UNIT-IV

POINTERS AND STRUCTURES.

* POINTERS:-

* Pointer:- A pointer is a variable that stores the address of another variable of the same data type.

* Declaration:

datatype *Pointer name;

ex: int *p;

* Initialisation.

datatype variable name;
datatype *Pointer name;
Pointer name = & variable name

int a;
int *p;
p = &a;

NOTE: Initialisation and declaration can be done at same time which is as follow.

datatype variable name;
datatype *ptrname = &variable;

Ex: int a;
int *p = &a;

* Advantages:-

1) Pointers save memory space.
2) Execution time with pointers is faster because data are manipulated with the address that is, direct access to memory location.
3) Memory is accessed efficiently with the pointers. The pointer assigns and releases the memory as well. Hence it can be said the memory of pointers is dynamically posted.

4. Pointers are used with data structures. They are useful for representing two dimensional and multi-dimensional arrays.
5. An array, of any type, can be accessed with the help of pointers, without considering its subscript range
6. Pointers are used for file handling.
7. Pointers are used to allocate memory dynamically

* Disadvantages:

1) If pointers are pointed to some incorrect location then it may end up reading a wrong value.
2) Errors in input always lead to erroneous output
3) Segmentation fault can occur due to unitialized pointer.
4) Pointers are slower than normal variable
5) It requires one additional dereferences step.
6) If we forgot to deallocate a memory then it will lead to a memory leak
7) There is always a chance of dangling pointers and wild pointers.

* ACCESSING of variables through pointers.

→ There are 2 ways
1) Direct access
2) Indirect access.

Direct access: We use directly the variable name to get its value.

Indirect access: We use a pointer to the variable and through an indirection operator access the variable.

* Write a C
through po

Program
#include
int main
{
int a
p = &a
scanf
printf
printf
print
print
return
}

* Size

size of
the data
because
location

→ size of
of floa
Pointer.

Program
#includ
int ma
{
int *
char

* Write a C program for accessing the variables through pointers.

Program:

```
#include<stdio.h>
int main()
{
    int a, *p;
    p = &a;
    scanf("%d", &a);
    printf("Direct Access a = %d", a);
    printf("\nIndirect Access a = %d", *p);
    printf("\nDirect Access address = %d", &a);
    printf("\nIndirect Access address = %d", p);
    return 0;
}
```

* Size of pointer variable:

Size of pointer of a variable is independent of the datatype of the variable it is pointing to because at the end it is pointing to a memory location.

, size of int pointer = size of char pointer = size of float pointer = size of any other type of Pointer.

Program:

```
#include<stdio.h>
int main()
{
    int *p1;
    char *p2;
    float *p3;
```

```
Printf("\n The size of integer pointer = %d", size of
                                        (int *));
Printf("\n The size of character pointer = %d",
                                        size of (p2));
Printf("\n The size of float pointer = %d", size of
                                        (p3));
Printf("\n The size of double pointer = %d",
                                        size of (double *));
return 0;
}
```

Output:

The size of integer pointer = 8
The size of character pointer = 8
The size of float pointer = 8
The " " double " = 8

Types of pointers:-

* There are 4 types of pointers, they are

1. Null Pointer
2. Void Pointer
3. Wild Pointer
4. Dangling pointer

*Null pointer:-

NULL pointer is a pointer which is pointing to nothing. In case, if we don't have address to be assigned to a pointer, then we can simply use NULL
If you assign a null

Write a C program to implement null pointer.

```
#include<stdio.h>
int main()
{
int *ptr = NULL;
Printf("The value of ptr is %p", ptr);
return 0;
}
```

Output:
The value of ptr is(nil)

* void pointer:
The void pointer in C is a pointer that is not associated with any datatypes. It points to some datalocation in this storage. This means that it points to the address of variables. It is also called the general purpose pointer or generic pointer. In C, malloc() and calloc() functions return void* or generic pointers.

