MODULE – 4

Strings and Pointers: Introduction, string taxonomy, operations on strings, miscellaneous string and character functions, arrays of strings.

Pointers: Introduction to pointers, declaring pointer variables, Types of pointers, passing arguments to functions using pointers.

Textbook: Chapter 13.1-13.6, 14 -14.7

CHAPTER 2 POINTERS

2.1 UNDERSTANDING THE COMPUTER'S MEMORY

- The computer's memory is a sequential collection of storage cells as shown in the below Fig.
- Each cell, commonly known as a byte, has a number called address associated with it.
- Typically, the addresses are numbered consecutively, starting from zero.
- The last address depends on the memory size.
- A computer system having 64 K memory will have its last address as 65,535.

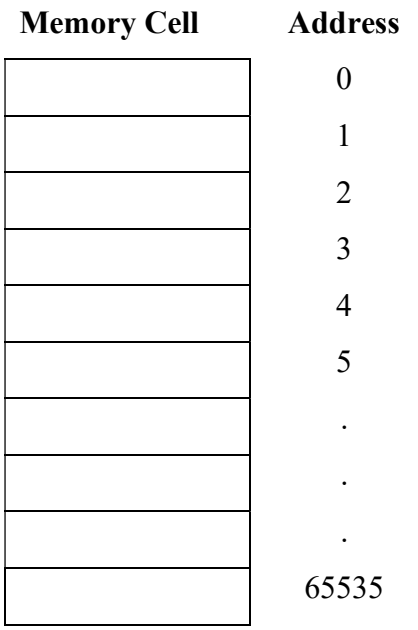


Fig : Memory Organisation

2.2 INTRODUCTION TO POINTERS

- A pointer is a derived data type in C.
- A pointer is variable that contains the memory location of another variable.
- Pointers contain memory addresses as their values.
- Every variable in C has a name and a value associated with it.
- When a variable is declared, a specific block of memory within the computer is allocated to hold the value of that variable.
- The size of the allocated block depends on the data type.
- Since a pointer is a variable, its value is also stored in the memory in another location. Suppose, we assign the address of quantity to a variable p.
- The link between the variables p and quantity can be visualized as shown in Fig. The address of p is 5048.

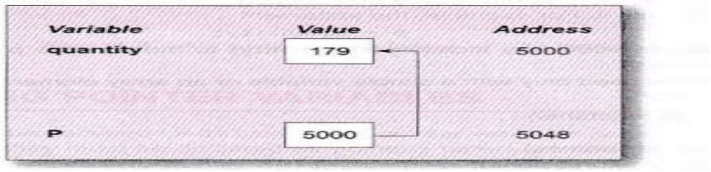


Fig : Pointer Variable

Finding the size of various data types

```
int main()
{
    printf("\n size of integer is %d bytes",sizeof(int));
    printf("\n size of floating point number is %d bytes",sizeof(float));
    printf("\n size of double is %d bytes",sizeof(double));
    printf("\n size of character is %d byte",sizeof(char));
    return 0;
}
```

Output

```
size of integer is 2 bytes
```

size of floating point number is 4 bytes

size of double is 8 bytes

size of character is 1 byte

Pointers are used frequently in C, as they offer a number of benefits (advantages) to the programmers.

They include (Advantages of pointers):

- Pointers are used to pass information back and forth between functions.
- Pointers enable the programmers to return multiple data items from a function via function arguments.
- Pointers provide an alternate way to access the individual elements of an array.
- Pointers are used to pass arrays and strings as function arguments.
- Pointers are used to create complex data structures, such as trees, linked lists, linked stacks, linked queues, and graphs.
- Pointers are used for the dynamic memory allocation of a variable.

Disadvantages of pointers

- If the pointers are not initialized properly, it causes segmentation fault.
- It is difficult to understand and debug
- It leads to memory leakage, if the pointers are not freed after usage in a dynamic memory management.

2.3 DECLARING POINTER VARIABLES

- The general syntax of declaring pointer variables can be given as below.

  data_type *ptr_name;

- Here, data_type is the data type of the value that the pointer will point to.
- For example,

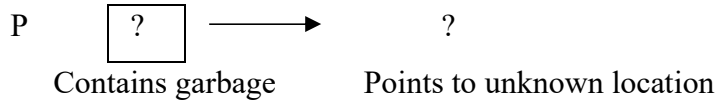
  int *pnum;

  char *pch;

  float *pfnum;
- In each of the above statements, a pointer variable is declared to point to a variable of the specified data type.

