variable

expression → It can be constant value or any variable containing value.

Ex:
```
int a;
int *p = &a;          // &a is stored in p variable
```

NOTE:

✓ We can initialize pointer variable to NULL or Zero.

```
int *p = NULL;
int *p = 0;
```

8. POINTER FLEXIBILITY

✓ We can make the same pointer to point to different data variables in different statements.

Ex:
```
int x, y, z, *p;
p = &x;
p = &y;
p = &z;
```

✓ We can also use different pointers to point to the same data variable.

Ex:
```
p1 = &x;
P2 = &x;
P3 = &x
```

9. ACCESSING A VARIABLE THROUGH ITS POINTER

✓ After defining a pointer and assigning the address, it is accessed with the help of unary operator asterisk (*) which is called as indirection or dereferencing operator.

Ex:
```
int quantity, n, *p;
quantity = 10;
p = &quantity;
n = *p;
```

10. POINTERS AND STRUCTURES

```
struct bookbank
{
        char author[50];
        char title[50];
        int year;
        float price;
};
struct bookbank book[2], *ptr;
```

- ✓ The above statement declares book as an array of two elements each of the type bookbank and ptr as a pointer to data objects of the type struct bookbank.
- ✓ Therefore, the assignment ptr = &book; would assign address of 0^{th} element of bookbank to ptr that is ptr will point to book[0].
- ✓ Its members can be accessed using " → " (This or arrow operator or member selection operator)
- ✓ **Ex:** ptr → author;
  ptr → title;
  ptr → year;
  ptr → price;

- ✓ To access all the elements of the book below statement is used:

```
for(ptr = book; ptr<book+2; ptr++)
        printf("%s%s%d%f", ptr → author, ptr → title, ptr → year, ptr → price);
```

CHAPTER 3

PRE-PROCESSOR DIRECTIVES

- ✓ Before a 'C' program is compiled the code is processed by program called pre-processor directives. This is called as pre-processing.
- ✓ Commands used in pre-processor are called pre-processor directives and they begin with #.
- ✓ The pre-processor directives in 'C' are:
  - i) Macro
  - ii) Header file inclusion
  - iii) Conditional compilation
  - iv) Other directives

i) Macro

This macro function defines constant value and can be of basic data type.

Syntax: #define

Example:
```
#include<stdio.h>
#define PI 3.142
void main( )
{
        float area, r;
```

```
        printf("Enter the radius");
        scanf("%f", &r);
        area = PI * r * r;
        printf("Area = %f", area);
}
```

ii) Header file inclusion

The source code of file "File name" is included in the main program at a specified place.

Syntax: #include<file_name>

Example:

```
#include<stdio.h>
void main( )
{
        int a, b, sum;
        printf("Enter two values");
        scanf("%d%d", &a, &b);
        sum = a+b;
        printf("Sum = %d", sum);
}
```

iii) Conditional Compilation

Set of commands are included or excluded in source program before compilation with respect to the condition.

Ex: #ifdef
#ifndef
#if
#ifelse

#ifdef [if define]

- It tests whether the identifier has been defined or substituted text.
- If the identifier is defined then #ifdef is executed else #else will be executed.

Syntax:

```
#ifdef <identifier>
{
        Statement 1;
        Statement 2;
}
#else
{
        Statement 3;
        Statement 4;
}
#endif
```

Example:

```
#include<stdio.h>
#define LINE 1
void main( )
{
        #ifdef LINE
                printf("LINE ONE");
        #else
                printf("LINE TWO");
        #endif
}
```

OUTPUT: LINE ONE

#ifndef [if not define]

- It tests whether the identifier has been defined or substituted text.
- If the identifier is defined then #else is executed else #ifndef will be executed.

Syntax:

```
#ifndef <identifier>
{
        Statement 1;
        Statement 2;
}
#else
{
        Statement 3;
        Statement 4;
}
#endif
```

Example:

```
#include<stdio.h>
#define LINE 1
void main( )
{
        #ifndef LINE
                printf("LINE ONE");
        #else
                printf("LINE TWO");
        #endif
}
```

OUTPUT: LINE TWO

iv) Other Directives

- **#undef [un-define]:** It is used to undefine a defined macro variable.
- **#pragma:** It is used to call a function before and after main function in a program.

