F.Y.B.Sc.IT-SEM 1

Programming Principles With C
(PUSIT101)

By,
Prof. Nikita Madwal

Unit IV

5. Pointer and Arrays

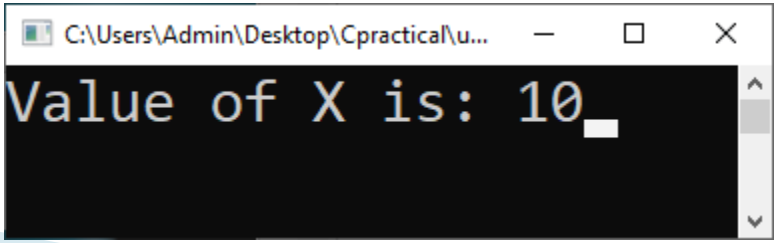
Why we need Array?

- For understanding the arrays properly, let us consider the following program:

usearray.c

```
1  #include<stdio.h>
2  void main()
3  {
4      int x;
5      x=5;
6      x=10;
7      printf("Value of X is: %d",x);
8      getch();
9  }
```

Output :



- No doubt, this program will print the value of x as 10. Why so? Because when a value 10 is assigned to x, the earlier value of x, i.e. 5, is lost. Thus, ordinary variables (the ones which we have used so far) are capable of holding only one value at a time (as in the above example). However, there are situations in which we would want to store more than one value at a time in a single variable.

- For example, suppose we wish to arrange the percentage marks obtained by 60 students in ascending order. In such a case we have two options to store these marks in memory:
- a) Construct 60 variables to store percentage marks obtained by 60 different students, i.e. each variable containing one student's marks.
- b) Construct one variable (called array or subscripted variable) capable of storing or holding all the sixty values.
- Obviously, the second alternative is better. A simple reason for this is, it would be much easier to handle one variable than handling 60 different variables.

Array

- An **array** is a **variable** to **store a group of elements of same data type.**
- Array might be belonging to any of the data types
- In array **only one type of elements into an array**
- The array can **store more than one value at a time in a single variable.**
- In an array, its elements are always stored in contiguous memory location
- An array is also known as a **subscripted variable**
- Before using an array its type and dimension must be declared
- **Types of C arrays:**
- **There are 2 types of C arrays.**
- One dimensional array
- Two dimensional or Multi dimensional array

One Dimensional Array

- **Array declaration**

data_type arr_name[arr_size];

e.g.

int age [5];

char str[10];

- **Array initialization**

data_type arr_name [arr_size]={value1, value2,….};

e.g.

int age[5]={0, 1, 2, 3, 4};

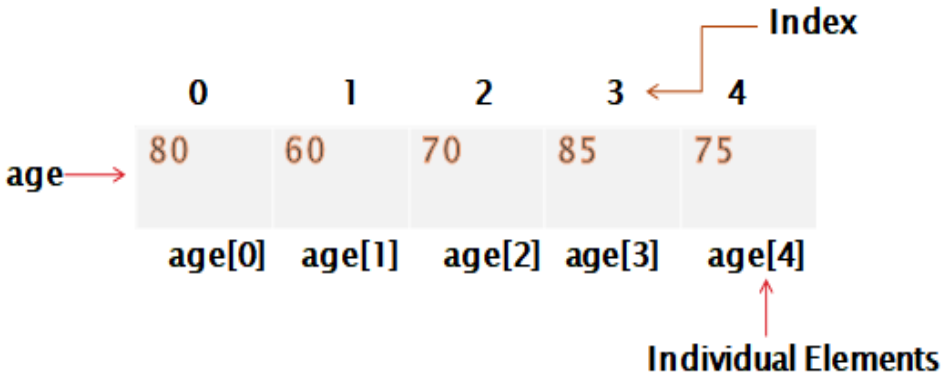
int n[ ] = { 2, 4, 12, 5, 45, 5 } ;

char str[10]={‘H’,‘a’,‘i’};

OR

```
age[0]=80;
age[1]=60;
age[2]=70;
age[3]=85;
age[4]=75;
```

```
char str[0] = ‘H’;
char str[1] = ‘a’;
char str[2] = ‘I’;
```



- **Accessing array**

arr_name[index];

e.g.

```
str[0];  /*H is accessed*/
str[1];  /*a is accessed*/
str[2];  /* i is accessed*/
```

