Section 3.2 of Programming Language and Its Application

Contents:

1. Pointer Arithmetic
2. Pointer and Array
3. Passing Pointer to Function
4. Structure Vs Union
5. Array of Structure
6. Passing structure to Function
7. Structure and Pointer
8. Input/Output Operations on Files
9. Sequential and Random Access to File

Pointers

Introduction

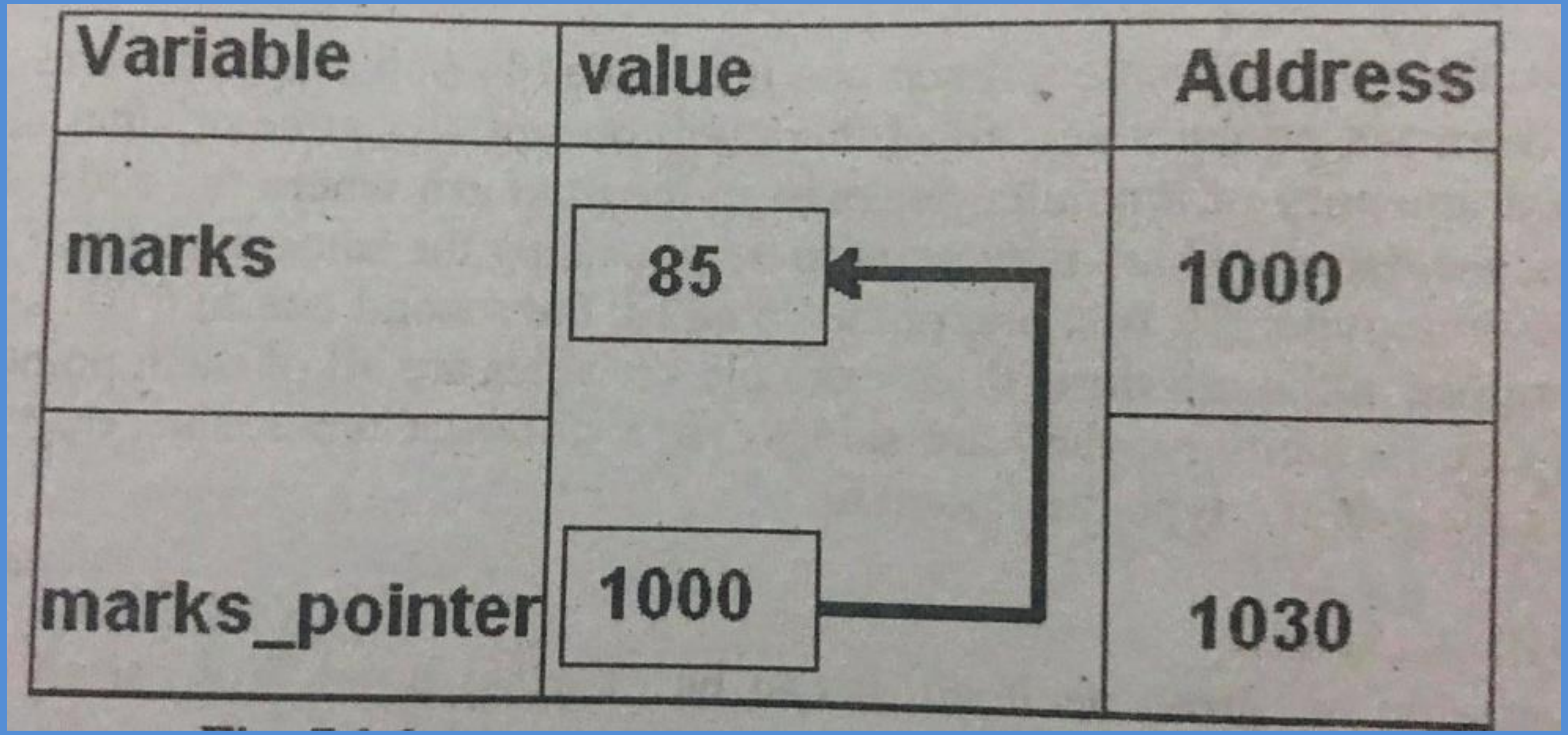
- A pointer is a variable, which contains the address of another variable in memory.
- We can have a pointer to any variable type.
- Pointers are said to "point to" the variables whose reference they store.
- The address of an interger variable num can be assigned to a pointer variable, ptr, say, as in

```
ptr = &num;
```

and we declare ptr as

```
int * ptr;
```

- Here the address of the variable num is assigned to the pointer variable ptr.



Variable	value	Address
marks	85	1000
marks_pointer	1000	1030

- Pointer is very useful in function and dynamic memory allocation.
- By the use of pointer in function we can access the local variables from different function and can logically return more than one value from the function.
- This feature in function is known as pass by reference or call by reference.

Reference Operator(&)

- The address that locates a variable within memory is what we call a reference to that variable.
- This reference to a variable can be obtained be preceding the identifier of a variable with an ampersand sign(&), known as reference operator and which can be literally translated as "address of".

  E.g. m_pointer = &marks assigns the address of marks to m_pointer

Dereference Operator(*)

- Using a pointer we can directly access the value stored in the variable, which it points to.
- To do this, we simply have to precede the pointer's identifier with an asterisk(*), which acts as dereference operator and that can be literally traslated to "value pointed by".
- Thus, & and * have complementary(or opposite) meanings.

  E.g. *m_pointer = *(&marks) = marks

For example:

```
#include <stdio.h>
void test(int *a)
{
        *a = *a + 3;
        printf("%d \n",*a);
}
void main()
{
        int n = 5;
        printf("%d \n",n);
        test(&n);
        printf("%d \n",n);
        test(&n);
}
```

```
OUTPUT:
5 8 8 11
```

Advantages of using pointer in C Programming

1. Pointers are more efficient in handling arrays and data tables.
2. Pointers can be used to return multiple values from a function via function arguments.
3. Pointers allow passing a function as argument to other functions.
4. The use of pointer arrays to character strings results in saving of data storage space in memory.
5. Pointers allow C to support dynamic memory management.
6. Pointers provide an efficient way for manipulating dynamic data structures such as structures, linked lists, queues, stacks, and trees.
7. Pointers increase the execution speed and thus reduce the program execution time.
8. It enables us to access a variable that is defined outside the function.

Pointer Arithmetic

- Pointer arithmetic refers to various operation that can be performed in pointers.

  Declaring the variables as:

  ```
  int a, b, *p, *q;
  ```

Following arithmetic operations in pointer variables are permitted:

i. A pointer variable can be assigned the address of an ordinary variable (e.g. p = &a)

ii. A pointer variable can be assigned content of another pointer variable provided both pointer point to objects of same data type (e.g. p = q)

iii. An integer quantity can be added to or subtracted from a pointer variable. (e.g. p+5,q-1,p++,--p)

iv. A pointer variable can be assigned a null value (e.g. p = NULL)

v. One pointer variable can be subtracted from another provided both pointer point to elements of same array

vi. Two pointer variables can be compared provided both pointer point to element of the same data type

Following operations are not allowed no pointer variables:

i. Pointer variables cannot be multiplied or divided by a constant

ii. Two pointer variables cannot be added.

