

UNIT 9

POINTERS

LH - 6HRS

PREPARED BY: **ER. SHARAT MAHARJAN**

C PROGRAMMING

PRIME COLLEGE, NAYABAZAAR

CONTENTS (LH - 6HRS)

9.1 Introduction

9.2 Declaration and Initialization of Pointer Variables

9.3 Array of Pointers

9.4 Passing Pointers to Functions

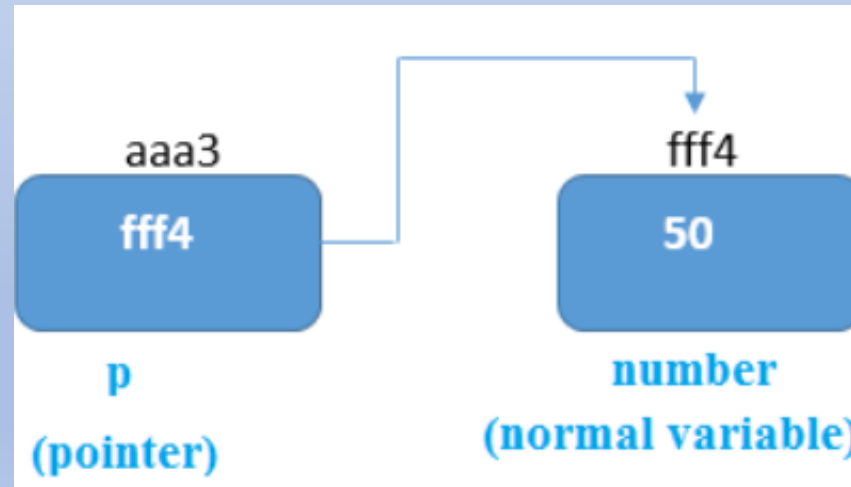
9.5 Pointers and Arrays

9.6 String and Pointer

9.7 Dynamic Memory Allocation

9.1 Introduction

- 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, one must declare a pointer before using it to store any variable address.



9.2 Declaration and Initialization of Pointer Variables

- The general form of a pointer variable declaration is –

type *var-name;

- Here, **type** is the pointer's **base type**; it must be a valid **C data type** and **var-name** is the **name of the pointer variable**. The **asterisk *** used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declarations –

int *ip; /* pointer to an integer */

double *dp; /* pointer to a double */

float *fp; /* pointer to a float */

char *ch; /* pointer to a character */

LAB 1: WAP for illustrating use of pointers.

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

```
int main(){
```

```
    int x=10, *p=&x;
```

```
    printf("The value of x=%d",x);
```

```
    printf("\nThe address of x=%u",p);
```

```
    printf("\nThe value of x=%d",*p);/* is indirection or dereference  
operator
```

```
    return 0;
```

```
}
```

```
The value of x=10  
The address of x=6684180  
The value of x=10
```

9.3 Array of Pointers

- A pointer variable always contains an address of a variable. So, **an array of pointers is actually an array of memory addresses of different variables.**

Syntax: data_type *pointer_name[size];

Example: int *p[5];

LAB 2: WAP for illustrating use of array of pointers.

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

```
int main(){
```

```
    int a[5]={10,20,30,40,50};
```

```
    int *p[5];
```

```
    for(int i=0;i<5;i++){
```

```
        p[i]=&a[i]; //store address of array element
```

```
        printf("Address=%u\n",p[i]);
```

```
    }
```

```
    return 0;
```

```
}
```

```
Address=6684160
Address=6684164
Address=6684168
Address=6684172
Address=6684176
```

9.4 Passing Pointers to Functions

- A pointer can be passed to a function as an argument.
- **Passing a pointer means passing address of a variable** instead of value of the variable.
- As address is passed in this case, **this mechanism is also called call by address or call by reference. (already done in previous unit)**
- Example for this topic: Call by reference/address done in user-defined function.

9.5 Pointers and Arrays

1-D Array and Pointer

- **Array name by itself is an address or pointer which points to the first or 0th element of the array(called base address).**
- **Address of first array element** can be expressed as **either &a[0] or a**. Similarly **address of second array element** can be written as either **&a[1] or a+1** and so on.
- **Value of first array element** can be expressed as **either a[0] or *a** and **value of second array element** can be expressed as **either a[1] or *(a+1)** and so on.

LAB 3: WAP to demonstrate the relationship between arrays and pointer.

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

```
int main(){
```

```
    int a[5]={1,2,3,4,5};
```

```
    int *p=a;    //p is assigned the address of 1st element
```

```
    for(int i=0;i<5;i++){
```

```
        printf("%d\n",*p);
```

```
        p++;    // move the p pointer to the next memory location
```

```
    }
```

```
    for(int i=0;i<5;i++){
```

```
        printf("%u\n",p);
```

```
        p++;
```

```
    }
```

```
    return 0;
```

```
}
```

```
1
2
3
4
5
6684144
6684148
6684152
6684156
6684160
```

9.6 String and Pointer

LAB 4: WAP that demonstrate the relationship between string and pointer.

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

```
int main() {
```

```
    char str[6] = "Hello"; // string variable-Hello\0
```

```
    char *ptr = str; // pointer variable
```

```
    // print the string
```

```
    while(*ptr != '\0') {
```

```
        printf("%c", *ptr);
```

```
        ptr++; // move the ptr pointer to the next memory location
```

```
    }
```

```
    return 0;
```

```
}
```

9.7 Dynamic Memory Allocation

- The process of allocating and freeing memory at run time is known as Dynamic Memory Allocation.
- There are four library functions `malloc()`, `calloc()`, `free()` and `realloc()` for memory management.

1. `malloc()` function: (memory allocation)

Syntax: `ptr = (data_type*)malloc(size_of_block);`

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

2. `calloc()` function: (contiguous allocation)

Syntax: `ptr = (data_type*)calloc(no_of_blocks, size_of_each_block);`

Example: `ptr = (int*)calloc(100, sizeof(int));`

3. free() function:

Dynamically allocated memory is deallocated with the free function.

Example: `free(ptr);`

4. realloc() function:

Syntax:

`ptr = malloc(size);`

`ptr = realloc(ptr, newsize);`

Lab 5: WAP to read an array of integers using DMA and display minimum and maximum value.

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    int n,i,min,max;
    int *ptr;
    printf("Enter the size of block:");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));
    printf("Enter elements of array:");
    for(i=0; i<n; i++){
        scanf("%d",ptr+i);
    }
    min=*ptr;
    max=*ptr;
    for(i=1;i<n;i++){
        if(min>*(ptr+i)){
            min=*(ptr+i);
        }
        if(max<*(ptr+i)){
            max=*(ptr+i);
        }
    }
    printf("The minimum value is: %d",min);
    printf("\nThe maximum value is: %d",max);
    return 0;
}
```

```
Enter the size of block:5
Enter elements of array:20
30
10
50
40
The minimum value is: 10
The maximum value is: 50
```

THANK YOU FOR YOUR ATTENTION