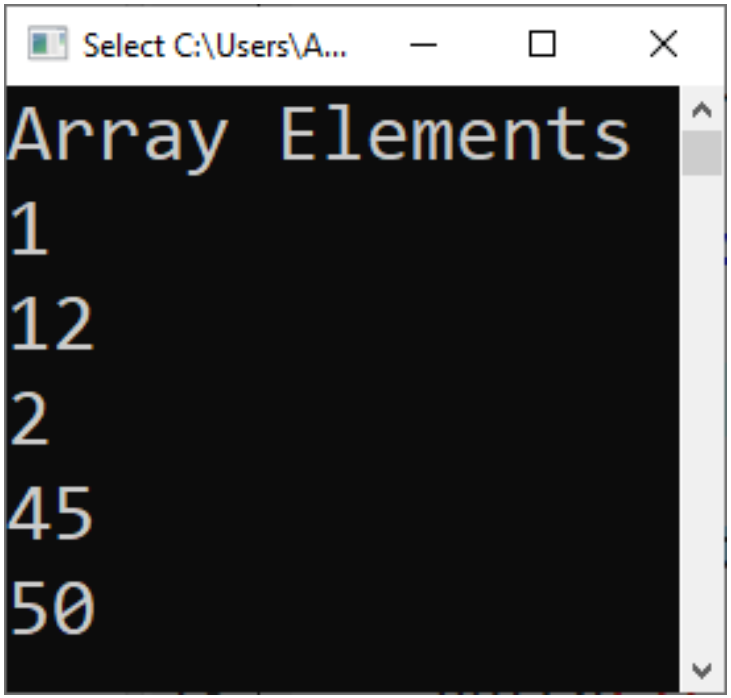
```
age[0];  /*0_is_accessed*/
age[1];  /*1_is_accessed*/
age[2];  /*2_is_accessed*/
```

e.g.

arr.c

```c
#include<stdio.h>
void main()
{
    int i;
    int my_array[5]={1,12,2,45,50}; //declaration and initialisation of array

    printf("Array Elements\n");

    for (i = 0; i < 5; i++)
    {
        printf("%d\n",my_array[i]);
    }
    getch();
}
```

Output :

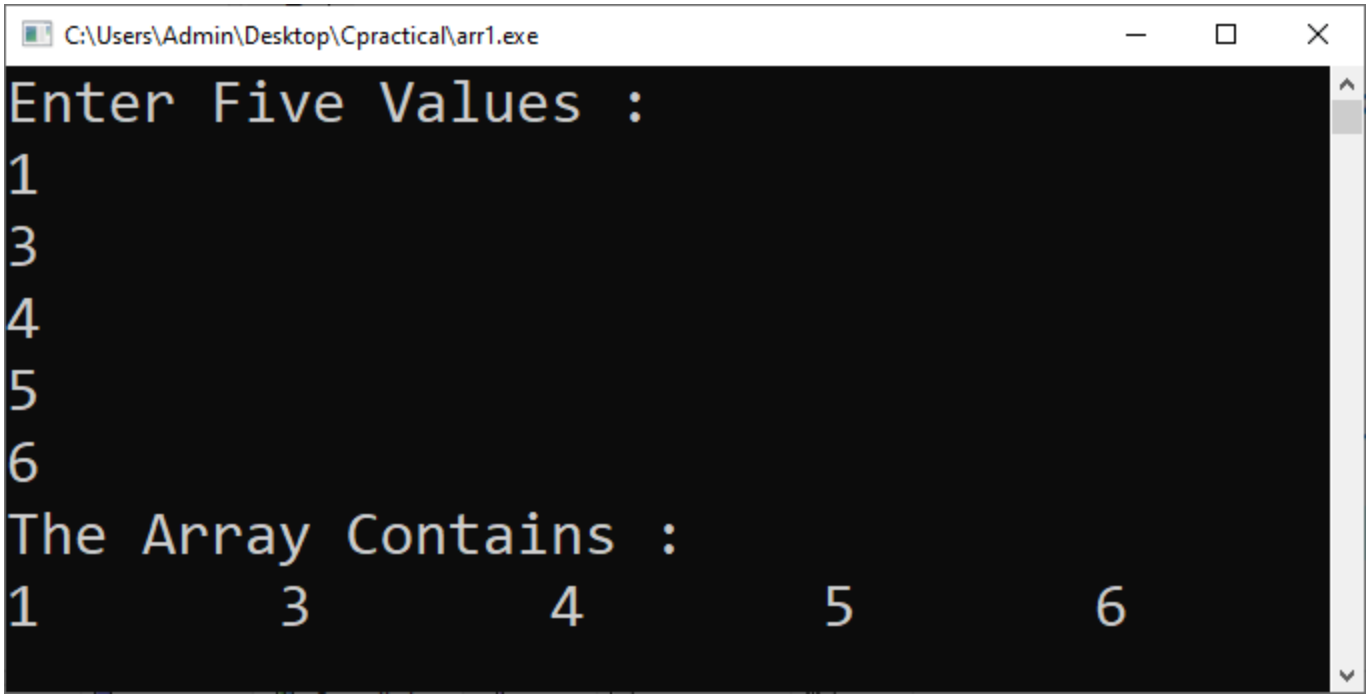


e.g.

arr1.c

```c
#include <stdio.h>
void main()
{
    int i,a[5];

    printf("Enter Five Values :\n");
    for (i = 0; i < 5; i++) //reading of values
    {
        scanf("%d",&a[i]);
    }

    printf("The Array Contains :\n");
    for (i = 0; i < 5; i++) //Prints all the data of array
    {
        printf("%d\t",a[i]);
    }
    getch();
}

```

Output :


C:\Users\Admin\Desktop\Cpractical\arr1.exe
Enter Five Values :
1
3
4
5
6
The Array Contains :
1 3 4 5 6

Two Dimensional Array OR Multidimensional Array

- Two dimensional array is nothing but array of array
- This is also called as matrix

❖ **Array declaration**

data_type arr_name [num_of_rows][num_of_column];

e.g

int arr[2][3];