- The declarations cause the compiler to allocate memory locations for the pointer variables p.
- Since the memory locations have not been assigned any values, these locations may contain some unknown values in them and therefore they point to unknown locations as shown:

  int *p ;



Initialization Of Pointer Variables

The process of assigning the address of a variable to a pointer variable is known as initialization. Once a pointer variable has been declared we can use the assignment operator **to initialize** the variable.

Example:

```
int x=10;
int *ptr;
ptr =&x;
```

We can also combine **the initialization with the declaration**. That is,

```
int *ptr = &x;
```

is allowed. The only requirement here is that the variable x must be declared before the initialization takes place.

- In the above statement, ptr is the name of the pointer variable.
- The * informs the compiler that ptr is a pointer variable and the int specifies that it will store the address of an integer variable.
- An integer pointer variable, therefore, 'points to' an integer variable.
- In the last statement, ptr is assigned the address of x.
- The & operator retrieves the lvalue (address) of x, and copies that to the contents of the pointer ptr.
- Now, since x is an integer variable, it will be allocated 2 bytes.
- Assuming that the compiler assigns it memory locations 1003 and 1004, the address of x (written as &x) is equal to 1003, that is the starting address of x in the memory.

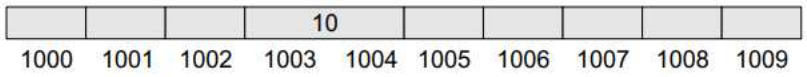


Figure Memory representation

- When we write, ptr = &x, then ptr = 1003.
- We can 'dereference' a pointer, i.e., we can refer to the value of the variable to which it points by using the unary * operator as in *ptr.
- That is, *ptr = 10, since 10 is the value of x.
- Look at the following code which shows the use of a pointer variable:

```
#include
int main()
{
        int num, *ptr;
        ptr = &num;
        printf("\n Enter the number : ");
        scanf("%d", &num);
        printf("\n The number that was entered is : %d", *ptr);
        printf("\n The address of number that was entered is : %d", ptr);
        return 0;
}
```

Output

```
Enter the number : 10
The number that was entered is : 10
The address of number that was entered is : 6487572
```

What will be the value of *(&num)?

It is equivalent to simply writing num.

We could also define a **pointer variable with an initial value of NULL or 0 (zero).** That is, the following statements are valid.

int *p = NULL;

int *p = 0;

POINTER FLEXIBILITY Pointers are flexible.

- We can make the same pointer to point to different data variables in different Statements.

  Example;

  int x, y, z, *p;

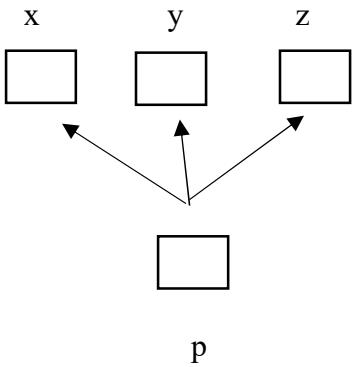
  p = &x;

  …

  p = &y;

  …

  p = &z;



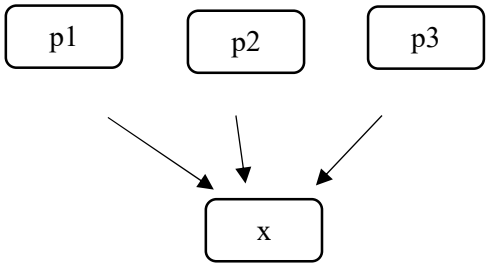
- We can also use different pointers to point to the same data variable. Example;

  int x;

  int *pl = &x;

  int *p2 = &x;

  int *p3 =&x;



With the exception of NULL and 0, no other constant value can be assigned to a pointer variable. For example, the following is wrong:

int *p = 5360; / *absolute address */

Differences between normal variable and a pointer variable

Normal variable	Pointer variable
Stores values	Stores address
Not dereferenced to print the value through variable	Dereferenced with the help of * the indirection operator to print the value(content) through pointer
Dereferenced with the help of & the address of operator to print the address through variable	Not dereferenced to print the address through pointer

Program to add 2 numbers using pointers

```
void main()
{
        int a,b,c,*p,*q;
        printf("enter 2 numbers");
        scanf("%d%d",&a,&b);
        p=&a;
        q=&b;
        c=*p + *q;
        printf("sum=%d",c);
}
```