Equivalence of Array and Pointers

- Array name holds the address of first element of array. So, compiler defines the array name as a constant pointer to the first element.

  Suppose we declare,

  ```
  int x[5] = {2,5,7,9,3};
  ```

  Here, x holds the address of first array element i.e., x = &x[0]

- Pointer is a variable that holds the address of the variable of its own type.

  Suppose we declare

  ```
  int *p;
  ```

  then, p = x is equivalent to p = &x[0]

  Now, we can access every element of x using p

  ```
  e.g., x[0] = 2 and also *p = 2
         x[1] = 5 and also *(p+1) = 5 and so on
  ```

  So, we can say that pointers and arrays are inseparably related but they are not synonyms.

- However, pointers and array are different in many aspects.
- We can assign address of other variable to pointer.
  E.g. p = x and p++ can be done
  But we cannot use name of array as variables.
  E.g. x = p and x++ are illegal operations

There is also difference in accessing the elements.

Array notation	**Pointer notation**
&x[i]	(x+i)
x[i]	*(x+i)
&x[i][j]	*(x+i)+j
x[i][j]	*(*(x+i)+j)

Pointers as Function Arguments

- Passing reference as parameter to function means to pass the address of the variable rather than the value and only pointer variable can hold the address.
- So, pointers play vital role while passing reference as parameter to function.
- By the help of pointer we can change the values of actual argument of calling function from the called function.

For example:

```
#include<stdio.h>
#include<conio.h>
int sum(int *x,int *y)
{
     int z; z = *x
     + *y;
     return z;
}
```

```
void main()
{
     int a = 10, b = 15, s;
     s = sum(&a, &b);'
     printf("Sum = %d\n",s);
     getch();
}
```

Here, x and y are pointer variables.

Array of Pointers

- There may be a situation when we want to maintain an array, which can store pointers to an int or char or any other data type available.
  Following is the declaration of an array of pointers to an integer:

  ```
  int *ptr[3];
  ```

  This declares **ptr** as an array of 3 integer pointers.
  Thus, each element in ptr, now holds a pointers to an int value.

Following example makes use of three integers, which will be stored in an array of pointers as follows:

```
#include<stdio.h>
int main() {

    int var[] = {10,100,200};
    int i, *ptr[3];
    for(i = 0; i < 3; i++)
        ptr[i] = &var[i];     //assign the address of integer
    for(i = 0; i < 3; i++)
        printf("Value of val[%d] = %d \n",i,*ptr[i]);
    return 0;
}
```

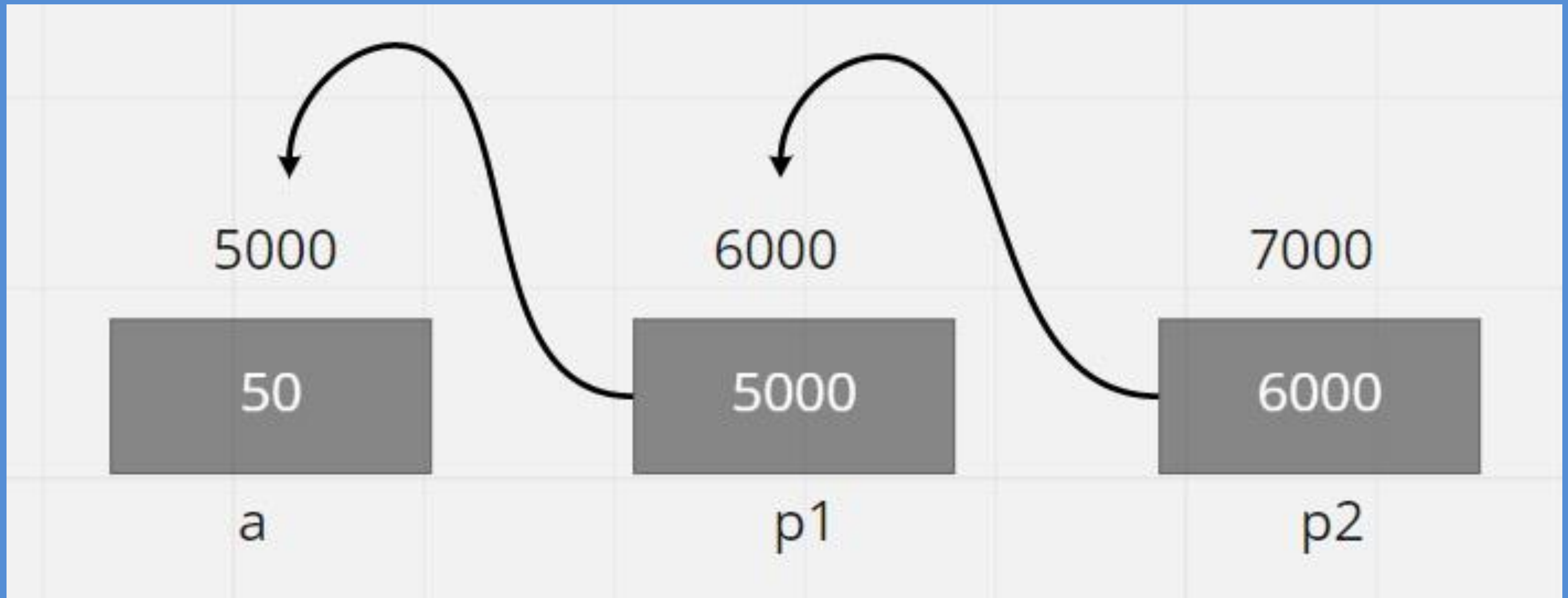
When the above code is compiled and executed, it produces the following result:

```
Value  of  val[0]  =  10
Value of val[1] = 100
Value of val[2] = 200
```

Pointer to Pointer

- A pointer points to a location in memory and thus is used to store the address of variables.
- So, when we define a pointer to pointer, the first pointer is used to store the address of the variable and the second pointer is used to store the address of the first pointer
- Declaring pointer to pointer is similar to declaring pointer in C. The difference is we have to place an additional '*' before the name of pointer.

Consider the figure given below where a is an integer variable, p1 is an integer pointer, whereas p2 is pointer to pointer(double pointer).



```
int a = 50;
int *p1 = &a; \\p1 is a pointer that points to address of a
int **p2 = &p1; \\p2 is a pointer that points to address of p1
```