* Write a C program to implement void pointer.

```
#include<stdlib.h>
int main()
{
int x=4;
float y=5.5;
void *ptr;
ptr = &x;
Printf("Integer variable is = %d", *(int*)ptr);
ptr=&y;
Printf("\n float variable is: %f", *(float*)ptr)
return 0;
}
```

Output:

Integer variable = 4

float variable x = 5.

* Dangling pointer:

A pointer pointing to a memory location that has been deleted (or freed) is called dangling pointer. There are three different ways where pointer acts as dangling pointer.

* constant pointer:

A constant pointer which points to a variable cannot be made to point to any another variable.

Ex: #include<stdio.h>

int main(void)

{

int var 1 = 0, var 2 = 0;

int *const ptr = &var 1;

ptr = &var 2;

Printf("%d\n", *ptr);

return 0;

}

Output:

Error

* Pointer to a constant:-

A pointer to constant is a pointer through which the value of variable that the pointer points cannot be changed. The address of these pointers can be changed, but the value of the variable

A pointer to constant is defined as:

const <type of pointer> * <name of pointer>

An example of definition could be:

```
const int *ptr;
```

Program:

```
#include<stdio.h>
int main(void)
{
 int var1=0;
const int *ptr=& var1;
  *ptr=1;
 Printf("%d\n", *ptr);
 return 0;
}
```

Output:

Error

* Pointer to pointer:-

It is also known as chain of pointers. It means It holds the address of another address.

* Write a C program to implement chain of Pointers:

```
#include<stdio.h>
 int main()
{
   int a;
   int *p, **q; ***r;
   p=&a;
   q=&p;
    r=&q;
   scanf("%d", &a);
   Printf("\n a=%d", a);
```

```
Printf("\n a= %d ", *p);
Printf("\n a= %d ", **q);
Printf("\n a= %d", ***r);
Printf("\n p= %d", p);
Printf("\n p= %d", *q);
Printf("\n p= %d", **r);
Printf("\n q= %d", q);
Printf("\n q= %d", *r);
Printf("\n r= %d", r);
return 0;
}
```

* Pointer Arithmetic :-

1. Pointer address arithmetic - Pointer arithmetic
2. Pointer value arithmetic - Pointer Expression

* Pointer address arithmetic :-

Valid Operations :-

The operation possible on addoess stored in pointers are -

1. A constant can be added to a pointer
2. A constant can be constructed from a pointer.
3. Two pointers of similar type can be subtracted
4. Pointer variables can be incremented and decremented
5. Pointer variables of same type can also be compared using relational operators.
6. Using assignment operator, pointers of similar type can be assigned.

Invalid Operations :-

1. Multiplication of pointers is not allowed.
2. Two pointers cannot be divided
3. Addition of pointers is also not valid.
4. The indirection operator cannot be applied to a void pointer
5. Multiplication of a pointer with a constant value
6. Division of a pointer with a constant value.

* Pointer expressions & value arithmetic :-

Like other variables pointer variables can be used in expressions.

If p_1 and p_2 are properly declared and initialized pointers, then the following statements are valid.

```
y = *p1 **p2;
sum = sum + *p1;
z = 5* - *p2/*p1;
*p2 = *p2 + 10;
*p1 = *p1 + *p2;
*p1 = *p2 - *p1;
```

* Pointers and Arrays :-

When an array is declared, the compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory locations. The base address is the location of first element (index 0) of the array. The compiler also defines the array name as a constant pointer to the first element.

1-D Array:-

Write a c program to read and display the contents of 1-D array using pointers.

```
#include <stdio.h>
int main()
{
    int a[20], n, i, *p;
    p = a;
    Printf("Enter n");
    scanf("%d", &n);
    Printf("\n Enter the elements");
    for(i=0; i<n; i++)
    scanf("%d", p+i);
    Printf("\n The elements are: ");
    for(i=0; i<n; i++)
    Printf("%d ", *(p+i));
    return 0;
}
```

* NOTE: To find the address of nth element in an array we have a formula

address of nth element	= Base address + n * size of (data type)

* 2-D array:-

scanf("%d", &a[i][j]);

Let x = a[i]

