Shri Vile Parle Kelavani Mandal's

SHRI BHAGUBHAI MAFATLAL POLYTECHNIC

Computer Engineering Department

# Pointers and Structure

**Course: Programming in C**

**Course Code: PRC238912**

**Name of Staff: Mr. Pratik H. Shah**

**SEMESTER : II**

**DIVISION : A**

## Course Outcome – 4

**Use primary, derived and user defined data types in application programs**

**Student will be able to**

- Use Pointer as derived data types in developing applications
- Implement Structure as user defined datatype

## Introduction to Pointers

1. Pointers are powerful features of C and (C++) programming that differentiates it from other popular programming languages like: Java and Python.
2. Pointers are used in C program to access the memory and manipulate the address.
3. **A pointer is a variable whose value is the address of another variable**, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address.

# Pointer Syntax

In C, there is a special variable that stores just the address of another variable. It is **called Pointer variable** or, simply, a pointer.

Declaration of Pointer

```
data_type* pointer_variable_name;
int* p;
```

Above statement defines, p as pointer variable of type int.

# Pointer Declaration

```
int* p;
```

Here, we have declared a pointer p of int type.
You can also declare pointers in these ways.

```
int *p1;
int * p2;
```

Let's take another example of declaring pointers.

```
int* p1, p2;
```

Here, we have declared a pointer p1 and a normal variable p2.

# Reference operator (&) and Dereference operator (*)

1. As discussed, **& is called reference operator.** It gives you the address of a variable.
2. Likewise, there is another operator that gets you the value from the address, it is called a **dereference operator (*).**

3. Note: The * sign when declaring a pointer is not a dereference operator. It is just a similar notation that creates a pointer.

# Memory Address

Before you get into the concept of pointers, let's first get familiar with address in C.

If you have a variable var in your program, &var will give you its address in the memory, where **&** is commonly called the **reference operator.**

You must have seen this notation while using scanf() function. It was used in the function to store the user inputted value in the address of var.

```
scanf("%d", &var);
```

```
/* Example to demonstrate use of reference operator in C programming. */
#include <stdio.h>
void main()
{
  int var = 5;
  printf("Value: %d\n", var);
  printf("Address: %u", &var);  //Notice, the ampersand(&) before var.
}
```

**Output**

```
Value: 5
Address: 2686778
```

## Assigning addresses to Pointers

```
int* pc, c;
c = 5;
pc = &c;
```

Here, 5 is assigned to the c variable. And, the address of c is assigned to the pc pointer.

## Get Value of Thing Pointed by Pointers

To get the value of the thing pointed by the pointers, we use the * operator.

```
int* pc, c;
c = 5;
pc = &c;
printf("%d", *pc);      // Output: 5
```

Here, the address of c is assigned to the pc pointer. To get the value stored in that address, we used *pc.

## Changing Value Pointed by Pointers

```
int* pc, c;
c = 5;
pc = &c;
c = 1;
printf("%d", c);            // Output: 1
printf("%d", *pc);      // Ouptut: 1
```

## Another Example

```
int* pc, c;
c = 5;
pc = &c;
*pc = 1;
printf("%d", *pc);      // Output: 1
printf("%d", c);        // Output: 1
```

We have assigned the address of c to the pc pointer.

Then, we changed *pc to 1 using *pc = 1;. Since pc and the address of c is the same, c will be equal to 1.

## Another Example

```
int* pc, c, d;
c = 5;
d = -15;

pc = &c;
printf("%d", *pc);          // Output: 5
pc = &d;
printf("%d", *pc);        // Ouptut: -15
```