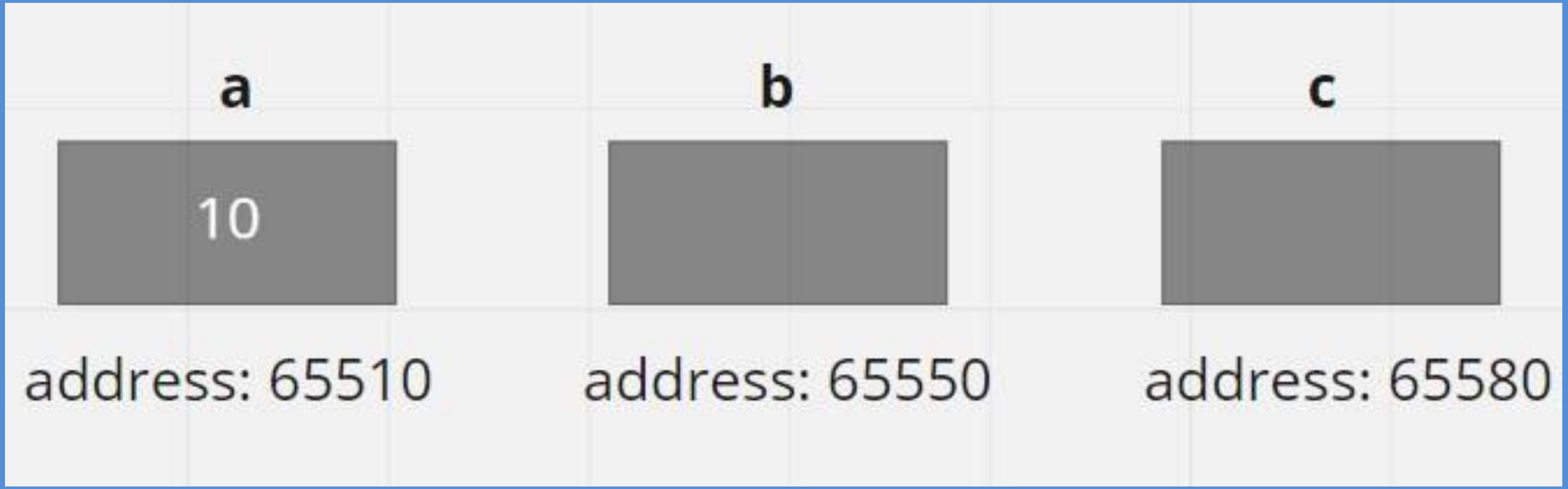
We can acess the value of a from both p1 and p2 as follows:

*p1 = a = 50 and **p2 = *(*p2) = *(p1) = a = 50

In this way we can declare chain of pointers also.

Q. Write the output of the following:

```
int a = 10, *b, **c;
b = &a; c = &b;
printf("%d \t %d \n",b, *c);
printf("%d \t %d \n",c,**c);
printf("%d \t %d \n",*b+5,&c+2);
```



Types of Pointer

1.Bad Pointer/ Wild Pointer

- A pointer is said to be a bad pointer if it is not being initialized to anything.
- These types of pointers are not efficient because they may point to some unknown memory location which may cause problems in our program and it may lead to crashing of the program.
- One should always be careful while working with bad pointers. Pointers should always be
- initialized with some valid address.

2.Null Pointer

- We can create a null pointer by assigning null value during the pointer declaration.
- This method is useful when you do not have any address assigned to the pointer.
- A null pointer always contains value 0. e.g., int *p = NULL or int *p = 0;

3.Void Pointer/Generic Pointer

- In C Programming, void pointer is also called as a generic pointer.
- It does not have any standard data type.
- A void pointer is created by using the keyword void.
- It can be used to store an address of any variable.

  e.g., void *p;

  Here p is a void pointer which can store the address of any type of variable. Before dereferencing void pointers, we have to typecast them to the particular type as shown below:

```
#include<stdio.h>
int main() {
    int a = 5;
    void *p;
    p = &a;
    printf("Value of *p = %d",*(int *)p);
 }
```

Output:

```
Value of *p = 5
```

Multiple Choice questions of Pointer

1.What is the output of the following code?

```
int a = 10;
int *p = &a;
printf(“%p”, p);
```

a) 10
b) 0
c) Address of a
d) Undefined

Ans: c

2. What is the output of the following code?

```
int a = 10;
int *p = &a;
printf(“%d”, *p);
```

a) 10
b) 0
c) Address of a
d) Undefined

Ans: a

3. What is the correct way to declare a pointer to a character in C?

a) char *str;
b) char *str();
c) char str*;
d) char (*str)();

Ans: a

4. What is the output of the following code?

```
int arr[5] = {1,2,3,4,5};
int *p = arr;
printf(“%d”, *(p+3));
```

a) 3
b) 4
c) 5
d) undefined

Ans: b

5.What is the output of the following code?

```
int a = 10;
int *p = &a;
int **q = &p;
printf(“%d”, **q);
```

a) 10
b) 0
c) Address of a
d) Undefined

Ans: a

6. What will be the output of the following C code?

```
#include <stdio.h>
int x = 0;
void main()
{
    int *ptr = &x;
    printf("%p\n", ptr);
    x++;
    printf("%p\n ", ptr);
}
```

a) Same address
b) Different address
c) Compile time error
d) Varies

7. Which of the following does not initialize ptr to null (assuming variable declaration of a as int a=0;)?

a) int *ptr = &a;
b) int *ptr = &a – &a;
c) int *ptr = a – a;
d) All of the mentioned

8. Which of the following is the valid way to declare a pointer to an integer array of size 5?

a) int *arr[5];
b) int **arr[5];
c) int arr*[5];
d) int (*arr)[5];

9. What will be the output of the following C code?

```
#include <stdio.h>
    void foo(int*);
    int main()
    {
        int i = 10, *p = &i;
        foo(p++);
    }
    void foo(int *p)
    {
        printf("%d\n", *p);
    }
```

a) 10
b) Some garbage value
c) Compile time error
d) Segmentation fault

10. What will be the output of the following C code?

```
    #include <stdio.h>
    void foo(int *);
    int main()
    {
        int i = 10;
        int *p = &i;
        foo(p);
        printf("%d ", *p);
    }
    void foo(int *p)
    {
        int j = 11;
        p = &j;
        printf("%d ", *p);
    }
```

a) 11 11
b) 11 10
c) Compile time error
d) Undefined-value

11.Which of the following is true about pointer arithmetic in C?

a) Addition and subtraction operations can be performed on pointers
b) Multiplication and division operations can be performed on pointers
c) Bitwise AND and OR opeations can be performed on pointers
d) None of the above

12. What is the size of *ptr in a 32-bit machine (Assuming initialization as int *ptr = 10;)?

a) 1
b) 2
c) 4
d) 8

13. What will be the output of the following C code?

```
#include <stdio.h>
void main()
{
    char *s = "hello";
    char *p = s;
    printf("%c\t%c", *p, s[1]);
}
```

a) e h
b) Compile time error
c) h h
d) h e

14. How can we pass a pointer to a function as an argument?

a) By reference
b) By value
c) By address
d) None of the above

15. How to call a function without using the function name to send parameters?

a) Function pointer

b) typedefs

c) Both typedefs and Function pointer

d) None of the mentioned

16. Which of the following operator is used to access the function pointer?

a) &

b) *

c) ->

d) None of the above

17. Which of the following is a correct syntax to pass a Function Pointer as an argument?

a) void pass(*fptr(int, float, char)){}

b) void pass(int (*fptr)){}

c) void pass(int (*fptr)(int, float, char)){}

d) void pass(*fptr){}

18. What will be the output of the following C code?

```
#include <stdio.h>
int mul(int a, int b, int c)
{
    return a * b * c;
}
void main()
{
    int (*function_pointer)(int, int, int);
    function_pointer  =  mul;
    printf("The product of three numbers is:%d",
    function_pointer(2, 3, 4));
}
```

a) The product of three numbers is:24

b) Run time error

c) Nothing

d) Varies

19. What will be the output of the following C code?

```
#include <stdio.h>
void main()
{
    char *a[10] = {"hi", "hello", "how"};
    int i = 0, j = 0;
    a[0] = "hey";
    for (i = 0;i < 10; i++)
    printf("%s\n", a[i]);
}
```

a) hi hello how Segmentation fault
b) hi hello how followed by 7 null values
c) hey hello how followed by 7 null values
d) hey hello how Segmentation fault

20. If ‘ptr’ is a pointer to an integer variable, what will be output of the following code?

```
printf(“%d”, *(ptr+1);
```

a) The memory address of ptr
b) The value of ptr
c) The memory address of the integer variable ptr is pointing to
d) The value of the integer variable ptr+1 is pointing to

Structures

Introduction

- Structures is a collection of data items of same or different data type in contiguous memory location.
- It is a heterogeneous, user defined data type. Structures can store non-homogenous data types into a single collection.
- Structures may contain pointers, arrays or even other structures other than the common data types (such as int, float, long int, char etc.)
- It is a convenient way of grouping several pieces of related information together.