2.4 TYPES OF POINTERS

Null Pointers

- A pointer variable is a pointer to a variable of some data type.
- However, in some cases, we may prefer to have a null pointer which is a special pointer value and does not point to any value.
- This means that a null pointer does not point to any valid memory address.
- To declare a null pointer, we may use the predefined constant NULL
- We can write int *ptr = NULL;
- We can always check whether a given pointer variable stores the address of some variable or contains NULL by writing,

```
if (ptr == NULL)
{
```

```
        Statement block;
}
```

- We may also initialize a pointer as a null pointer by using the constant 0

```
int *ptr;
ptr = 0; This is a valid statement in C
```

Generic Pointers

- A generic pointer is a pointer variable that has void as its data type.
- The void pointer, or the generic pointer, is a special type of pointer that can point to variables of any data type.
- It is declared like a normal pointer variable but using the void keyword as the pointer's data type.
- For example, void *ptr;
- In C, since we cannot have a variable of type void, the void pointer will therefore not point to any data and, thus, cannot be dereferenced.
- We need to cast a void pointer to another kind of pointer before using it.
- Generic pointers are often used when you want a pointer to point to data of different types at different times.

```
#include<stdio.h>
int main()
{
        int x=10;
        char ch = 'A';
        void *gp;
        gp = &x;
        printf("\n Generic pointer points to the integer value = %d", *(int*)gp);
        gp = &ch;
        printf("\n Generic pointer now points to the character= %c", *(char*)gp);
        return 0;
}
```

Output

```
Generic pointer points to the integer value = 10
```

Generic pointer now points to the character = A

Double Pointers (Chain of Pointers or Pointers to Pointers)

Double pointers is used to make a pointer to point to another pointer, thus creating a chain of pointers.

```
void main()
{
        int a=10;
        int *p1,**p2;
        p1=&a;
        p2=&p1;
        printf("%d",**p2);
}
```

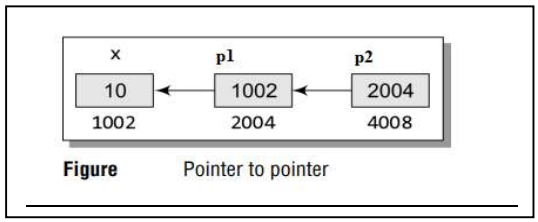


Figure Pointer to pointer

Output

10

Here p2 contains address of p1. This is also known as multiple indirections.

int **p2; tells the compiler that p2 is a pointer to a pointer of int type.

2.5 POINTER EXPRESSIONS AND POINTER ARITHMETIC

POINTER EXPRESSIONS

Consider p1 and p2 are pointer variables:

1. Like any other variable, content at pointer can be used in expression.
   y=*p1 * *p2
2. Pointers can be compared: p1==p2, p1>p2 is valid
3. p1/p2, p1*p2, p1+p2 is illegal
4. p2-p1 is legal; if p1 and p2 are character pointers to same array. Then p2-p1 gives the number of number of elements between p1 and p2.
5. We can add or subtract integers; p1+5, p1-2
6. p1++, p1-- is legal.
7.

POINTER ARITHMETIC (ADDRESS ARITHMETIC) POINTERS INCREMENT AND SCALE FACTOR

Let us consider starting address as 8000. Then the operation p++ is done to point to next subsequent element:

1. int
   p++ (i.e) p=p+1 address is 8002.
2. char
   p++ (i.e) p=p+1 address is 8001.
3. float
   p++ (i.e) p=p+1 address is 8004.
4. double
   p++ (i.e) p=p+1 address is 8008.

Let us consider starting address as 8008. Then the operation p-- is done to point to previous element:

1. int
   p-- (i.e) p=p-1 address is 8006.
2. char
   p-- (i.e) p=p-1 address is 8007.
3. float
   p-- (i.e) p=p-1 address is 8004.
4. double
   p-- (i.e) p=p-1 address is 8000.

Let us consider starting address as 8000. Then the operation p+5 is done to point to the fifth element from that element (i.e) element at index 5:

1. int
   p=p+5 (i.e) p=p+5 (starting address+index*sizeof(int))=8000+5*2
   address is 8010.
2. char
   p=p+5 (i.e) p=p+5 (starting address+index*sizeof(char))=8000+5*1
   address is 8005.
3. float
   p=p+5 (i.e) p=p+5 (starting address+index*sizeof(float))=8000+5*4
   address is 8020.