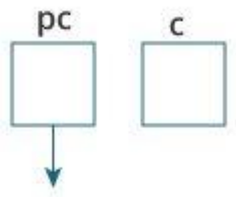
```
/* Source code to demonstrate, handling of pointers in C program */
#include <stdio.h>
void main()
{
  int* pc;
  int c;
  c=22;
  printf("Address of c:%u\n",&c);
  printf("Value of c:%d\n\n",c);
  pc=&c;
  printf("Address of pointer pc:%u\n",pc);
  printf("Content of pointer pc:%d\n\n",*pc);
  c=11;
  printf("Address of pointer pc:%u\n",pc);
  printf("Content of pointer pc:%d\n\n",*pc);
  *pc=2;
  printf("Address of c:%u\n",&c);
  printf("Value of c:%d\n\n",c);
}
```

## Output

```
Address of c: 2686784
Value of c: 22

Address of pointer pc: 2686784
Content of pointer pc: 22

Address of pointer pc: 2686784
Content of pointer pc: 11

Address of c: 2686784
Value of c: 2
```

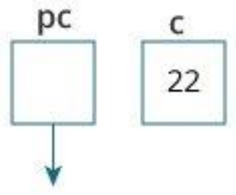
## Explanation of the program

1. **int* pc**, c;



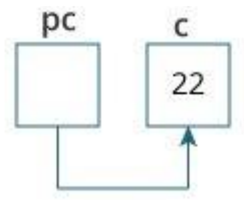
Here, a pointer pc and a normal variable c, both of type int, is created.
Since pc and c are not initialized at initially, pointer pc points to either no address or a random address. And, variable c has an address but contains random garbage value.

2. c = 22;



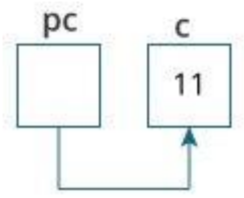
This assigns 22 to the variable c. That is, 22 is stored in the memory location of variable c.

3. pc = &c;



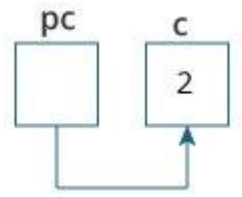
This assigns the address of variable c to the pointer pc.

4. c = 11;



This assigns 11 to variable c.

5. *pc = 2;



This change the value at the memory location pointed by the pointer pc to 2.

# Common mistakes when working with pointers

**Suppose, you want pointer pc to point to the address of c. Then,**
**int c, *pc;**

```
// pc is address but c is not
pc = c;                    // ERROR

// &c is address but *pc is not
*pc = &c;                  // ERROR

// both &c and pc are addresses
pc = &c;                   // NOT AN ERROR

// both c and *pc are values
*pc = c;                   // NOT AN ERROR
```

## Relationship Between Arrays and Pointers

An array is a block of sequential data. Let's write a program to print addresses of array elements.

```
#include <stdio.h>
void main()
{
      int x[4];
      int i;

      for(i = 0; i < 4; i++)
      {
            printf("&x[%d] = %p\n", i, &x[i]);
       }
       printf("Address of array x: %p", x);
}
```

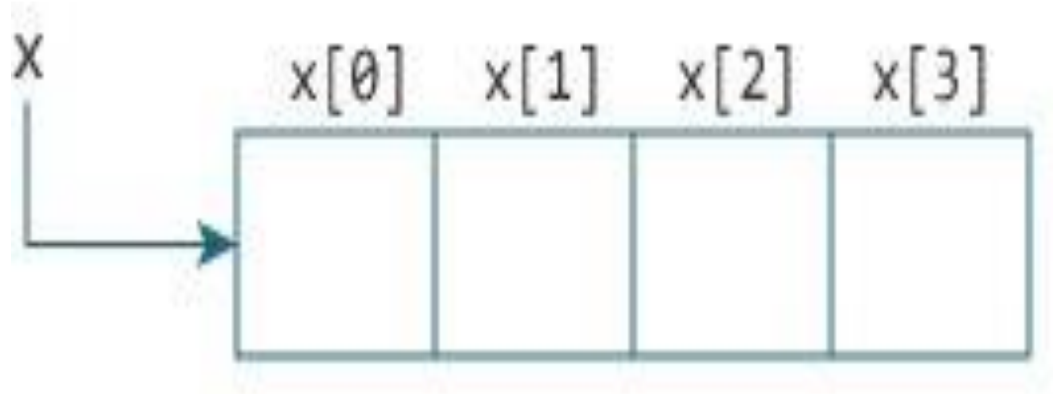
**OUTPUT:**

```
&x[0] = 1450734448
&x[1] = 1450734452
&x[2] = 1450734456
&x[3] = 1450734460
Address of array x: 1450734448
```

**%p is a format specifier which is used for displaying the address in hex format.**

There is a difference of **4 bytes** between two consecutive elements of **array x**. It is because the **size of int is 4 bytes** (on our compiler).