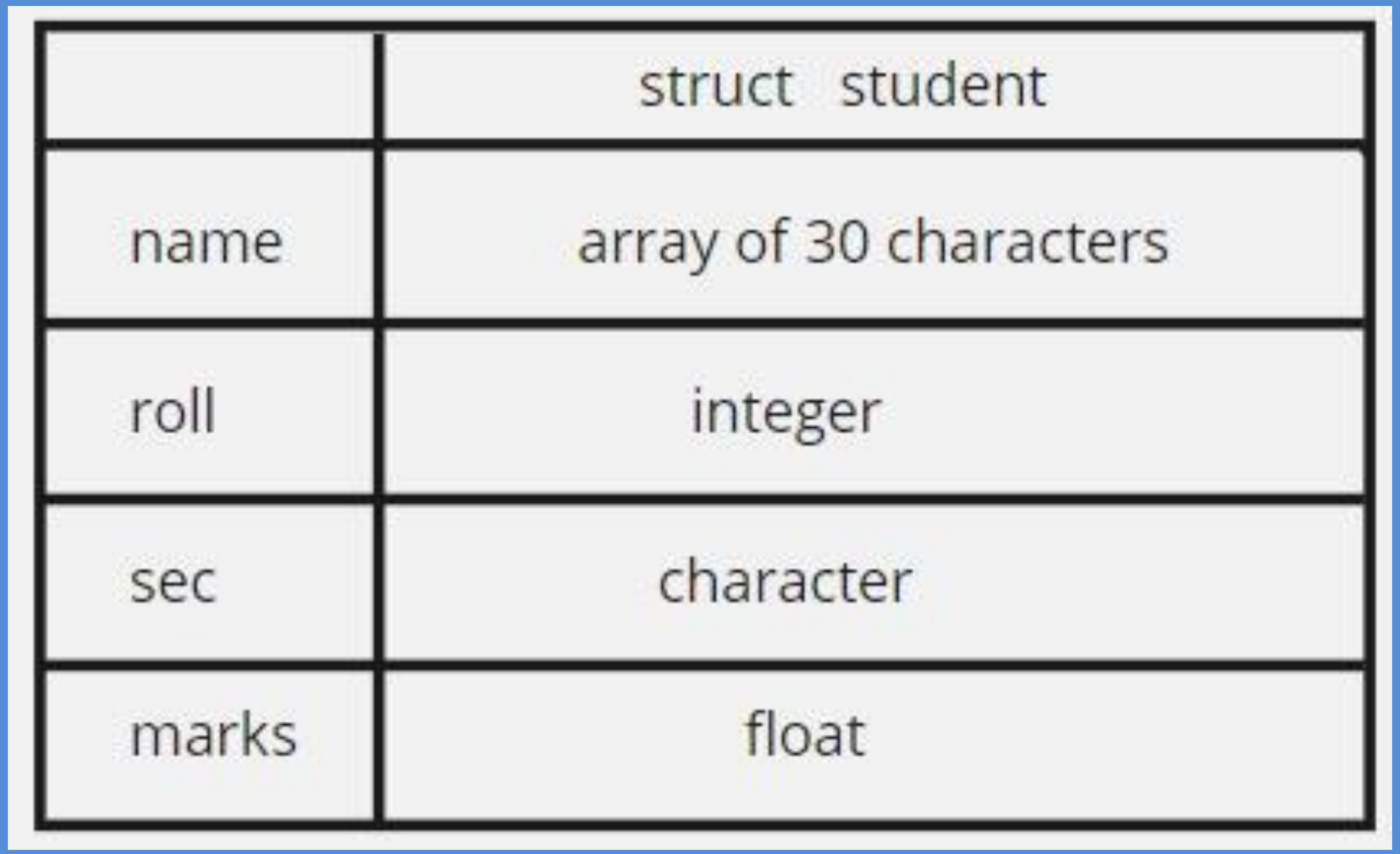
Syntax: A structure is defined or declared as:

```
struct tag_name
{
        data_type    member1;
        data_type member2;
        ..................    ..................
        ..................    ..................
        data_type member n;
};
   struct tag_name variable_name;
```

Example: Let us define a student information system consisting name, roll number, section and average marks in final exam. We define the structure to hold this information as follows:

```
struct student
{
    char name[30];
    int  roll;
    char sec;
    float marks;
};
struct student s;
```

Fig: A sample template of a structure



	struct student
name	array of 30 characters
roll	integer
sec	character
marks	float

Here student is the struct name (also known as tag name) whereas name, roll, sec and marks are its members. There is no any memory allocation until we declare the struct variable. Here s is a structure type variables whose type is "struct student".

The advantages of using structure are as follows:

- To create new data type according to the need of the user.
- It is useful for applications that needs a lot more features than those provided by primitive data types.
- It is convenient way of grouping several pieces of related information together.
- Structure could be used to hold the location of points in multidimensional space.

Structure is an user-defined data type which can be used to store various information related to an entity at one place. This information can be of different data types. But in array we can store information of same data type only. Hence structure should be preferred over array it is a convenient way of grouping several pieces of related information together and we can model the real world problem more effectively.

Accessing members of a structure

There are two types of operators to access members of a structure. They are:

- Member operator (dot operator or period operator(.))
- Structure pointer operator (->)

1) Member operator (.):
It is used to access the members of structure if structure variable is declared as a normal variable.

Syntax:

```
structure_variable.member_name
```

Example:

```
#include<stdio.h>
struct employee

{
    char ename[30];

    int   sal;
};
```

```
void main()
{
        struct employee e = {"Shyam", 50000} ;
        printf( " Name = %s \n " , e.ename ); // using dot operator to acess the member
        printf( " Salary = %d \n " , e.sal );
}
```

OUTPUT:

```
Name = Shyam
Salary = 50000
```

2) Structure pointer operator (Arrow pointer):

It is used to access the members of the structure if structure variable is declared as pointer variable.