❖ **Array initialization**

e.g

int arr[2][3] = {1,2, 3, 4,5,6};

OR

int arr[2][3]={{1,2,3},{4,5,6}};

❖ **Accessing array**

arr_name[index];

e.g

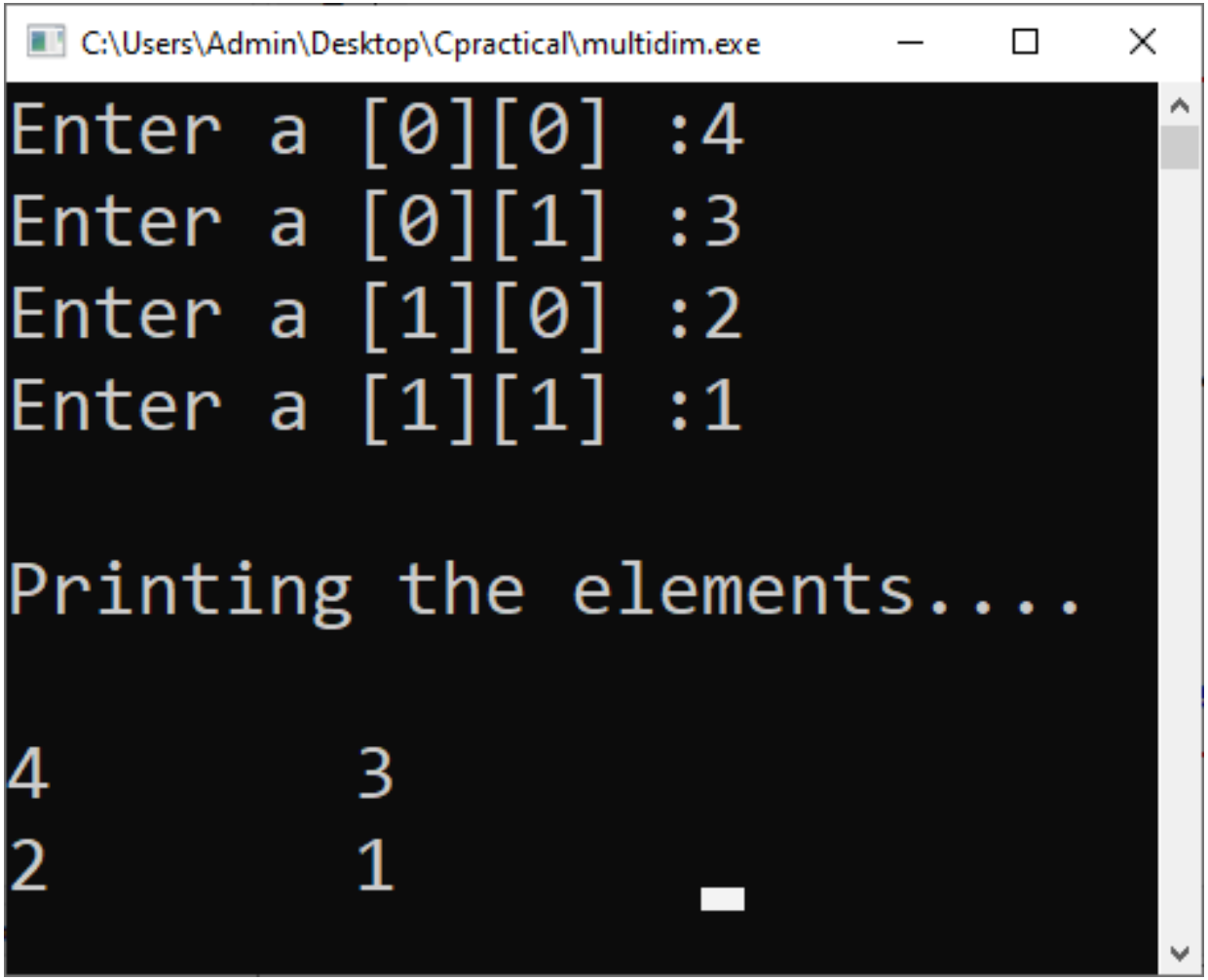
arr [0] [0] = 1;

arr [0] [1] = 2;

multidim.c

```c
#include <stdio.h>
void main()
{
    int i,j,arr[2][2];
    for (i = 0; i < 2; i++)
    {
        for(j = 0; j < 2; j++)
        {
            printf("Enter a [%d][%d] :",i,j);
            scanf("%d",&arr[i][j]);
        }

    }
    printf("\nPrinting the elements....\n");
    for (i = 0; i < 2; i++)
    {
        printf("\n");
        for(j = 0; j < 2; j++)
        {
            printf("%d\t",arr[i][j]);
        }
    }
    getch();
}
```

Output :


C:\Users\Admin\Desktop\Cpractical\multidim.exe
Enter a [0][0] :4
Enter a [0][1] :3
Enter a [1][0] :2
Enter a [1][1] :1
Printing the elements....
4 3
2 1

ARRAYS AND STRINGS

- A string is a **sequence of characters** that is treated as a single data item.
- Strings are actually one-dimensional array of characters terminated by a **null** character '\0'.

