Additional Notes for ADS Lab

Pointers

- The pointer in C language is a variable which stores the address of another variable.
- This variable can be of type int, char, array, function, or any other pointer.
- A **pointer** is a variable that stores the memory address of another variable as its value.
- A **pointer variable points to a data type** (like int) of the same type, and is created with the * operator.
- The address of the variable we are working with is assigned to the pointer:

Address in C

- If var is a variable in the program, &var will give its address in the memory.
- We have used address numerous times while using the scanf() function.
- scanf("%d", &var);
- Here, the value entered by the user is stored in the address of var variable.

```
#include <stdio.h>
 int main()
 {
int var = 5;
printf("var: %d\n", var);
printf("address of var: %p", &var);
 return 0;
}
```

Output

var: 5
address of var: 2686778

- **Assigning addresses to Pointers**

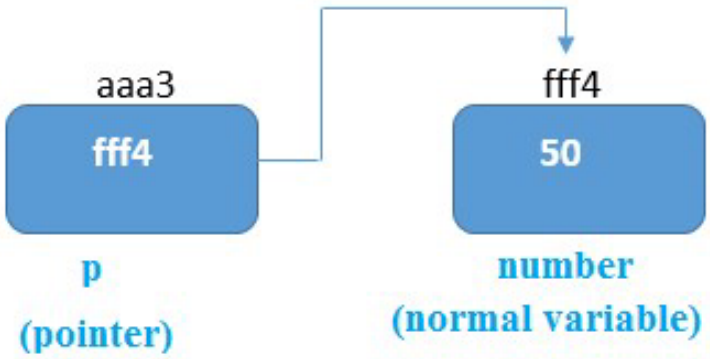
```
int* pc, c;

c = 5;

pc = &c;
```

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

- **Syntax:**
  - datatype *var_name;
- Declaring a pointer
  - The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.
  - **int** *a;//pointer to int
  - **char** *c;//pointer to char

Pointer Example

An example of using pointers to print the address and value is given below.




```
#include<stdio.h>
int main()
{
int number=50;
int *p;
p=&number;
printf("Address of number variable is: %x \n",p);
printf("Value of p variable is: %d \n",*p);
return 0;
}
```

Output

Address of number variable is: fff4

Value of p variable is: 50

Example: Working of Pointers

.

```
#include <stdio.h>
 int main() {
 int* pc, c; c = 22;
printf("Address of c: %p\n", &c);
printf("Value of c: %d\n\n", c); // 22
pc = &c;
printf("Address of pointer pc: %p\n", pc);
 printf("Content of pointer pc: %d\n\n", *pc); // 22
 c = 11;
printf("Address of pointer pc: %p\n", pc);
 printf("Content of pointer pc: %d\n\n", *pc); // 11
*pc = 2; printf("Address of c: %p\n", &c);
printf("Value of c: %d\n\n", c); // 2
 return 0;
}
```

Output

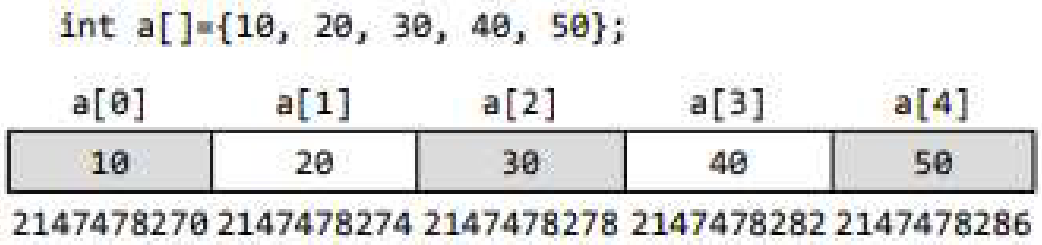
```
Address of c: 2686784
 Value of c: 22
Address of pointer pc: 2686784
Content of pointer pc: 22
Address of pointer pc: 2686784
Content of pointer pc: 11
 Address of c: 2686784
Value of c: 2
```

Arrays and Pointers

- Array notation is a form of pointer notation.
- The name of an array is the beginning address of the array, called the base address of the array.
- That is, the base address of an array is the address of the zeroth element of the array.
- The array name is referred to as an address constant.
- Mentioning the name of the array fetches its base address.