Syntax:

```
        structure_pointer_variable-> member_name
```

Example:

```
#include<stdio.h>
struct employee {
    char ename[30];
    int sal;
};
void main()
{
     struct employee e = {"Shyam", 50000} ;
     struct employee *p;
     p = &e;
     printf( " Name = %s \n " , p->ename ); // using arrow operator to acess the member
     printf( " Salary = %d \n " , p->sal );
}
```

OUTPUT:

```
Name = Shyam
Salary = 50000
```

Array of Structure

- An array of structure can be defined as the collection of multiple structure variables where each variable contains information about different entities.
- The array of structures in C are used to store information about multiple entities of different data types.
- The array of structures is also known as the collection of structures.

Example: An array of structures that stores information of 5 students and prints it.

```
#include<stdio.h>
struct student
{

    int rollno;
    char name[10];

};
```

```
void main()
{
    int i;
    struct student st[5];
    printf("Enter Records of 5 students: ");
    for(i=0; i<5; i++)
    {
        printf("Enter roll no.: ");
        scanf("%d",&st[i].rollno);
        printf("Enter name:");
        scanf("%s",st[i].name);
    }
    printf("Student Information List:");
    for(i=0;i<5;i++)
    {
        printf("Rollno.: %d, Name:%s",st[i].rollno,st[i].name);
    }
}
```

Output:

```
Enter Records of 5 students
Enter rollno.: 1
Enter name: Ram
Enter rollno.: 2
Enter name: Shyam
Enter rollno.: 3
Enter name: Vimal
Enter rollno.: 4
Enter name: Sita
Enter rollno.: 5
Enter name: Gita
Student Information List:
Rollno.:1, Name: Ram
Rollno.:2, Name: Shyam
Rollno.:3, Name: Vimal
Rollno.:4, Name: Sita
Rollno.:5, Name: Gita
```

Array within Structure

- We can use arrays of any type as the member of structure according to our programming requirement.
- Arrays within structures can be of single structure or of an array of structures
- The following program shows a structure definition having an integer type array to read marks of 5 subjects of a student and display the total marks:

```
#include<stdio.h>
#include<conio.h>
struct student
{
    int marks[5]; //array within structure
};
void main()
{
    struct student s;
    int i, sum=0;
    for(i=0; i<5; i++)
    {
        printf("marks of subject %d",i+1);
        scanf("%d",&s.marks[i]);
        sum = sum+s.marks[i];
    }
    printf("total marks = %d",sum);
    getch();
}
```

Nested Structure

- Structure within a structure i.e. a structure definition having another structure as a member is known as nested structures or nesting of structure.

For example, person and student are two structures; person is used as a member of student(i.e. person within student):

```
struct person
{
    char name[50];
    int age;
};
struct student
{
    int roll; char
    sec; struct
    person p;
}s;
```

Equivalent form of nested structure:

```
struct student
{
    int roll;
    char sec;
    struct person
    {
        char name[50];
        int age;
    }p;
}s;
```

Members can be accessed as: s.roll, s.sec, s.p.name, s.p.age

Passing Structure to Function

- We can pass an entire structure variable as a function argument to send structure members to a function. It can be both pass by value or pass by reference:
  a. Pass by value
  b. Pass by reference

Simple Structure to function

```
struct student
{
	int roll;
	char name[20];
}std[5];
void display(struct student s)
{
	printf("Roll no.:%d, Name:%s\n",s.roll,s.name);
}
void main()
{
	struct student std[5] = {{2,"Ram"},{3,"Hari"},{4,"Shiva"},{5,"Bishnu"},{6,"Ramesh"}};
	int i;
	display(std[2]);
}
```

Structure Pointer to function

```
struct student
{
    int roll;
    char name[20];
}std[5];
void display(struct student *s)
{
    printf("Roll no.:%d, Name:%s\n",s->roll,s->name);
}
void main()
{
    struct student std[5] = {{2,"Ram"},{3,"Hari"},{4,"Shiva"},{5,"Bishnu"},{6,"Ramesh"}};
    int i;
    for(i = 0; i < 5; i++)
        display(&std[i]);
}
```

Array of a Structure to function

```
struct student
{
    int roll;
    char name[20];
}std[5];
void display(struct student s[])
{
    for(i = 0; i < 5; i++)
        printf("Roll no.:%d, Name:%s\n",s[i].roll,s[i].name);
}
void main()
{
    struct student std[5] = {{2,"Ram"},{3,"Hari"},{4,"Shiva"},{5,"Bishnu"},{6,"Ramesh"}};
    int i;
    display(std);
}
```

Structure Array as Pointer to function

```
struct student
{
    int roll;
    char name[20];
}std[5];
void display(struct student *s)
{
    for(i = 0; i < 5; i++)
        printf("Roll no.:%d, Name:%s\n",(s+i)->roll,(s+i)->name);
}
void main()
{
    struct student std[5] = {{2,"Ram"},{3,"Hari"},{4,"Shiva"},{5,"Bishnu"},{6,"Ramesh"}};
    int i;
    struct student *ptr;
    ptr = std;
    display(ptr);
}
```

Typedef and Structures

- The C programming language provides a keyword called **typedef**, which you can use to give a type a new name.
- For example:

  typedef int kec; //this allows us to declare **int** type variable using **kec**
  kec a,b; ---> This creates two integer variables a and b;

- Similarly, you can use typedef to give a name to your user-defined data types as well.

```
#include<stdio.h>
#include<string.h>
typedef struct Books
{
    char title[50];
    char author[50];
    int book_id;
} Book;
```

```
int main( )
{
    Book b;
    strcpy( b.title, "C Programming");
    strcpy( b.author, "Nuha Ali");
    b.book_id = 6495407;
    printf( "Book title : %s\n", b.title);
    printf( "Book author : %s\n", b.author);
}   printf( "Book book_id : %d\n", b.book_id);
```

Union

Just like a structure, a union is also a user-defined data type that groups together logically related variables into a single unit.

Almost all the properties and syntaxes are same, except some of the factors.