- **String Variable Declaration**

char string_variablename[size];

e.g

char name[20];

- **String Variable Initialization**
  - When the compiler assigns a character string to a character array, it automatically supplies a *null character* (‘\0’) at the end of the string

e.g

char name[4]=“RAM”;

char name[]=“RAM”;

OR

- When we initialize a character array by listing its elements, the null terminator or the size of the array must be provided explicitly.

e.g

char name[4]={‘R’,‘A’,‘M’, ‘\0’};

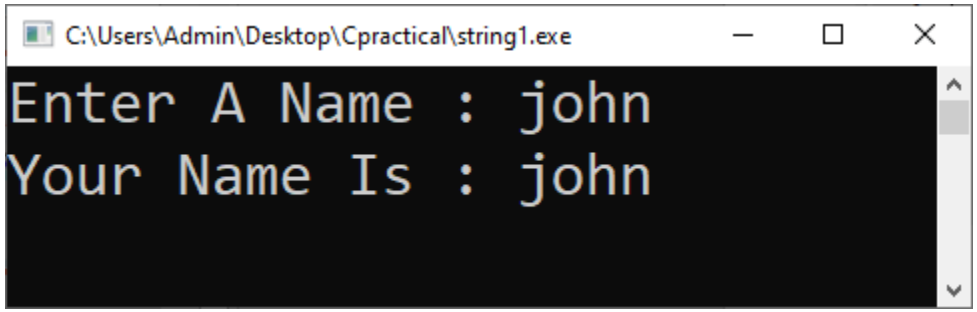
char name[]={‘R’,‘A’,‘M’, ‘\0’};

e.g.

string1.c

```c
#include<stdio.h>
void main()
{
 char name[20];

 printf("Enter A Name : ");
 scanf("%s",&name);
 printf("Your Name Is : %s",name);
 getch();
}
```

Output :



String Handling function

- C programming language provides a set of pre-defined functions called **string handling functions** to work with string values.
- The string handling functions are defined in a header file called **string.h**.

- **gets()**

function enables the user to enter string

- **puts()**

function is used to print the string on the console which is previously read by using gets() or scanf() function

- **strcat()**

This function is used to concatenates two given strings(Appends one string at the end of another)
(*for example refer unit 3-chp 4 ppt*)

- **strcmp( )**

strcmp( ) function compares two given strings (*for example refer unit 3-chp 4 ppt*)

- **strcpy( )**

strcpy( ) function copies contents of one string into another string. (*for example refer unit 3-chp 4 ppt*)

- **strrev( )**

strrev() function reverses the given string (*for example refer unit 3-chp 4 ppt*)

- **strlen( )**

strlen( ) function gives the length of the given string. (*for example refer unit 3-chp 4 ppt*)

- **strlwr( )**

strlwr() function converts a string to lowercase

- **strupr( )**

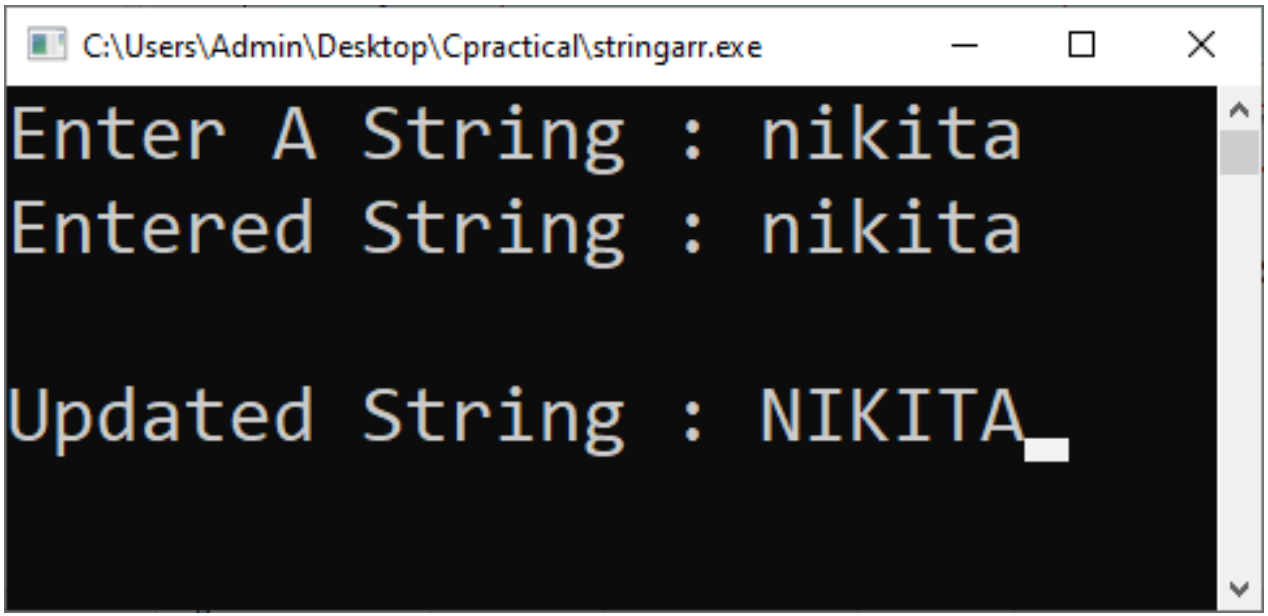
strupr() function converts a string to uppercase.