Notice that, the address of **&x[0]** and **x is the same**. It's because the variable name x points to the first element of the array.



## Pointers and Arrays

```
#include <stdio.h>
void main()
{
         int i, x[6], sum = 0;
         printf("Enter 6 numbers: ");
         for(i = 0; i < 6; ++i)
        {
            scanf("%d", x+i);                    // Equivalent to scanf("%d", &x[i]);
            sum += *(x+i);                      // Equivalent to sum += x[i]
        }
        printf("Sum = %d", sum);
}
```

## Arrays and Pointers

```
#include <stdio.h>
void main()
{
  int x[5] = {1, 2, 3, 4, 5};
  int* ptr;

  ptr = &x[2]; // ptr is assigned the address of the third element

  printf("*ptr = %d \n", *ptr);                  // 3
  printf("*(ptr+1) = %d \n", *(ptr+1));          // 4
  printf("*(ptr-1) = %d", *(ptr-1));             // 2
}
```

# Array of Pointer

```
#include <stdio.h>
const int MAX = 3;
void main ()
{
  int  var[] = {10, 100, 200};
  int i, *ptr[MAX];
  for ( i = 0; i < MAX; i++)
  {
     ptr[i] = &var[i];
  }
  for ( i = 0; i < MAX; i++)
  {
    printf("Value of var[%d] = %d\n", i, *ptr[i]);
  }
}
```

## Pointers and Functions

In C programming, it is also possible to **pass addresses as arguments** to functions.

To accept these addresses in the function definition, we can **use pointers**. It's because pointers are used to store addresses.

## Example 1 : Pass Addresses to Functions

```
#include <stdio.h>
void swap(int *n1, int *n2);

void main()
{
    int num1 = 5, num2 = 10;
     swap( &num1, &num2); // address of num1 and num2 is passed

    printf("num1 = %d\n", num1);
    printf("num2 = %d", num2);
}
void swap(int* n1, int* n2)
{
    int temp;
     temp = *n1;
     *n1 = *n2;
     *n2 = temp;
}
```

## Example 2: Passing Pointers to Functions

```
#include <stdio.h>
void addOne(int* ptr)
{
      (*ptr)++;           // adding 1 to *ptr
}
void main()
{
  int* p, i = 10;
  p = &i;
  addOne(p);

  printf("%d", *p);          // 11
}
```

## C Dynamic Memory Allocation

As you know, an array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it.

Sometimes the size of the array you declared may be insufficient. To solve this issue, you can allocate memory manually during run-time. This is known as **dynamic memory allocation in C programming.**

To allocate memory dynamically, library functions are **malloc(), calloc(), realloc() and free()** are used. These functions are defined in the **<stdlib.h> header file.**

| Function | Use of Function |
|---|---|
| malloc() | Allocates requested size of bytes and returns a pointer first byte of allocated space |
| calloc() | Allocates space for an array elements, initializes to zero and then returns a pointer to memory |
| free() | deallocate the previously allocated space |
| realloc() | Change the size of previously allocated space |

# C malloc()

1. The name malloc stands for "memory allocation".
2. The function malloc() reserves a block of memory of specified size and return a pointer of type void which can be casted into pointer of any form.

**Syntax of malloc()**

```
ptr = (cast-type*) malloc(byte-size)
```

1. Here, ptr is pointer of cast-type.
2. The malloc() function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.

```
ptr = (int*) malloc(100 * sizeof(int));
```

## C calloc()

The name calloc stands for "contiguous allocation".