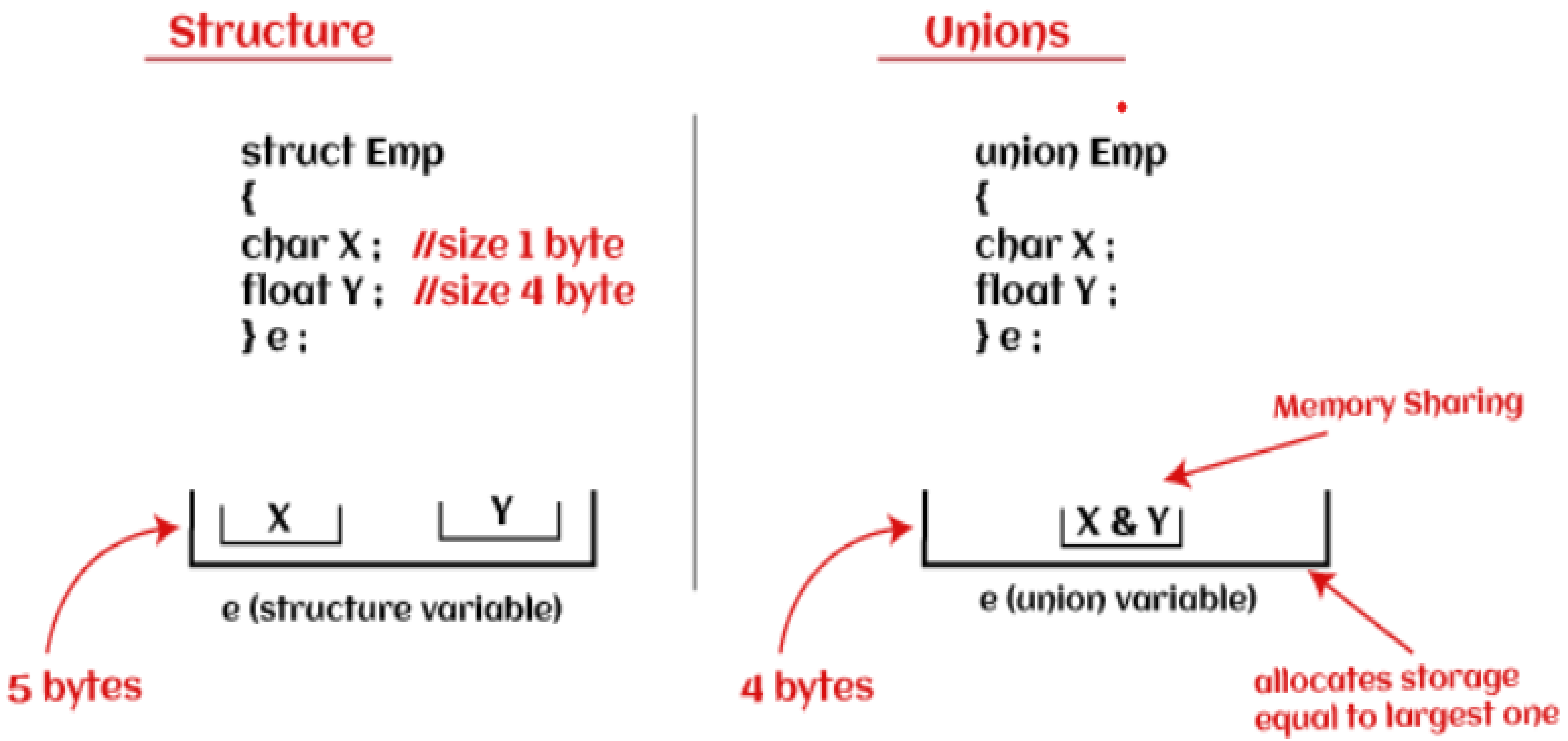
Syntax:

```
union union_name
{
data_type variable_name;
data_type variable_name;
……..
data_type variable_name;
};
```

Example:

```
union Emp
{
   char address[50];
   int dept_no;
   int age;
};
```

Memory allocation in a union:



Structure

```
struct Emp
{
char X ;   //size 1 byte
float Y ;   //size 4 byte
} e ;
```

Unions

```
union Emp
{
char X ;
float Y ;
} e ;
```

Structures	Unions
The keyword "struct" is used to define a structure.	The keyword "union" is used to define a structure.
A unique memory location is given to every member of a structure	One single memory location is shared by all its members
Modifying the value of one item won't affect the other items or the structure at all.	Modifying one single data item affects other members of the union thus affecting the whole unit.
We initialize several members at once.	We can 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
It is mainly used for storing various data types	It is mainly used for storing one of the many data types that are available.
It occupies space for each and every member written in inner parameters	It occupies space for a member having the highest size written in inner parameters
We can retrieve any member at any time	We can only access one member at a time in the union.

Multiple Choice questions of Structure and Union

1.Which of the following are themselves a collection of different data types?

a) string

b) structures

c) char

d) all of the mentioned

Ans: b

2.Which operator connects the structure name to its member name?

a) –

b) <-

c) .

d) Both <- and .

Ans: c

3. Which of the following cannot be a structure member?

a) Another structure

b) Function

c) Array

d) None of the mentioned

Ans: b

4. User-defined data type can be derived by___________

a) struct

b) enum

c) typedef

d) all of the mentioned

Ans: d

5. What will be the output of the following C code?

```
#include <stdio.h>
struct student
{       int no;
    char name[20];
}
void main()
{
    struct student s;
    s.no = 8;
    printf("hello");
}
```

a) Compile time error

b) Nothing

c) hello

d) Varies

6. What will be the output of the following C code?

```
#include <stdio.h>
void main()
{       struct student
    {
        int no;
        char name[20];
    };
    struct student s;
    s.no = 8;
    printf("%d", s.no);
}
```

a) Nothing

b) Compile time error

c) Junk

d)8

7. Which of the following is an incorrect syntax for pointer to structure?

```
struct temp{
        int b;
}*my_struct;)
```

a) *my_struct.b = 10;

b) (*my_struct).b = 10;

c) my_struct->b = 10;

d) Both *my_struct.b = 10; and (*my_struct).b = 10;

8. Which of the following is an incorrect syntax to pass by reference a member of a structure in a function?

```
struct temp
{
    int a;
}s;
```

a) func(&s.a);

b) func(&(s).a);

c) func(&(s.a));

d) none of the mentioned

9. How many bytes in memory taken by the following C structure?

```
#include <stdio.h>
    struct test
    {
        int k;
        char c;
    };
```

a) Multiple of integer size
b) integer size+character size
c) Depends on the platform
d) Multiple of word size

10. Which of the following operation is illegal in structures?

a) Typecasting of structure
b) Pointer to a variable of the same structure
c) Dynamic allocation of memory for structure
d) All of the mentioned

11. Presence of code like “s.t.b = 10” indicates __________

a) Syntax Error
b) Structure
c) double data type
d) An ordinary variable name

12. What is the correct syntax to declare a function foo() which receives an array of structure in function?

a) void foo(struct *var);
b) void foo(struct *var[]);
c) void foo(struct var);
d) none of the mentioned

13. What will be the output of the following C code?

```
#include <stdio.h>
struct student
{
};
void main()
{
    struct student s[2];
    printf("%d", sizeof(s));
}
```

a) 2

b) 4

c) 8

d) 0

14. The correct syntax to access the member of the ith structure in the array of structures is?

Assuming: struct temp

```
{
    int b;
}s[50];
```

a) s.b.[i];

b) s.[i].b;

c) s.b[i];

d) s[i].b;

15. Which of the following is the correct syntax to pass a structure by pointer to a function in C?

a) structure_pointer->member_name

b) *structure_pointer.member_name

c) Strucuture_pointer.member_name

d) None of the above

16. Which of the is the correct syntax to pass a structure by value to a function in C?

a) function_name(&structure_variable)

b) function_name(*structure_variable)

c) function_name(structure_variable)

d) function_name(&structure_variable.member_name)

17. How do you declare a pointer to an array of structures in C?

a) struct *student[5];
b) struct (student*)[5];
c) struct *(student)[5];
d) struct (*student)[5];

18. What is the size of the following structure?

```
struct student{
        char name[50];
        int age;
        double gpa;
};
```

a) 54 bytes
b) 60 bytes
c) 62 bytes
d) 64 bytes

19. What is the purpose of union in C?

a) To combine multiple structures
b) To create a new data type
c) To declare a variable
d) To share the same memory location for different data types

20. In a union, how much memory is allocated for the union itself?

a) Size of the largest member
b) Size of the smallest member
c) Size of all the members combined
d) Size of union is not allocated

Files

Introduction

- The file is a place on the disk where a group of related data is stored.
- The programs, that we executed so far accepts the input data from the keyboard at the time of execution and writes outputs to the visual display unit. This type of I/O is called console I/O.
- For those operations, We have been using printf(), scanf(), getche(), getch(), puts(), gets() etc. functions. Console I/O works fine as long as the amount of data is small.
- But, in many real life problems involve a large volume of data. In such situations, the console oriented I/O operations have two main problems:

  - It becomes very inconvenient and time consuming to handle the large volume of data through the terminal.
  - The entire data is lost when either the program is terminated or the computer system is turned off.