e.g. 1) strupr()

stringarr.c

```c
#include<stdio.h>
#include<string.h>
void main()
{
 char str[20];

 printf("Enter A String : ");
 scanf("%s",&str);
 printf("Entered String : %s",str);

 printf("\n\nUpdated String : %s",strupr(str));
 getch();
}
```

Output :


C:\Users\Admin\Desktop\Cpractical\stringarr.exe
Enter A String : nikita
Entered String : nikita
Updated String : NIKITA

Pointer

- In the c programming language, we use normal variables to store user data values. When we declare a variable, the compiler allocates required memory with the specified name
- **Pointers** in C language is a **special type of variable** that **stores/points the address of another variable**.
- The pointer variable might be **belonging to any of the data type** such as int, float, char, double, short etc.
- For example, an integer variable holds (or stores) an integer value, however an integer pointer holds the address of a integer variable.
- **The data type of pointer and the variable must match**, an *int pointer can hold the address of int variable*, similarly a pointer declared with *float data type can hold the address of a float variable*.

- It makes you able to **access any memory location**
- Pointers support dynamic memory management
- The purpose of pointer is to save memory space and achieve faster execution time.
- Execution time with pointers is faster because data are manipulated with the address, that is, direct access to memory location.

- **Declaring pointer variable**

  data_type *pointer_name;

  e.g

  int *ptr;

- **Accessing the address of variable**

  ptr=&n;

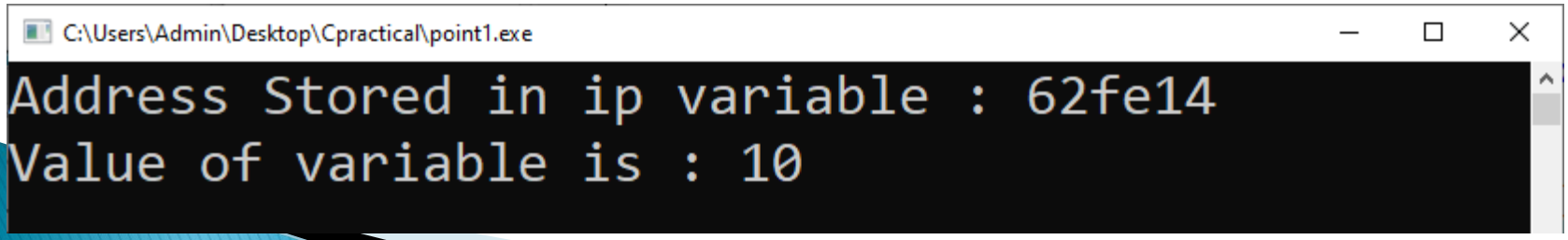
NOTE:

- **"&" symbol is used** to get the **address of the variable**.
- "&" this is also known as "*Address of operator*"
- Where, **"*" is used** to **denote** that "ptr" is **pointer variable** and not a normal variable and also **"*"symbol is used** to get the **value of the variable** that the **pointer is pointing** to.
- "*" this is also known as "*dereferencing operator*"

point1.c

```c
#include<stdio.h>
void main()
{
    int a=10;  //actual variable declaration
    int *ip;  //pointer variable declaration

    ip=&a;   //stores the address of a in pointer variable

    /*Address Stored in pointer variable*/
    printf("Address Stored in ip variable : %x\n",ip );

    /*Accessing the value using pointer*/
    printf("Value of variable is : %d\n",*ip );
    getch();
}
```

Output :



Pointer Arithmetic

- A pointer in C is an address, which is a numeric value. Therefore, the arithmetic operations can be performed on a pointer just as on a numeric value.
- There are arithmetic operators that can be used on pointers such as : + (Addition) and – (Subtraction)
- When the arithmetic operation performed on a pointer ,**it will display the changes by the "length" of the data type** that is points to . This length is called the scale factor.
- The length of various data types as follows

  Character 1 byte

  Integer 2 bytes or 4 bytes

  Float 4 bytes

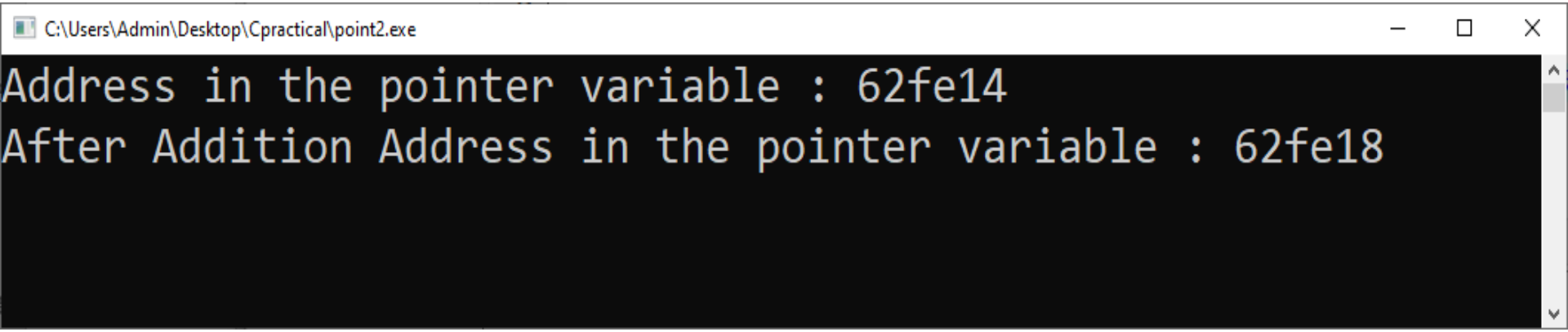
  Doubles 8 bytes

Addition: The value can be added to pointer variable .Adding any number to a pointer will give an address