➢ Example 1

```
#include<stdio.h>
 int main()
 {
 int a[ ]={10, 20, 30, 40, 50};
printf("%u %u %u", a, &a[0], &a );
return 0; }
```



Output:

2147478270
2147478270
2147478270

➢ Example 2

```
#include<stdio.h>
int main()
{ int a[ ]={10, 20, 30, 40, 50};
 int i ;
printf("Using index method:");
for(i=0;i<5;i++)
{
        printf("%d\t", a[i]);\
}
printf("\nUsing  pointer method:”);
for(i=0;i<5;i++)
{
        printf("%d\t", *(a+i));
}
return 0;
}
```

Output:

Using index method: 10 20 30 40 50

Using pointer method: 10 20 30 40 50

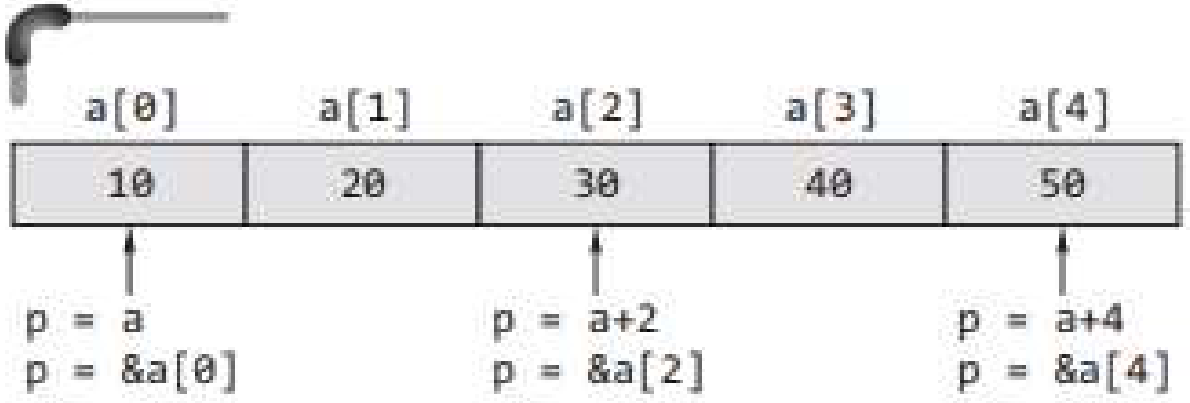
- **The integer identifier i is added to the base address of the array.**
- **The C compiler computes the resulting address that will be accessed by taking the value held in i multiplied by the size in bytes of the type of array a and adds the proper offset to a to give the correct memory address.**
- **Subscript notation is converted by the compiler into the pointer notation.**

- **A pointer variable (of the appropriate type) can also be used to initialize or point to the first element of the array.**

```
# include<stdio.h>
 int main()
 {
 int a[]={10, 20, 30, 40, 50};
 int i, *p;
p=a; /* it can also be written as p=&a[0]; */
 printf("\nUsing  pointer method:");
 for(i=0;i<5;i++)
 {
                printf("%d\t", p[i]);
 }
return 0;
}
```

Output:

Using pointer method: 10 20 30 40 50



The operations between array and pointers are as follows.

- The first such operation is that it is possible to (apparently) assign an array to a pointer.

```
int a[10];
int *p;
p = a;
```

- C defines the result of this assignment to be that p receives a pointer to the first element of a.
- In other words, p = &a[0];
- The second aspect of the equivalence is that the array subscripting notation [i] can be applied on pointers, too.
- p[3] can be written as *(p + 3).