- To overcome these difficulties, we need to create and use files.
  We combine all the input data into a file and then design a C program to read the data from the file whenever required.

Types of data files:

There are two types of data files: high level (stream oriented, standard) and low level (system oriented).

In this chapter, we will study only high-level files. High level data files are further sub divided into two categories:

- Text Files
- Binary Files

1) Text Files:

- A text file is a human-readable sequence of characters and the words they form that can be encoded into computer-readable formats such as ASCII.
- A text file contains only textual characters, with no special formatting.
- There is no graphical information, sound or video files.
- Text files stores information in consecutive characters that can be interpreted as individual data items or as a component of strings or numbers.
- A good example of text file is any C program file.

2) Binary Files:

- Binary files contain more than plain text i.e. sound, image, graphic, etc, which is quite different as compared to text files
- A binary file is made up of machine-readable symbols that represent 1's and 0's.
- The binary file contents must be interpreted by the program that understands in advance how it is formatted.
- These files organize data into blocks containing contiguous bytes of information
- These blocks represent more complex data structures e.g., arrays and structures.
- A good example of binary file is the executed C program file, for example **first.exe**.
- A very easy way to find out whether a file is a text or binary file is to open that file in Turbo C/C++. If we can read the contents of the file, it is a text file else binary file.
- Other examples of binary files are sound files, graphic files, etc.

Basic File Operations:

1. Naming a file
2. Opening a file
3. Reading data from a file
4. Writing data to the file
5. Closing a file

There are two distinct ways to perform file operations in C. The first one is low level I/O and UNIX system calls. The second method is referred to as high level I/O operations and uses functions in C's standard I/O library.

Opening a File

- Opening a file is the process of loading the file stream to modify or review a file.
- We can use **fopen( )** function to create a new file or open an existing file.
- This call will initialize an object of the type FILE new or existing which contains all the information necessary to control the stream (pointer to input and edit data from a file)
- The syntax of this function is as follows:

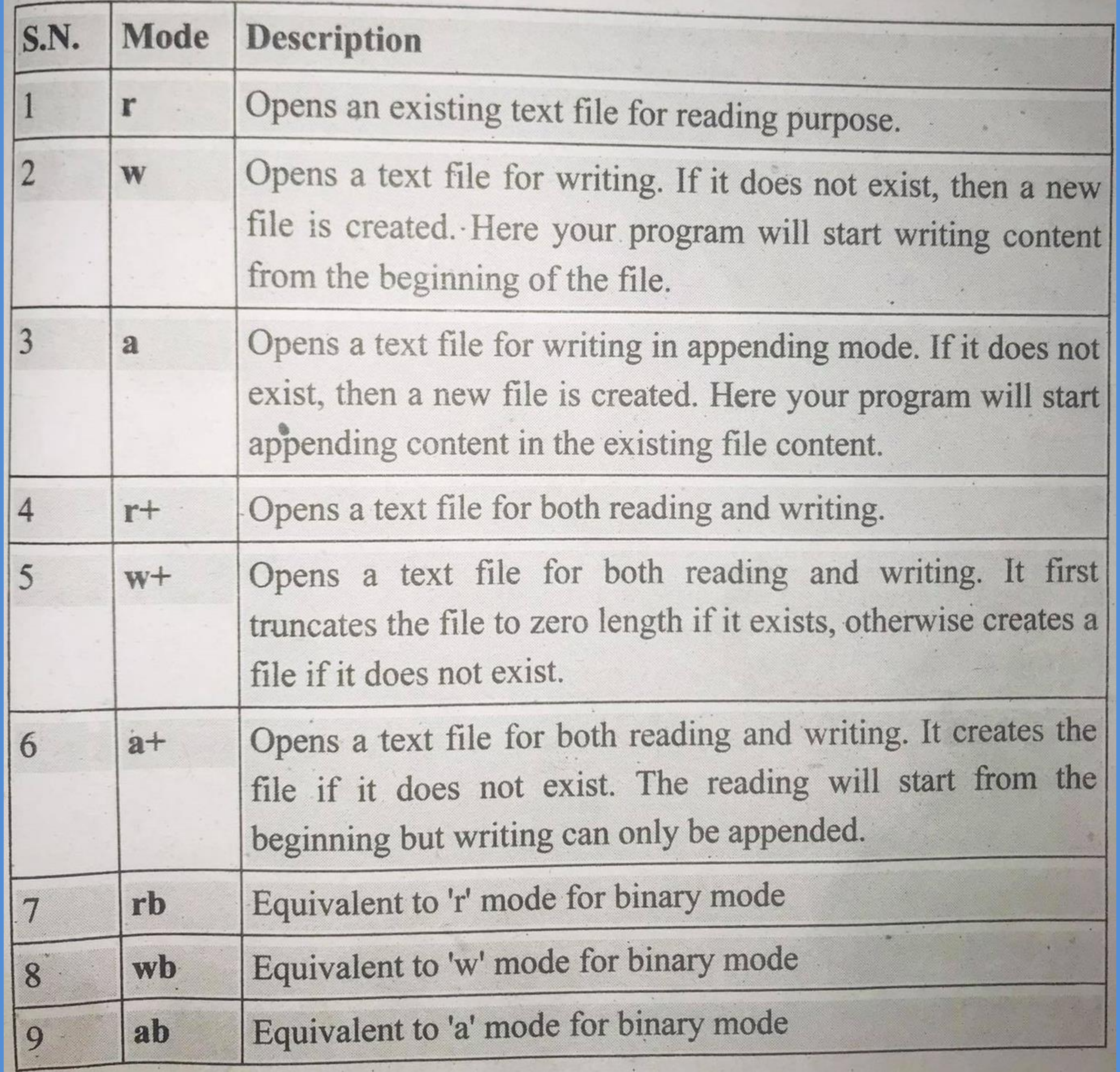
  file_pointer = fopen("file name","opening mode");

Example:

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

Here, filename is a string which you will use to name your file and access mode (here "r" can have one of the following values:

File Opening Modes



S.N.	Mode	Description
1	r	Opens an existing text file for reading purpose.
2	w	Opens a text file for writing. If it does not exist, then a new file is created. Here your program will start writing content from the beginning of the file.
3	a	Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here your program will start appending content in the existing file content.
4	r+	Opens a text file for both reading and writing.
5	w+	Opens a text file for both reading and writing. It first truncates the file to zero length if it exists, otherwise creates a file if it does not exist.
6	a+	Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.
7	rb	Equivalent to 'r' mode for binary mode
8	wb	Equivalent to 'w' mode for binary mode
9	ab	Equivalent to 'a' mode for binary mode

Closing a File

- Closing the file is the process of terminating the file stream to close or exit the modification or review of a file.
- To close a file, use the fclose() function. The syntax of this function is

  fclose(file_pointer);

  where, file_pointer points to the file that is to be closed.
- If successful, the function returns a zero and a non-zero value is returned if an error occured

Writing to a File

- Before writing something to a file, it must be opened in write mode.
- Following is the syntax to write individual characters to a stream

  ```
  fputc(character, file_pointer);
  ```

  The fputs() function writes the character value of the argument to the output file referenced by file pointer. **It returns the written character written on success, otherwise EOF if there is an error.**

- The following function syntax allows to write string line to the filestream

  ```
  fputs(string, file_pointer);
  ```

- You can also use

  ```
  fprintf(file_ptr, "control string", variables);
  ```

  to write any type of data in the file.