point2.c

```c
#include<stdio.h>
void main()
{
    int number=10;
    int *ptr;

    ptr=&number;
    printf("Address in the pointer variable : %x\n",ptr);

    ptr=ptr+1;
    printf("After Addition Address in the pointer variable : %x\n",ptr);
    getch();
}
```

Output :

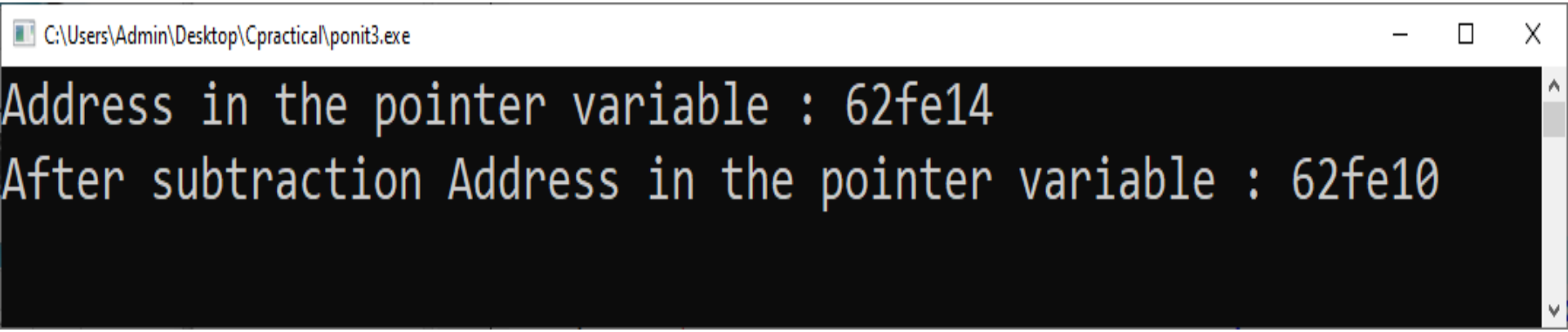

C:\Users\Admin\Desktop\Cpractical\point2.exe
Address in the pointer variable : 62fe14
After Addition Address in the pointer variable : 62fe18

Subtraction: Like pointer addition, The value can be subtracted from the pointer variable. Subtracting any number from a pointer will give an address.

ponit3.c

```c
#include<stdio.h>
void main()
{
    int number=10;
    int *p;

    p=&number;
    printf("Address in the pointer variable : %x\n",p);

    p=p-1;
    printf("After subtraction Address in the pointer variable : %x\n",p);
    getch();
}
```

Output :


C:\Users\Admin\Desktop\Cpractical\ponit3.exe
Address in the pointer variable : 62fe14
After subtraction Address in the pointer variable : 62fe10

Functions and Pointers

- 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.
- The process of calling a function using pointers to pass the addresses of variables is called as “call by address” or “call by reference”.