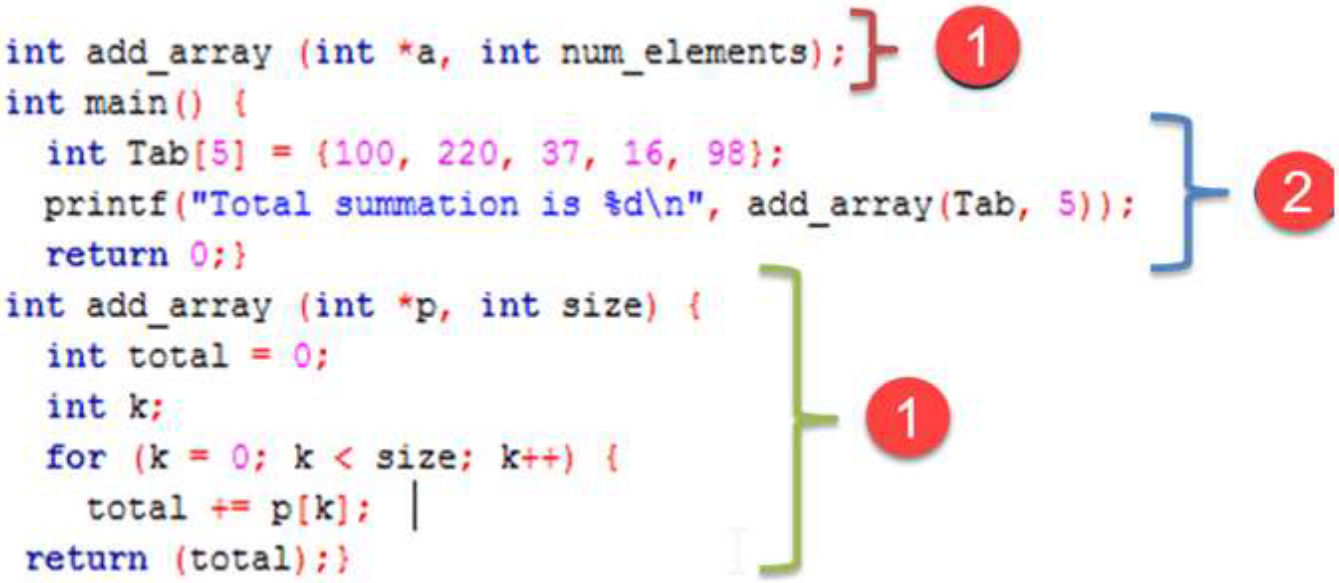
Passing an Array to a Function

- When an array is passed to a function, it degenerates to a pointer.
- All array names that are function parameters are always converted into pointers by the compiler.
- Because when passing an array to a function, the address of the zero-th element of the array is copied to the pointer variable which is the formal parameter of the function.
- However, arrays and pointers are processed differently by the compiler, represented differently at runtime.

Example program

```
#include <stdio.h>
int add_array (int *a, int num_elements);
Void addtwo(int *, int);
int main()
 { int sum;
  int Tab[5] = {100, 220, 37, 16, 98};
 sum=add_array(Tab, 5);
printf("Total summation is %d\n", sum);
addtwo(Tab,5);

 return 0;
}

int add_array (int *p, int size)
 {
int total = 0;
  int k;
 for (k = 0; k < size; k++)
{   total += p[k];  /* it is equivalent to total +=*p ;p++; */}
 return (total);
}
```


```
void addtwo(int *a, int s)
{
   int k;
 for (k = 0; k < size; k++)
  {
```



1. We declare and define add_array() function which takes an array address(pointer) with its elements number as parameters and returns the total accumulated summation of these elements. The pointer is used to iterate the array elements (using the p[k] notation), and we accumulate the summation in a local variable which will be returned after iterating the entire element array.

2. We declare and initialize an integer array with five integer elements. We print the total summation by passing the array name (which acts as address) and array size to the **add_array()**called function as arguments.

```
#include <stdio.h>
 #define MAX 50
 int main() {
 int arr[MAX],s;
void getdata(int *, int);
void show(int *, int);
printf("Enter the array size: ");
 scanf("%d", &s);
getdata(arr, s);
show(arr, s);
 return 0; }

/* Function reads scores in an array. */
void getdata(int *a, int n) {
 int x, i = 0;
 printf("\n Enter the array elements one by one\n");
 while(i < n)  {
 scanf("%d", &x);
 *(a + i) = x;
  i++;
 }
 }
void show(int *a, int n) {
 int i; for(i=0;i<n;i++)
printf("\n %d", *(a+i));
 }
```

Structure

- A structure is a collection of variables under a single name.
- These variables can be of different types, and each has a name which is used to select it from the structure.
- Therefore, a structure is a convenient way of grouping together several pieces of related information.