Reading a File

- To read information from a file, the file must be opened in read mode.
- Given below is the syntax to read a single character from a file

  char c = fgetc(file_pointer);

  The fgetc() function reads a character from the input file referenced by fp, the return value is the character read.
- The following function syntax allows to read a string from a stream

  fgets(string variable, int n, file_pointer);

  where, it reads up to n-1 characters from the file referenced by the file pointer and copies to the string variable.
- You can also use

  fscanf(file_pointer, string variable); (this syntax valid for string only)

  to read strings from a file, but it stops reading after encountering the first space character.

Input/Output function for file I/O

1) fscanf () : Formatted input function that read data items from a file.

Syntax: fscanf(file_pointer, "control string", address_of_variable);

2) fprintf () : Formatted output function that write data items to a file.

Syntax: fprintf(file_ptr, "control string", variables);

3) fread () : Unformatted input function that read data items from a file.

Syntax: fread(address_of_data_item, size_of_data_item, n, file_ptr);

where, n is number of data item to be read at a time.

4) fwrite () : Unformatted output function that write data items to a file.

Syntax: fwrite(address_of_data_item, size_of_data_item, n, file_ptr);

where, n is number of data item to be written at a time.

5) fputc () : Writes a character to a file.

Syntax: fputc (character, file_ptr);

6) fgetc () : Reads a character from a file.

Syntax: character_variable = fgetc (file_ptr);

7) fgets () : Reads a string from a file.

Syntax: fgets (string_variable, no_of_characters+1, file_ptr);

8) fputs : Write a string variable to a file.

Syntax: fputs (string_variable, file_ptr);

Random Access in File

- The contents in the file can be randomly accessed using the functions fseek, ftell and rewind available in I/O library.

1.ftell() - returns a number of the type long that corresponds to the current position of a file pointer.

Syntax: variable = ftell(file_pointer);

2. rewind() - takes a file pointer to the start of the file

Syntax: rewind(file_pointer);

3. fseek() - move the file pointer to the desired location within a file

Syntax: fseek(file_pointer, offset, position) here, offset is a number or variable that specifies the number of positions (bytes) to be moved from the location specified by position. Position can take one of the three values:

0 or **SEEK_SET -** Beginning of file
1 or **SEEK_CUR -** Current position
2 or **SEEK_END -** End of file

Error Handling in Data file

- It is always possible that an error may occur during I/O operations in a file. Therefore, it is required to check whether required file is successfully opened or not.
- C provides two status inquiry macros feof() and ferror() that help us to detect I/O errors in file.

1.feof() - It can be used to test for an end of file condition. It takes file pointer as its argument . It returns non-zero integer when the end-of-file is reached otherwise returns zero.

Syntax: feof(file_pointer);

2. ferror() - It reports the status of the file indicated. It also takes file pointer as its argument and returns a non-zero integer if an error has detected during processing otherwise returns zero.

Syntax: ferror(file_pointer);

If file cannot be opened for some reasons, then function fopen returns a NULL pointer. So, NULL pointer can be used to check whether the file has been opened successfully or not.

Multiple Choice questions of File Handling

1.Which function is used to open a file in C?
a) read()
b) fopen()
c) fwrite()
d) fclose()

2.Which one of the following is correct syntax for opening a file.
a) FILE *fopen(const *filename, const char *mode)
b) FILE *fopen(const *filename)
c) FILE *open(const *filename, const char *mode)
d) FILE open(const*filename)

3.What is the function of the mode ‘ w+’?
a) create text file for writing, discard previous contents if any
b) create text file for update, discard previous contents if any
c) create text file for writing, do not discard previous contents if any
d) create text file for update, do not discard previous contents if any

4.If the mode includes b after the initial letter, what does it indicates?
a) text file
b) big text file
c) binary file
d) blueprint text

5.Choose the right statement for fscanf() and scanf()

a) fscanf() can read from standard input whereas scanf() specifies a stream from which to read
b) fscanf() can specifies a stream from which to read whereas scanf() can read only from standard input
c) fscanf() and scanf() has no difference in their functions
d) fscanf() and scanf() can read from specified stream

6.what is the function of fputs()?

a) read a line from a file
b) read a character from a file
c) write a character to a file
d) write a line to a file

7.Which function will return the current file position for stream?

a) fgetpos()
b) fseek()
c) ftell()
d) fsetpos()

8.What does the following C code snippet mean?
char *gets(char *s)

a) reads the next input line into the array s
b) writes the line into the array s
c) reads the next input character into the array s
d) write a character into the array

9.The______function reads atmost one less than the number of characters specified by size from the given stream and it is stored in the string str.

a) fget()
b) fgets()
c) fput()
d) fputs()

10.Select the right explanation for the following C code snippet.

```
int fgetpos(FILE *stream, fpos_t *s)
```

a) records the current position in stream in *s
b) sets the file position for stream in *s
c) positions stream at the position recorded in *s
d) reads from stream into the array ptr

11.Which function will return the current file position for stream?

a) fgetpos()
b) fseek()
c) ftell()
d) fsetpos()

12.What does the following C code snippet mean?

```
char *gets(char *s)
```

a) reads the next input line into the array s
b) writes the line into the array s
c) reads the next input character into the array s
d) write a character into the array

13.Which of the following is used to open a file in C for writing data?

a) “w”
b) “r”
c) “a”
d) “x”

15.Which of the following is used to create a binary file in C?

a) “w” mode
b) “a” mode
c) “wb” mode
d) “ab” mode

14.What is the purpose of the “fseek” function in C programming?

a) It sets the file pointer to a specific position in the file
b) It reads a line from a file
c) It writes a line to a file
d) It tests for end-of-file on a file stream

16.Which of the following is true about sequential access to a file in C?

a) It allows us to read or write data at any position within the file
b) It allows us to read or write data only from the beginning of the file
c) It allows us to read or write data only from the end of the file
d) It does not allow us to read or write data from a file

17.Which of the following is an example of a file mode used for opening a file for random access in C?

a) “r”

b) “w”

c) “a”

d) “r+”

18.Which of the following function is used to write data to a file in C?

a) fputc()

b) fputs()

c) fwrite()

d) fprintf()

19.Which of the following is true about random access to a file in C?

a) It allows us to read or write data only from the beginning of the file

b) It allows us to read or write data only from the end of the file

c) It allows us to read or write data at any position within the file

d) It does not allow us to read or write data from the file

20.Which function is used to check if the end of a file has been reached in C?

a) feof()

b) fseek()

c) fgetpos()

d) ftell()