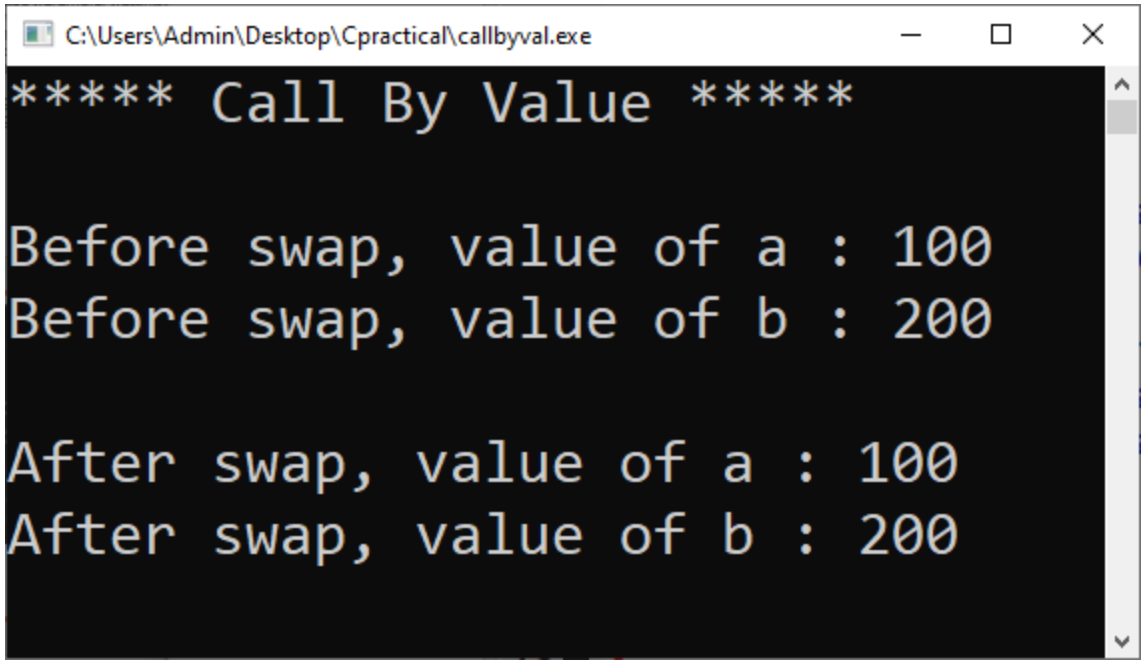
Call by value

- By default, C uses **call by value** to pass arguments
- In **call by value** parameter passing method, the copy of actual parameter(Arguments which are mentioned in the function call) values are copied to formal parameters(Arguments which are mentioned in the definition) and these formal parameters are used in called function. **The changes made on the formal parameters does not effect the values of actual parameters**.

callbyval.c

```c
#include <stdio.h>
void swap(int x, int y);
void main()
{
   printf("***** Call By Value *****");
   int a = 100;
   int b = 200;
   printf("\n\nBefore swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n\n", b );

   swap(a, b);   /* calling a function to swap the values */
   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );
   getch();
}

void swap(int x, int y)
{
   int temp;
   temp = x; /* save the value of x */
   x = y;    /* put y into x */
   y = temp; /* put temp into y */
}
```

Output :


C:\Users\Admin\Desktop\Cpractical\callbyval.exe
***** Call By Value *****
Before swap, value of a : 100
Before swap, value of b : 200
After swap, value of a : 100
After swap, value of b : 200

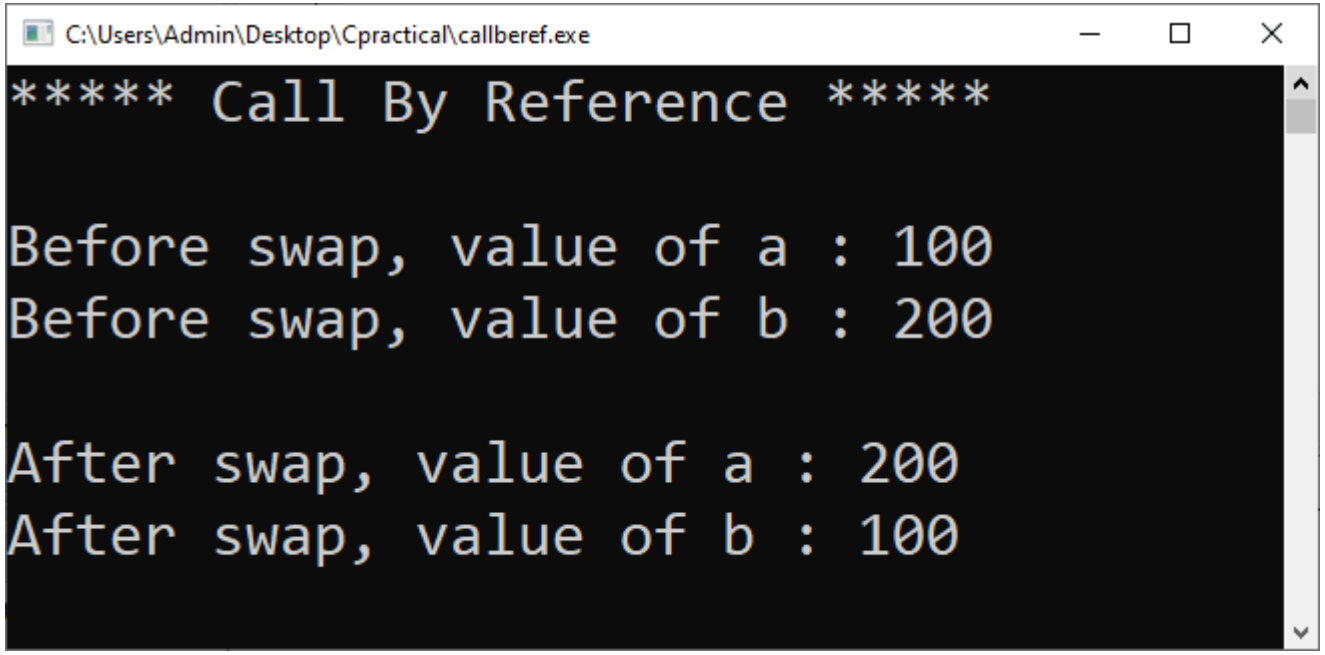
Call by reference

- In **Call by Reference** parameter passing method, the memory location address of the actual parameters is copied to formal parameters.
- In call by reference parameter passing method, the address of the actual parameters is passed to the called function and is received by the formal parameters
- Whenever we use these formal parameters in called function, they directly access the memory locations of actual parameters. So **the changes made on the formal parameters effects the values of actual parameters**

callberef.c

```c
#include <stdio.h>
void swap(int *x, int *y);
void main()
{
    printf("***** Call By Reference *****");
    int a = 100,b = 200;

    printf("\n\nBefore swap, value of a : %d\n", a );
    printf("Before swap, value of b : %d\n\n", b );

    swap(&a,&b); //address of a abd b is passed
    printf("After swap, value of a : %d\n", a );
    printf("After swap, value of b : %d\n", b );
    getch();
}
void swap(int *x, int *y)
{
    int temp;
    temp = *x; /* save the value of x */
    *x = *y;    /* put y into x */
    *y = temp; /* put temp into y */
}
```