&a[i][j] = &x[j]

= x + j

= a[i] + j

= *(a+i) + j

a[i][j] = *(*(a+i) + j)

* write a C
using point
#include <
int main(
{
int a[10]
Printf("
scanf("
Printf("
for(i=
{
for(
scan
}
Printf
for(i=
{
for(
Prin
Prin
}
retur
}

* Pointers
A pointer
base addre
be derefere
get the valu
while loop

* Write a
using point
#include <
int main(
{

* Write a C program to implement 2-D array using pointers.

```
#include<stdio.h>
int main()
{
    int a[10][10], r, c, i, j, *p;
    Printf("Enter order of matrix");
    scanf("%d %d", &r, &c);
    Printf("\n Enter the elements");
    for(i=0; i<r; i++)
    {
        for(j=0; j<c; j++)
        scanf("%d", *(a+i)+j);
    }
    Printf("\n The etements are :\n");
    for(i=0; i<r; i++)
    {
        for(j=0; j<c; j++)
        Printf("%d ", *(*(a+i)+j));
        Printf("\n");
    }
    return 0;
}
```

* Pointers and strings:-

A pointer to a string in C can be used to point to the base address of the string array, and its value can be dereferenced to get the value of the string. To get the value of the string array is iterated using a while loop until a null character is encountered.

* Write a C program to read and display a string using pointers.

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

```
char s[20], *p;
p=s;
Printf("Enter a string ");
scanf("%s", p);
Printf("The string is : ");
Printf("%s", p);
return 0;
}.
```

* Array of pointers:

An array of pointers would be an array that hol memory locations. such a construction is often necessary in the c programming language. In computer programming, an array of pointers is an indeed set of variables in which the variables are Pointers (a reference to a location in memory).

Ptr &

* Arrays & functions:-

functions & pointers can be accessed in 4 ways.

1) Passing pointer as an argument to a function
2) function returning a pointer
3) Returning multiple values through functions. using pointer
4) Pointer to a function or function pointer..

* Passing pointer as an argument to a function:

Pointer as a function parameter is used to hold addresses of arguments passed during function call. This is also known as call by reference. when a function is called by reference any change made to the reference variable will effect the original variable. The actual arguments are addresses of variable. The formal arguments are pointers of specific datatype like int, char, float, structure, file etc.

```
#include <stdio.h>
void swap(int *, int *);
int main()
{
  int a, b;
  scanf("%d %d", &a, &b);
  printf("\n Before Swapping a=%d, b=%d", a, b);
  swap(&a, &b);
  printf("\n After swapping a=%d, b=%d", a, b);
}
void swap(int *x, int *y);
{
  int t;
  t = *x;
  *x = *y;
  *y = t;
}
```

* function returning a pointer:-

Write a C program to implement function returning a pointer:

```
#include<stdio.h>
int *large(int *, int *);
int main()
{
  int a, b, *p;
  scanf("%d %d", &a, &b);
  p = large(&a, &b);
  printf("The largest number = %d", *p);
  return 0;
}
int *large(int *x, int *y)
{
  if (*x > *y)
  return x;
```

```
else
return y;
}
```

* Returning multiple values through functions using pointers:-

```
#include<stdio.h>
void compute(int *, int *, int *, int *);
int main()
{
 a, b, sum, diff, prod;
 scanf("%d %d", &a, &b);
 compute(&a, &b, &sum, &diff, &prod);
 Printf("\n a = %d \n b = %d \n Sum = %d \n Diff = %d \n
        Product = %d", a, b, sum, dif, Prod);
}
void compute(int *x, int *y, int *s, int *d, int *p)
{
    *s = *x + *y;
    *d = *x - *y;
    *p = *x * *y;
}
```

* Pointer to a function (or) function pointer:-

Decla A pointer to a function is declared as follows

Return type(*Pointer-name)(Parameter);

Ex:

int (*sum)(); → legal declaration

int *sum(); → not a declaration of pointer to function

* Write a C

```
#include<s
int sum(int
int main()
{
 int res;
 int (*fpt
 fptr = sum
 res = fpt
 printf("
 return 0;
}
int sum(in
{
 return x
}
```