4. double

   p=p+5 (i.e) p=p+5 (starting address+index*sizeof(double))=8000+5*8

   address is 8040.

Let us consider starting address as 8040. Then the operation p-5 is done to point to the five elements before that element:

1. int

   p=p-5 (i.e) p=p-5 (starting address-index*sizeof(int))=8040-5*2

   address is 8030.

2. char

   p=p-5 (i.e) p=p-5 (starting address-index*sizeof(char))=8040-5*1

   address is 8035.

3. float

   p=p-5 (i.e) p=p-5 (starting address-index*sizeof(float))=8040-5*4

   address is 8020.

4. double

   p=p-5 (i.e) p=p-5 (starting address-index*sizeof(double))=8040-5*8

   address is 8000.

When we increment a pointer, its value is increased by the length of the data type that its points. This length is called scale factor.

- ➔ Scale factor for **int** in a 16-bit machine is **2 bytes** (i.e) the size of the datatype.
- ➔ Scale factor for **char** in a 16-bit machine is **1 byte** (i.e) the size of the datatype.
- ➔ Scale factor for **float** in a 16-bit machine is **4 bytes** (i.e) the size of the datatype.
- ➔ Scale factor for **double** in a 16-bit machine is **8 bytes** (i.e) the size of the datatype.

2.6 POINTERS AND ARRAYS

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

Suppose the base address is 8000, each integer requires two bytes.

Here **a** refers to starting address. Also, &a[0] refers to the starting address.

Initialization

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

(or)

int a[5];

int *p;

p=&a[0];

Now, we can access the next element in the array by using p++.

p=&a[0] 8000

p+1=&a[1] 8002

p+2=&a[2] 8004

p+3=&a[3] 8006

p+4=&a[0] 8008

Program

```
void main()
{
    int a[10],n,I,*p;
    printf("enter n");
    scanf("%d",&n);
    printf("enter elements");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    p=a;
    printf("elements are");
    for(i=0;i<n;i++)
        printf("%d\t",*(p+i));
}
```

OUTPUT

enter n 5

enter elements 1 2 3 4 5

elements are 1 2 3 4 5

2.7 PASSING ARGUMENTS TO FUNCTION USING POINTERS

- When an array is passed to a function as an argument, only the address of the first element of the array is passed. It works like call by reference.

- Similarly, we can pass the address of a normal variable as an argument to function- It works like functions that return multiple values

In above both cases the parameters receiving the address should be pointers- so it works like call by reference. So changes in the formal parameters will affect the actual parameters.

- The function parameters are declared as pointers or arrays.
- Dereferenced pointers are used in function body
- When the function is called, the addresses are passed as actual arguments.

Example : To swap two numbers

```
void swap(int  *p,int  *q);
void main()
{
        int a,b;
        printf("enter two numbers");
        scanf("%d%d",&a,&b);
        printf("before swapping");
        printf("a=%d,b=%d",a,b);
        swap(&a,&b);
        printf("after swapping");
        printf("a=%d,b=%d",a,b);
}
void swap(int  *p,int  *q)
{
        int t;
        t=*p;
        *p=*q;
        *q=t;
}
```

Output:

```
enter two numbers 5 4
before swapping
a=5,b=4
```

after swapping

a=4,b=5

program (bubble sort)

```
#include<stdio.h>
void bubblesort(int a[20],int n);
void main()
{
	intn,a[20],i,j, temp;
	printf("enter the number of elements n");
	scanf("%d",&n);
	printf("enter the array elements");
	for(i=0;i<n;i++)
	{
		scanf("%d",&a[i]);
	}
	bubblesort(a,n)
	printf("\n the sorted elements are\n");
	for(i=0;i<n;i++)
	{
		printf ("%d\t",a[i]);
	}
}
void bubblesort(int a[20],int n)
{
	int  i,j;
	for(i=0;i<n-1;i++)
	{
		for(j=0;j<n-1-i;j++)
		{
			if(a[j]>a[j+1])
			{
```

```
                    temp=a[j];
                    a[j]=a[j+1];
                    a[j+1]=temp;
                }

            }
        }
```

Output

enter the number of elements n5

enter the array elements 50 20 40 10 30

the sorted elements are

10 20 30 40 50

2.8 Troubles with Pointers (Drawbacks of Pointers)

In most of the cases, compiler may not detect the error and produce unexpected results, when we use pointers. The output does not give us a clue regarding where we went wrong.