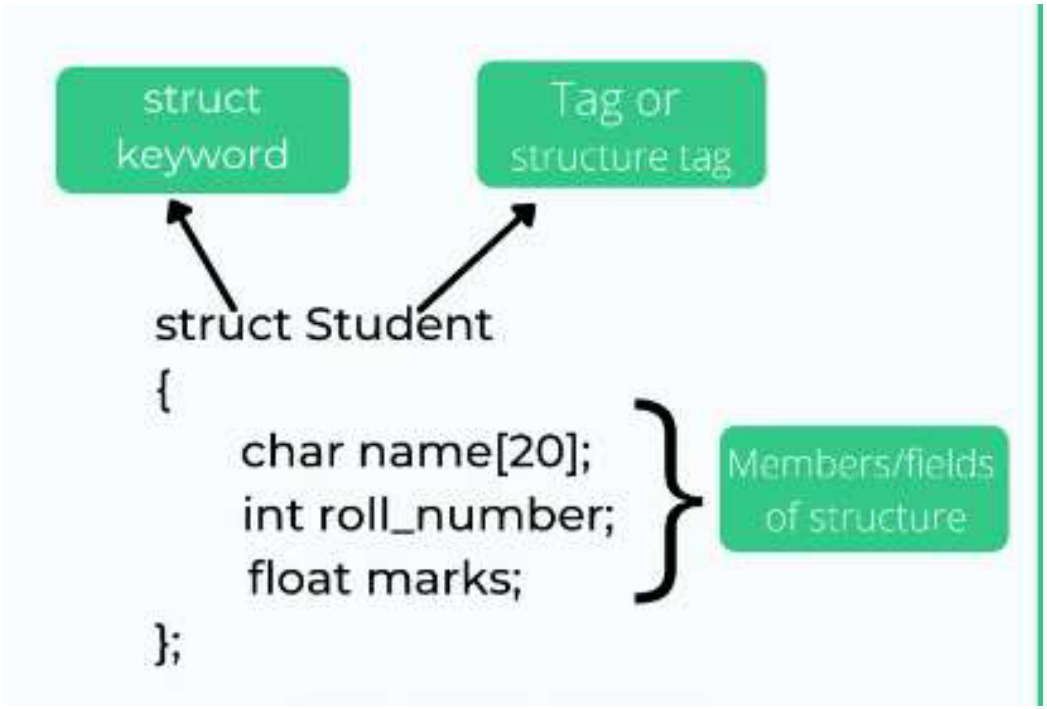
- A structure is declared by using the keyword **struct** followed by an optional structure tag followed by the body of the structure.
- The variables or members of the structure are declared within the body.

Syntax

keyword ⟹

```
struct structure_name
{
    data_type variable 1;
    data_type variable 2;
    ------------------------
    data_type variable n;

};
```

body



- Example 2

```
struct MyStructure // Structure declaration
{
   int myNum;         // Member (int variable)
   char myLetter;     // Member (char variable)
}; // End the structure with a semicolon
```

1.Here MyStructure is the name of structure

2.struct is a keyword.

- **There are 2 methods are available for access the structure,**

Method 1 - use the struct keyword inside the main() method, followed by the structure tag and then the name of the structure variable:

```
struct student
{
    int rollno;
    float marks;
};
 int main()
{
    struct student student1;
     return 0;
 }
```

Method 2

```
struct student
{
    int rollno;
    float marks;
}student1;
int main()
{
    return 0;
}
```

- Accessing the data members of structure
  - The data member of structure can be accessed as

    structure_variable.data_member
  - For example if we want to access the rollno of student then we can write as

    student1.rollno

Example: Write a program to store and display student information

Method 1

```
#include<stdio.h>
struct student
{
int rollno;
 float marks;
 };
int main()
{
struct student student1;//Declaring structure variable
student1.rollno=201;
student1.marks=85.9;
printf("Student Rollno=%d\n",student1.rollno);
printf("Student Marks=%f\n",student1.marks);
}
```

Method 2

```
#include<stdio.h>
struct student
 {
   int rollno;
   float marks;
 }s1;
 int main()
 {
   s1.rollno=201;
   s1.marks=85.9;
   printf("Student Rollno=%d\n",s1.rollno);
   printf("Student Marks=%f\n",s1.marks);
   return 0;
 }
```