The only difference between malloc() and calloc() is that, malloc() allocates single block of memory whereas calloc() allocates multiple blocks of memory each of same size and sets all bytes to zero.

**Syntax of calloc()**

**ptr = (cast-type*)calloc(n, element-size);**

This statement will allocate contiguous space in memory for an array of n elements.

**For example:**

**ptr = (float*) calloc(25, sizeof(float));**

This statement allocates contiguous space in memory for an array of 25 elements each of size of float, i.e, 4 bytes.

## C free()

Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on its own. You must explicitly use free() to release the space.

**syntax of free()**

```
    free(ptr);
```

This statement frees the space allocated in the memory pointed by ptr.

## Example #1: Using C malloc() and free()

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
   int num, i, *ptr, sum = 0;
   printf("Enter number of elements: ");
   scanf("%d", &num);
   ptr = (int*) malloc(num * sizeof(int));  //memory allocated using malloc
   if(ptr == NULL)
   {
      printf("Error! memory not allocated.");
      exit(0);
   }
   printf("Enter elements of array: ");
   for(i = 0; i < num; i++)
   {
      scanf("%d", ptr + i);
      sum += *(ptr + i);
   }
   printf("Sum = %d", sum);
   free(ptr);
}
```

Write a C program to find sum of n elements entered by user.
To perform this program, allocate memory dynamically using malloc() function.

## Example #2: Using C calloc() and free()

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
   int num, i, *ptr, sum = 0;
   printf("Enter number of elements: ");
   scanf("%d", &num);
   ptr = (int*) calloc(num, sizeof(int));
   if(ptr == NULL)
   {
      printf("Error! memory not allocated.");
      exit(0);
   }
   printf("Enter elements of array: ");
   for(i = 0; i < num; i++)
   {
      scanf("%d", ptr + i);
      sum += *(ptr + i);
   }
   printf("Sum = %d", sum);
   free(ptr);
}
```

Write a C program to find sum of n elements entered by user. To perform this program, allocate memory dynamically using calloc() function.

# C realloc()

If the previously allocated memory is insufficient or more than required, you can change the previously allocated memory size using realloc().

**Syntax of realloc()**

```
ptr = realloc(ptr, newsize);
```

Here, ptr is reallocated with size of newsize.

## Example #3: Using realloc()

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
   int *ptr, i , n1, n2;
   printf("Enter size of array: ");
   scanf("%d", &n1);

   ptr = (int*) malloc(n1 * sizeof(int));
   printf("Address of previously allocated memory: ");
   for(i = 0; i < n1; i++)
       printf("%u\t",ptr + i);

   printf("\nEnter new size of array: ");
   scanf("%d", &n2);

   ptr = realloc(ptr, n2);

   for(i = 0; i < n2; i++)
       printf("%u\t", ptr + i);
}
```

## Structure

## Introduction

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

**For example:**
You want to store some information about a person: his/her name, citizenship number and salary.

You can easily create different variables name, citNo, salary to store these information separately.

## Define Structures

```
struct structureName
{
    dataType member1;
    dataType member2;
    ...
};
```

## Example

```
struct Person
{
  char name[50];
  int citNo;
  float salary;
};
```

## Create struct Variables

1. When a struct type is declared, no storage or memory is allocated.
2. To allocate memory of a given structure type and work with it, we need to create variables.

### First Method

```
struct Person
{
     // code
};

void main()
{
  struct Person person1, person2, p[20];
  // remaining code
}
```

### Second Method

```
struct Person
{
    // code
} person1, person2, p[20];
```

## Access Members of a Structure

There are two types of operators used for accessing members of a structure.

**.  Member operator**

**->  Structure pointer operator**

Suppose, you want to access the salary of person2. Here's how you can do it.

**person2.salary**

## Example 1: C structs