Debugging is difficult.

Some possible errors

- Assigning values to uninitialized pointers

```
int m=10,*p;
*p=m;
```

- Assigning a value to a pointer

```
int m=10,*p;
p=m;
```

- Not dereferencing to print the value

```
int m=10,*p;
p=&m;
printf("%d",p);
```

- Assigning the address of uninitialized variable

```
int m,*p;
p=&m;
```

- Comparing pointers that point to different objects

```
char name1[20],name2[20];
char *p1=name1;
char  *p2=name2;
if(p1>p2)
```

Lab Program 11

Develop a program using pointers to compute the sum, mean and standard deviation of all elements stored in an array of N real numbers.

Algorithm

Step 1: [Initialize]
Start

Step 2:[Read the no of elements and array elements]
Read n
Read a[]

Step 3: [Set starting address of array to a pointer variable]
ptr=a

Step 4:[Iterate using a for loop to find sum using pointers]
for(i=0;i<n;i++)
sum=sum+*ptr
ptr++
end for

Step 5:[Calculate mean]
mean=sum/n

Step 6: [Set starting address of array to a pointer variable]
ptr=a

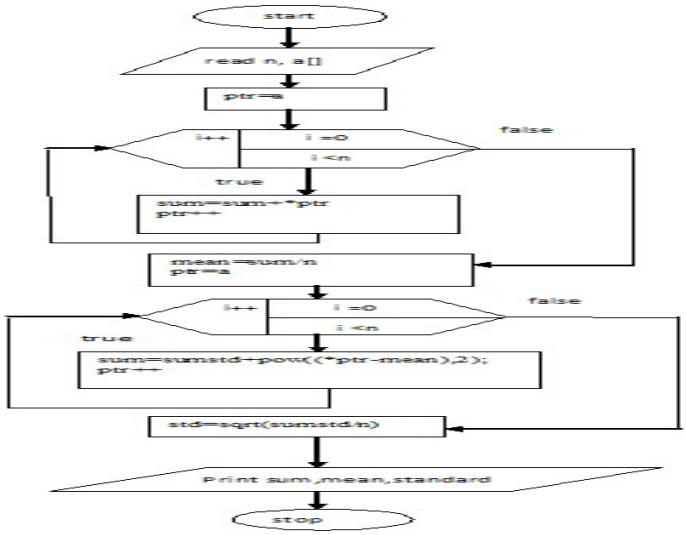
Step 7:[Iterate using a for loop to find sumstd using pointers]
for(i=0;i<n;i++
sumstd=sumstd+pow((*ptr-mean),2)
ptr++
end for

Step 8:[Calculate standard deviation]
std=sqrt(sumstd/n)

Step 9:[Display the result]
Print sum,mean,std

Step 10:[Finished]
Stop

Flow Chart



Program

```
#include<stdio.h>
#include<math.h>
int main()
{
        float a[10],*ptr,mean,std,sum=0,sumstd=0;
        int n,i;
        printf("\n Enter the number of elements");
        scanf("%d",&n);
        printf('\n Enter the array elements");
        for(i=0;i<n;i++)
        {
                scanf("%f",&a[i]);
        }
        ptr=a;
        for(i=0;i<n;i++)
        {
                sum=sum+*ptr;
                ptr++;
        }
        mean=sum/n;
        ptr=a;
        for(i=0;i<n;i++)
        {
                sumstd=sumstd+pow((*ptr-mean),2);
                ptr++;
        }
        std=sqrt(sumstd/n);
        printf("Sum=%f\n",sum);
        printf("Mean=%f\n",mean);
        printf("Standard Deviation=%f\n",std);
```

```
    return 0;
}
```

Test cases

Test No	Input Parameters	Expected Output	Obtained Output
1	Enter the number of elements 5 Enter the array elements 1 5 9 6 7	Sum= 28 Mean= 5.6 Standard Deviation= 2.09	Sum= 28 Mean= 5.6 Standard Deviation= 2.09
2	Enter the number of elements 4 Enter the array elements 2.3 1.1 4.5 2.78	Sum= 10.68 Mean= 2.67 Standard Deviation= 0.863	Sum= 10.68 Mean= 2.67 Standard Deviation= 0.863