Output :


C:\Users\Admin\Desktop\Cpractical\callberef.exe
***** Call By Reference *****
Before swap, value of a : 100
Before swap, value of b : 200
After swap, value of a : 200
After swap, value of b : 100

Arrays and Pointers

- When an array in C language is declared, compiler allocates a base address and sufficient memory to contain all its elements of the array in contiguous memory locations.
- If we declare p is an integer pointer ,then we can make the pointer p to point to the array by the following way:
- Declaration of array and pointer

```
int a[10];
        OR
int a[]={1,2,3,4,5};
int *p;
```

- Assigning the address of array to pointer

```
p=a;    (this is equivalent to →   p=&a[0];)
```

pointarr.c

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

   int  a[10],i,*p;
   printf("\n enter 10 numbers");
       for(i=0;i<10;i++)
       {
        scanf("%d",&a[i]);
       }
   p=a;
   printf("\n array elements accessed through pointer");
   printf("\n .....\n");
       for(i=0;i<10;i++)
       {
        printf("a[%d]\t",i);
       }
       for(i=0;i<10;i++)
       {
        printf("%d\t",*p);
        p++;
       }
   getch();
}
```

Output :

```
enter 10 numbers
1
2
3
4
5
6
7
8
9
10

array elements accessed through pointer
.....................................
a[0]    a[1]    a[2]    a[3]    a[4]    a[5]    a[6]    a[7]    a[8]    a[9]
1       2       3       4       5       6       7       8       9       10
```

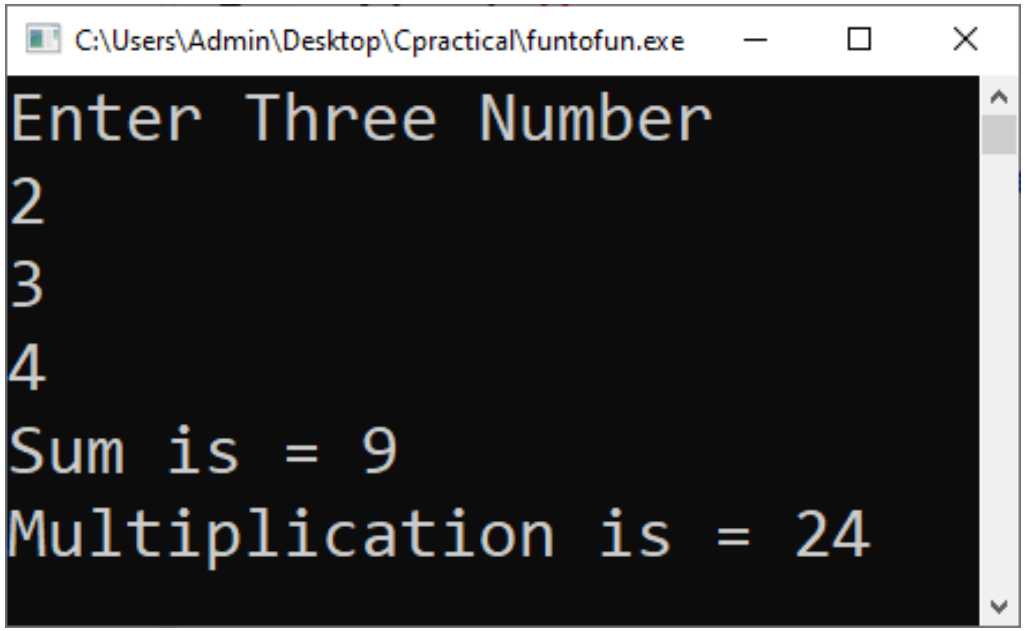
Passing functions to other functions

- A **function pointer** is a pointer that stores the **memory address of a function**
- Function pointer are pointers i.e. variables, which point to the address of a function.
- Both ,the executable compiled code and the used variables ,are put inside the main memory

funtofun.c

```c
#include<stdio.h>
void (*f2p)(int,int,int);

void sum(int m,int n,int o);
void prod(int m,int n,int o);
void main()
{
    int a,b,c;
    printf("Enter Three Number");
    scanf("%d",&a);
    scanf("%d",&b);
    scanf("%d",&c);

    f2p=&sum;
    f2p(a,b,c);

    f2p=&prod;
    f2p(a,b,c);

    getch();
}
void sum(int m,int n,int o)
{
    printf("\nAddition is = %d",(m+n+o));
}
void prod(int m,int n,int o)
{
    printf("\nMultiplication is = %d",(m*n*o));
}
```

Output :


C:\Users\Admin\Desktop\Cpractical\funtofun.exe
Enter Three Number
2
3
4
Sum is = 9
Multiplication is = 24