VTU SOLVED QUESTIONS

1 **WACP that finds the addition of two squared numbers, by defining macro for Square(x).**

```
#include<stdio.h>
#define square(x) (x*x)
void main( )
{
        Int n1, n2, sum;
        printf("Enter two numbers");
        scanf("%d%d", &n1, &n2);
        sum = square(n1) + square(n2);
        printf("Sum of two squared numbers = %d", sum);
}
```

2 **WACP that accepts a structure variable as a parameters to a function from a function call.**

```
#include<stdio.h>
#include<string.h>
struct student
{
        int rollno;
        char name[50];
        float percentage;
```

```
};
void func(struct student s1);
int main( )
{
        struct student s1;
        s1.id = 1;
        strcpy(s1.name, “Suvika”);
        s1.percentage = 85.5;
        func(s1);
        return 0;
}
void func(struct student s1)
{
        printf(“ID = %d”, s1.id);
        printf(“Name = %s”, s1.name);
        printf(“Percentage = %f”, s1.percentage);
}
```

3 **WACP to add two numbers using pointers.**

```
#include <stdio.h>
int main()
{
  int n1, n2, *p, *q, sum;
  printf("Enter two numbers");
  scanf("%d%d", &n1, &n2);
  p = &n1;
  q = &n2;
  sum = *n1 + *n2;
  printf("Sum = %d", sum);
  return 0;
}
```

4 **WACP to read details of 10 students and to print the marks of the student if his name is given as input.**

```
#include<stdio.h>
#include<string.h>
 struct student
{
        int rollno;
        float marks,
        char name[100], grade[10];
 };
void main()
 {
        struct student s[20];
        int i;
```

```
        char checkname[100];
        for(i=0;i<10;i++)
        {
                printf("Enter the detail of %d students",i+1);
                printf("Enter rollno=");
                scanf("%d",&s[i].rollno);
                printf("Enter marks=");
                scanf("%f",&s[i].marks);
                printf("Enter Name=");
                scanf("%s",s[i].name);
                 printf("Enter Grade=");
                scanf("%s",s[i].grade);
        }
        printf("Enter the student name to check the marks");
        scanf("%s", checkname);
        for(i=0;i<10;i++)
        {
            if((strcmp(checkname, s[i].name)) == 0)
                  printf("The marks of the student is %f", s[i].marks);
        }
}
```

5 **Differentiate between Structures and Unions.**

Structures	Unions
struct keyword is used to define a structure.	union keyword is used to define a union.
Every member within structure is assigned a unique memory location.	In union, a memory location is shared by all the data members.
It enables you to initialize several members at once.	It enables you to initialize only the first member of union.
The total size of the structure is the sum of the size of every data member.	The total size of the union is the size of the largest data member.
We can retrieve any member at a time.	We can access one member at a time in the union.
It supports flexible array.	It does not support a flexible array.

6 **WACP to maintain record of n students using structures with 4 fields (Rollno, marks, name and grade). Print the names of students with marks >= 70.**

```
#include<stdio.h>
 struct student
{
        int rollno;
        float marks,
        char name[100], grade[10];
 };
void main()
 {
        struct student s[20];
        int n,i;
        printf("Enter the number of students");
        scanf("%d",&n);
         for(i=0;i<n;i++)
        {
                printf("Enter the detail of %d students",i+1);
                printf("Enter rollno=");
                scanf("%d",&s[i].rollno);
                printf("Enter marks=");
                scanf("%f",&s[i].marks);
                printf("Enter Name=");
                scanf("%s",s[i].name);
                 printf("Enter Grade=");
                scanf("%s",s[i].grade);
        }
        printf("The students who scored above 70 marks are:");
        for(i=0;i<n;i++)
        {
            if(s[i].marks>=70)
                 printf("%s \t %f", s[i].name, s[i].marks);
        }
 }
```

7 **Explain the array of pointers with examples.**

We can declare array of pointers same as of any other data type. The syntax is:

Syntax: data_type *array_name[size];

Ex: int *x[2];

Here, x is an array of 2 integer pointers. It means that this array can hold the address of 2 integer variables.

Ex:
```
int *x[2];
int a=10, b=20;
x[0] = &a;
x[1] = &b;
```