* Write a C program to implement function pointer

```
#include <stdio.h>
int sum(int x, int y);
int main()
{
  int res;
  int (*fptr)(int, int);
  fptr = sum;
  res = fptr(10,15);
  printf("\n Sum is %d", res);
  return 0;
}
int sum(int x, int y)
{
  return x+y;
}
```

Structures:-

⇒ A structure is user defined data type in C. A structure creates a datatype that can be used to group items of possibly different types into a single type/entity. Structures are a way to group several related variables into one place. Each variable in the structure is known as a member of the structure. The keyword that creates structures is struct.

⇒ Types of Structures:-

1) Tagged structure
2) Typedef structure

* Declaration of a structure:-

⇒ Create the beginning and ending statements for the structure using keyword "struct".
→ You can specify the name to it.
⇒ Add elements to the body of the structure
→ A structure must have atleast one element.

* Tagged Structure:-

- The structure definition associated with the structure name is referred as tagged structure.
- It doesn't create an instance of a structure and does not allocate any memory
- Tagname is further used to access different structures defined in a program
- Since each structure has different attributes, tag name helps to identify them uniquely.
- The tag name is also optional in C.
- If tag name is not given it restricts variable creation to only along with template.

Syntax of

struct TAG
{
Data Type va
Data Type
.....
members o
.....
Data type
};

Example:-

struct st
{
int roll
char nam
int mark
};

* Typedef

⇒ The stru
keyword
→ This is
Structure
⇒ Typedef
implement
→ They also
function
→ They ex

Syntax of a Tagged Structure:

```
struct TAGNAME // Tagname is the name of the structure.
{          // struct is keyword
DataType variable 1;
Data Type variable 2;
.....  // variable 1, variable 2.... are called
members of the structure.
.....
Data type variable-n;
};
```

Example:-

```
struct student
{
int roll no;
char name [20];
int marks [6];
};
```

* Typedef Structure:-

⇒ The structure definition associated with the keyword typedef is called type-defined structure

→ This is the most powerful way of defining the structure

⇒ Typedefined structures are very handy in implementing Data structures in C.

⇒ They also become useful in passing structures to functions.

⇒ They enhance readability.

Syntax:

```
typedef struct
{
Type variable 1;
Type variable 2;
.....
Type variable n;
} Type;
```

Example:

```
typedef struct
{
  int rollno;
  char name[20];
  int marks[6];
} student;
```

→ members / fields

*~~NOTE:- members/fields~~

* Declaration and initialisation of structure variables:-

The declaration and initialisation will be done in 3 ways

(i) Initialization along with structure template or definition

(ii) Initialization inside main during variable creation.

(iii) Initialization outside main " " "

(i) Initialization along with structure template or defn:-

Tagged structure:-

```
struct student
{
int rollno;
```

Typedef structure

```
typedef struct
{
int rollno;
char name[20];
```

```
char name[2
float mark
};
```

* Initializ

Tagged

```
struct stu
{
int rollno;
char name[2
float mark
} s={10,"ab
```

(iii) Initialize

Tagged

```
Struct stude
{
int rollno;
char name[
float mark
};
Struct studen
```

Initializatio

Tagged

```
Struct stude
{
int rollno;
char name[
float mark
};
Struct stude
```

```
char name[20];
float marks;
};
```

```
float marks
} student s;
```

* Initialization :-

Tagged

```
struct student
{
 int rollno;
 char name[20];
 float marks;
} s={10,"abc",90};
```

Typedef

```
typedef struct
{
 int rollno;
 char name[20];
 float marks;
}student s={10,"abc",90};
```

(iii) Initialization outside main during variable creation

Tagged

```
struct student
{
 int rollno;
 char name[20];
 float marks;
};
struct student s;
```

Initialization :-

Tagged

```
struct student
{
 int rollno;
 char name[20];
 float marks;
};
struct student s; ={10,'abc',90};
```

* Initialization inside main during variable

Initialization
Declaration

Tagged

```
struct student
{
  int rollno;
  char name[20];
  float marks;
};
void main()
{
float
struct student s;
```

Initialization.

```
struct student
{
  int rollno;
  char name[20];
  float marks;
};
void main()
{
struct student s;={10, "abc", 90};
}
```

*Accessing the member of structures:-

⇒ The members of a structures can be accessed using dot/member/ Period operator (.)