```
#include <stdio.h>
#include <string.h>
struct Person
{
   char name[50];
   int citNo;
   float salary;
} person1;

void main()
{
      strcpy(person1.name, "George Orwell");          // assign value to name of person1
       person1.citNo = 1984;
       person1. salary = 2500;

     printf("Name: %s\n", person1.name);
     printf("Citizenship No.: %d\n", person1.citNo);
     printf("Salary: %.2f", person1.salary);
}
```

## Nested Structures

**The structure can be nested in the following ways.**

1. **By Separate structure**
2. **By Embedded structure**

### 1) Separate structure

Here, we create two structures, but the dependent structure should be used inside the main structure as a member.

### 2) Embedded structure

The embedded structure enables us to declare the structure inside the structure. Hence, it requires less line of codes but it can not be used in multiple data structures. Consider the following example.

## Example

1) Separate structure

```
struct Date
{
  int dd;
  int mm;
  int yyyy;
};
struct Employee
{
  int id;
  char name[20];
  struct Date doj;
}emp1;
```

2) Embedded structure

```
struct Employee
{
  int id;
  char name[20];
  struct Date
   {
     int dd;
     int mm;
     int yyyy;
   }doj;
}emp1;
```

## Nested Structures

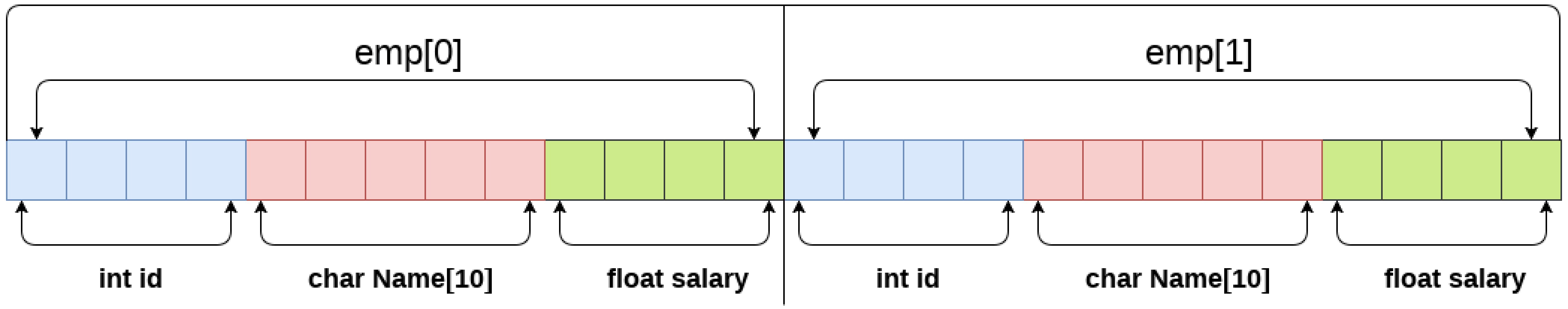
```
#include <stdio.h>
struct complex
{
       int imag;
       float real;
};
struct number
{
        struct complex comp;
        int integer;
} num1;
void main()
 {
     num1.comp.imag = 11;         // initialize complex variables
     num1.comp.real = 5.25;
     num1.integer = 6;                // initialize number variable

    printf("Imaginary Part: %d\n", num1.comp.imag);
    printf("Real Part: %.2f\n", num1.comp.real);
    printf("Integer: %d", num1.integer);
}
```

## Array of Structures in C

1. An array of structures in C can be defined as the collection of multiple structures variables where each variable contains information about different entities.

2. 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.

# Array of structures



```
struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

sizeof (emp) = 4 + 5 + 4 = 13 bytes

sizeof (emp[2]) = 26 bytes

## Example: Array of Structure

```
#include<stdio.h>
#include <string.h>
struct student
{
int rollno;
char name[10];
};
void main()
{
    int i;
    struct student st[5];
    for(i=0;i<5;i++)
    {
      printf("\nEnter Rollno:");
      scanf("%d",&st[i].rollno);
      printf("\nEnter Name:");
      scanf("%s",&st[i].name);
    }
    for(i=0;i<5;i++)
    {
      printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
    }
}
```

## Why structs in C?

1. Suppose you want to store information about a person: his/her name, citizenship number, and salary. You can create different variables name, citNo and salary to store this information.