Syntax:

structure variable . structure member.

Ex: S. rollno.

* Write a C program to read and display the elements of a structure.

```
#include<stdio.h>
Struct student
{
  int rollno;
  char name[30];
  float marks
};
void main()
{
  struct student s;
  Printf("Enter the details");
  scanf("%d%s%f", &s.rb, s.name, &s.marks);
  Printf("\n The details are:");
  Printf("\n Rollno = %d", &s.rno);
  Printf("\n Name = %s, s.name);
  Printf("\n Marks = %f", s.marks);
}
```

* Operations on structures:

There are 2 possible operations

1) Operations on structures Data members
2) Operations on structure variables.

* Operations on Structure Datamembers:-

```
s1.rollno = s1.rollno - 2;        → valid
strcpy(s1.name, "Hello");         → valid
s1.marks = s1.marks/10;           → valid
s1.avg = s1.avg % 5;              → Invalid.
```

* Operations on Structure variables:

```
Struct student
{
  int rollno;
  char name[20];
} s1 = {73, "Sindhu"}, s2;
```

s2 = s1 is valid. but we can't compare two structure variable. i.e ⇒ s1 == s2 is not valid

* Arrays & Structures:-

⇒ There are 2 ways

1) Arrays within structures
2) Array of Structures

1) Arrays within structures:-

If the members of the structure are using arrays then it is called array within structures.

* Write a C program to implement arrays within Structures.

```
#include<stdio.h>
Struct student
{
   int rno;
   char name[30];
   float marks[3];
};
void main
{
 struct
 Printf("
 scanf("

 Print f
 Print
 Print
 Print
 Print
 Print
 Print
}
```

* Array of

If we wa
of struct

* write a

```
#includ
 Struct
 {
   int r
   char
   float
 };
 void m
{
 Struct
 int n,
 Scanf(
 for(i=
 {
```

```
};
void main()
{
 struct student s;
 Printf("Enter the details");
 scanf("%d%s%f%f%f", &s.rno, s.name, &s.marks[0],
                    &s.marks[1], &s.marks[2]);

 Printf("\n The details are:");
 Printf("\n Roll no = %d", s.rno);
 Printf("\n Name = %s", s.name);
 Printf("\n Marks 1 = %f", s.marks[0]);
 Printf("\n Marks 2 = %f", s.marks[1]);
 Printf("\n Marks 3 = %f", s.marks[2]);
}
```

(*) Array of Structures:-

If we want ~~to~~ maintain the details of n number of structures then we will go for array of structures.

*write a C program to implement array of structures.

```
#include<stdio.h>
 struct student
 {
   int rno;
   char name[30];
   float marks;
 };
 void main()
{
 struct student s[10];
 int n,i;
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
Printf("Enter the details of student %d", i+1);
```

```
Scanf("%d%s%f", &s[i].rno, s[i].name, &s[i].marks);
}
Printf("\n The details are: ");
Printf("\n Rollno\t Name\t Marks");
for(i=0; i<n; i++)
Printf("\n%d\t%s\t%f", s[i].rno, s[i].name, s[i].marks);
}.
```

* size of a structure:

Size of a structure is sum of the sizes of each members.

Ex: ~~#include<st~~ size struct student

```
{
  int rno;
  char name[30];
  float marks;
};
```

size of (struct student) = size of (s) = 4+30+4 = 38

* Nested Structures :-

Defining a structure within another structure is known as nested structures.

There are 2 methods.

1) Defining template and variable of one structure inside another structure.
2) Creating variable of one structure inside another structure.

* Defining template and variable of one structure inside another structure.

Syntax:

```
Struct outerstructure
```

```
{ data type
  data type
  ---------
  ---------
struct Inn
{ data type
  data type
  ---------
} Inner va
  ---------
  ---------
} <outer va
```

Ex: struct

```
{
  int
  char
  stru
  {
    i
    i
    i
  } x
```

* creating
another str

Syntax:

```
struct
{
  datatype
  datatype
  ---
```

```
{
datatype mem1;
datatype mem2;
---------
---------
struct Innerstructure
{
  datatype dm1;
  datatype dm2;
  ---------
  ---------
}Inner variable;
---------
---------
}<outer variable>;
```

Ex:
```
struct student
  {
    int rno;
    char n[30];
    struct dob
    {
      int d;
      int m;
      int y;
    }x;
```

* creating variable of one structure inside another structure.

Syntax:-

```
struct Innerstructure
{
  datatype dm1;
  datatype dm2;
  ---------
  ---------
};
```

```
struct Outerstructure
{
    datatype mem1;
    datatype mem2;
    ----------
    ----------
    struct Innerstructure Innervariable;
    ----------
}<outer variable>;
```

Ex:-
```
struct stud
{
    int rno;
    char n[30];
    struct dob x;
};
struct dob
{
    int d;
    int m;
    int y;
};
```

* Structures and pointers:-

The members of a structure can be accessed using arrow operator (⟶)

* write a c program to implement the members of a structure using pointers.

```
#include<stdio.h>
struct stud
{
    int r.no;
```

```
char n[
float m
};
int main
{
    struc
    p = &S
    print
    scanf
    print
    print
    retur
}
```

* Struc

The stru
in 2 way
1) Struct
2) Fu

* Pas sin
function
There ar
1) Passing
2) Passing
3) Passing

* Passin

Ex: #incl
Struc
{
int
Ch

```
    char n[20];
    float marks;
};
int main()
{
    struct stud s, *p;
    p = &s;
    Printf(" Enter the details ");
    Scanf("%d %s %f", &p->rno, p->n, &p->marks);
    Printf("\n The details are : ");
    Printf("\n %d %s %f", p->r.no, p->n, p->marks);
    return 0;
}
```

(*) Structures and functions :-

The structures and functions can be accessed in 2 ways

1) Structure as a parameter to a function
2) Function returning a structure

* Passing structure as an argument to a function :

There are 3 ways

1) Passing individual elements
2) Passing whole structure
3) Passing address of a structure.

* Passing Individual elements.

Ex:
```
#include <stdio.h>
struct stud
{
    int rno;
    char n[20];
    float marks;
```

```
};
void display(int,char[],float);
void main()
{
    struct stud s;
    Printf("Enter the details");
    scanf("%d%s%f", &s.rno, s.n, &s.marks);
    display(s.rno, s.n, s.marks);
}
void display(int rno, char n[], float marks)
{
    Printf("\nRoll no = %d", rno);
    Printf("\nName = %s", n);
    Printf("\nMarks = %f", marks);
}
```

ii) Passing the whole structure :-

```
#include <stdio.h>
struct stud
{
  int rno;
  char n[20];
  float marks;
};
void display(struct stud);
void main()
{
  struct stud s;
  Printf("Enter the details");
  Scanf("%d%s%f", &s.rno, s.n, &s.marks);
  display(s);
}
```

```
void Display (struct stud s)
{
printf("\n Roll no = %d", s.rno);
printf("\n Name = %s", s.n);
printf("\n Marks = %f", s.marks);
}
```

* Passing the address of a structure.

```
#include <stdio.h>
struct stud
{
    int rno;
    char n[30];
    float marks;
};
void display (struct stud*);
void main()
{
    struct stud *s, *p;
    p = &s;
    printf("Enter the details");
    scanf("%d %s %f", &p->rno, p->n, &p->marks);
    display (&s);
}
void display (struct stud *s)
{
    printf("\n Roll no = %d", s->rno);
    printf("\n Name = %s", s->n);
    printf("\n Marks = %f", s->marks);
}
```