2. What if you need to store information of more than one person? Now, you need to create different variables for each information per person: name1, citNo1, salary1, name2, citNo2, salary2, etc.

3. A better approach would be to have a collection of all related information under a single name Person structure and use it for every person.

## Example: Access members using Pointer

```
#include <stdio.h>
struct person
{
   int age;
   float weight;
};
void main()
{
    struct person *personPtr, person1;
    personPtr = &person1;

    printf("Enter age: ");
    scanf("%d", &personPtr->age);
    printf("Enter weight: ");
    scanf("%f", &personPtr->weight);

    printf("Age: %d\n", personPtr->age);
    printf("weight: %f", personPtr->weight);
}
```

## Passing structs to functions

```
#include <stdio.h>
struct student
{
   int age;
};
void display(struct student s);
void main()
{
   struct student s1;
   printf("Enter age: ");
   scanf("%d", &s1.age);
   display(s1); // passing struct as an argument
}
void display(struct student s)
 {
     printf("\nAge: %d", s.age);
}
```

## C Unions

A union is a user-defined type similar to structs in C except for one key difference.

**Structures allocate enough space to store all their members**, whereas **unions can only hold one member value at a time**.

## How to define a union?

```
union car
{
  char name[50];
  int price;
};
```

## Create union variables

```
union car
{
  char name[50];
  int price;
};

void main()
{
  union car car1, car2, *car3;
}
```

## Example: Accessing Union Members

```
#include <stdio.h>
union Job
{
   float salary;
   int workerNo;
} j;

void main()
{
   j.salary = 12.3;           // when j.workerNo is assigned a value,
   j.workerNo = 100;     // j.salary will no longer hold 12.3

   printf("Salary = %.1f\n", j.salary);
   printf("Number of workers = %d", j.workerNo);
}
```

**Output**

```
Salary = 0.0
Number of workers = 100
```

## Difference between unions and structures

```
#include <stdio.h>
union unionJob
{
   char name[32];
   float salary;
   int workerNo;
} uJob;

struct structJob
{
   char name[32];
   float salary;
   int workerNo;
} sJob;

void main()
{
   printf("size of union = %d bytes", sizeof(uJob));
   printf("\nsize of structure = %d bytes", sizeof(sJob));
}
```

**Output**

```
size of union = 32
size of structure = 40
```

## Why this difference in the size of union and structure variables?

**Here, the size of sJob is 40 bytes** because

1. the size of name[32] is 32 bytes
2. the size of salary is 4 bytes
3. the size of workerNo is 4 bytes

**However, the size of uJob is 32 bytes.**

1. It's because the size of a union variable will always be the size of its largest element.
2. the size of its largest element, (name[32]), is 32 bytes.
3. With a union, all members share the same memory.

# Exercise

1. Write a program in C to find the maximum number between two numbers using a pointer.
2. Write a program in C swap elements using call by reference.
3. Write a program in C to sort an array using a pointer.
4. Write a program in C to find the factorial of a given number using pointers.
5. Programs given in the presentation
6. Write a program in C to Add two distances (in inch-feet).
7. Write a program in C to Store information of a student using structure
8. Write a program in C to Store information of 10 Employees using structures
9. Programs given in the presentation

# Theory Questions

1. Define pointer with example
2. Define the following terms: a) Reference operator  b)Dereference operator
3. Explain Arrays of pointer with example program
4. Explain pointer to function with example program
5. State the use of following dynamic memory allocation function:
   a) malloc()   b)calloc()    c)free()    d)realloc()
6. Define Structure and union
7. Write the syntax of Structure declaration and Union Declaration
8. Explain nested structure with example program
9. Explain Arrays of Structure with example program
10. Explain structure to function with example program
11. Differentiate between Structure and Union

THANK YOU !!!!