(2) Function returning a structure.

```
#include<stdio.h>
Struct book
{
  char author[20];
  Char title[30];
  int pag;
  float pri;
 };
Struct book update(struct book, int, float);
void main()
{
  struct book b={"balaguruswamy", "C programming",
                 200, 150}, ub;
  int p;
  float pr;
  Printf("\n Enter the pages to be added");
  Scanf("%d", &p);
  Printf("Enter the price to be increased");
  Scanf("%f", &pr);
  ub= update(b, p, pr);
  Printf(" The uptated details are:");
  Printf("\n Author = %s", ub.author);
  Printf("\n Title= %s", ub.title);
  Printf("\n Pages= %d", ub.pag);
  Printf("\n Price= %f", ub.pri);

}
struct book update(struct book b, int p, float pr)
{ b.pag= b.pag + p;
  b.pri= b.pri + pr;
  return b;
```

* Unions:-

A union is one of the User Defined data types. Union is a collection of variables referred under a single name. The syntax, declaration and use of union is similar to the structure but its functionality is different.

Syntax:

```
union union_name
{
 <data-type>element 1;
 <data-type>element. 2;
 ---------
} union-variable;
```

Example:

```
union techno
{
  int comp-id;
  char nm;
  float sal;
} tch;
```

* Unions can be defined in any of the following three ways.

Tagged

```
union u
{
 char c;
 int i;
 float f;
};
union ua;
```

variable

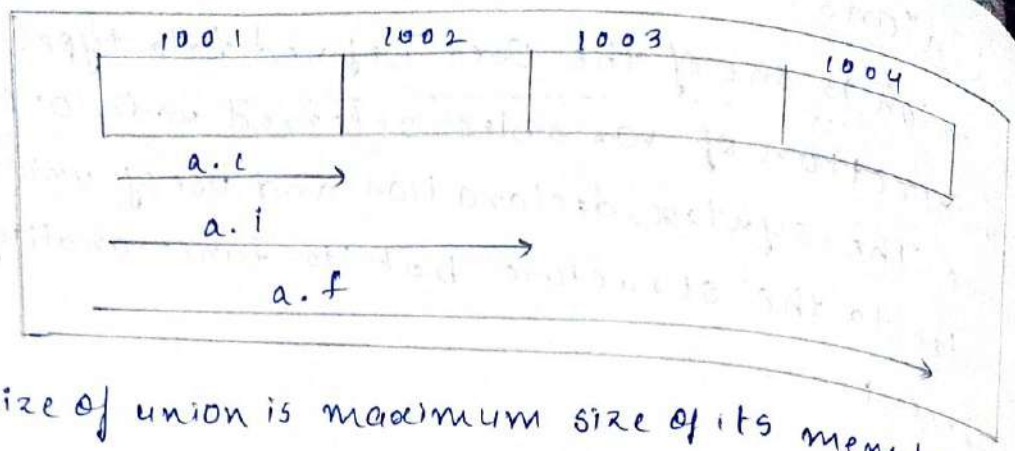
```
union u
{
 char c;
 int i;
 float f;
} a;
```

typedef :-

```
typedef union
{
 char c;
 int i;
 float f;
} u;
u a;
```

* memory occupancy of members are of a union are as follows

(a)



The size of union is maximum size of its members

(b) Differences b/w structure and union:-

Structure	Union
we use a struct key word to define a structure	we use a union keyword to define a union
Every member within structure is assigned a unique memory location	In union, a memory location is shared by all data members
It enables you to initialize several members at once	It enables you to initialize only the first member of union
changing the value of one data member will be not affect other data members in structure	changing the value of one data member will change the value of other data members in union
The total size of 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 type of are variable
It occupies for each and every member written in inner parameters	It occupies space for a member having the highest size written in inner parameters

You can retreive any member ata time	You can access one member at a time in the union

* Typedef:

It is used to rename built it and user defined data types.

Syntax: typedef oldname newname;

Ex: ① typedef unsigned int uni;

uni x.

② typedef struct student STUD;

STUD s;