INSTITUTE OF ENGINEERING

Pulchowk Campus, Lalitpur

Subject: C Programming

Lab Report 9

Tittle: **Pointer**

Submitted by:

Susheel Thapa 077BCT090

Submitted to:

Department of Electronics and

Computer Engineering

Checked by

Content of Lab Report:

Background Information

What is C Programming?

C programming is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system. C is the most widely used computer language.

Why to Learn C Programming?

- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities
- It can be compiled on a variety of computer platforms

Editor

Here, I have used Visual Studio Code as my editor. You can download the editor from Download Visual Studio Code - Mac, Linux, Windows . Select your operating system and download it.

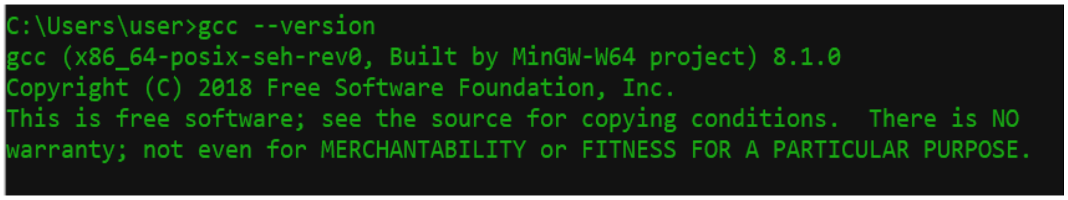
Compiler

Here, I have used **gcc** as my compiler provided by MinGWw64. You can download it via Download MinGW-w64 - for 32 and 64 bit Windows from SourceForge.net. Your download will start automatically. Run the downloaded .exe file. After, you have installed MinGW-w64, you need to configure it.

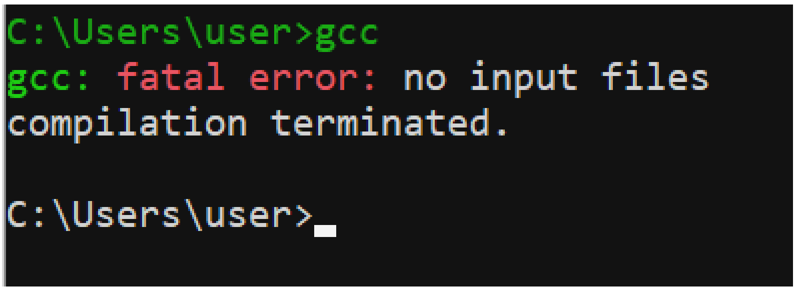
- In the Windows search bar, type 'settings' to open your Windows Settings.
- Search for Edit environment variables for your account.
- Choose the Path variable and then select Edit.
- Select New and add the Mingw-w64 destination folder path to the system path. The exact path depends on which version of Mingw-w64 you have installed and where you installed it. If you used the settings above to install Mingw-w64, then add this to the path: **C:\Program Files\mingw-w64\x86_64-8.1.0-posix-seh-rt_v6-rev0\mingw64\bin.**
- Select OK to save the updated PATH. You will need to reopen any console windows for the new PATH location to be available.

Check your installation

Open command prompt or power shell and type:



```
C:\Users\user>gcc --version
gcc (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```



```
C:\Users\user>gcc
gcc: fatal error: no input files
compilation terminated.

C:\Users\user>
```

If you get similar result, you are good to go.

POINTERS AND DYNAMIC MEMORY ALLOCATION

Pointer:

Pointer is a variable that represents the location of a data item rather than the value. A pointer variable of particular type can hold the address of its own type only.

Why is pointer useful?

- Pointers are useful as they are more efficient in handling arrays and data tables.
- It can be used to return multiple values from a function via function arguments.
- It allows passing a function as argument to other functions.
- The use of pointer arrays to character strings results in saving of data storage space in memory.
- It provides an efficient way for manipulating dynamic data structures, linked lists, queues, stacks and trees.

Pointer variable declaration:

The general syntax of pointer declaration is,

datatype *pointer_name;

Copy

The data type of the pointer and the variable to which the pointer variable is pointing must be the same.

Pointer Initialization:

Pointer Initialization is the process of assigning address of a variable to a pointer variable. It contains the address of a variable of the same data type. In C language address operator & is used to determine the address of a variable. The & (immediately preceding a variable name) returns the address of the variable associated with it.

int a = 10;

int *ptr;   //pointer declaration

ptr = &a;   //pointer initialization

Pointer expression:

Pointers are valid operands in arithmetic expressions, assignment expressions and comparison expressions. However, not all the operators normally used in these expressions are valid with pointer variables.

Arithmetic operations on pointer variables:

Declaring variables as :

int a,b,*p,*q;

- A pointer variable can be assigned the address of an ordinary variable(eg. p=&a)
- A pointer variable can be assigned the content of another pointer variable provided both pointer point to objects of same data type.(eg. p=q)
- An integer quantity can be added to or subtracted from a pointer variable. (eg. p+5, q-1, q++, --p)
- A pointer variable can be assigned a null value. (eg. p=NULL)
- One pointer variable can be subtracted from another provided both pointer point to elements of same array.

- Two pointer variables can be compared provided both pointer point to element of the same data type.

Following operations are not allowed on pointer variables:

1. Pointer variables cannot be multiplied or divided by a constant.
2. Two pointer variables cannot be added.

Dynamic memory allocation:

C dynamic memory allocation refers to performing manual memory management for dynamic memory allocation in the C programming language via a group of functions in the C standard library, namely malloc, realloc, calloc and free.

Malloc ():

The malloc() function takes a single parameter, which is the size of the requested memory area in bytes. It returns a pointer to the allocated memory. If the allocation fails, it returns NULL. The prototype for the standard library function is like this:

```
void *malloc(size_t size);
```

Calloc():

The calloc() function does basically the same job as malloc(), except that it takes two parameters – the number of array elements and the size of each element – instead of a single parameter (which is the product of these two values). The allocated memory is also initialized to zeros. Here is the prototype:

```
void *calloc(size_t nelements, size_t elementSize);
```

Realloc():

The realloc() function resizes a memory allocation previously made by malloc(). It takes as parameters a pointer to the memory area and the new size that is required. If the size is reduced, data may be lost. If the size is increased and the function is unable to extend the existing allocation, it will automatically allocate a new memory area and copy data across. In any case, it returns a pointer to the allocated memory. Here is the prototype:

```
void *realloc(void *pointer, size_t size);
```

Free():

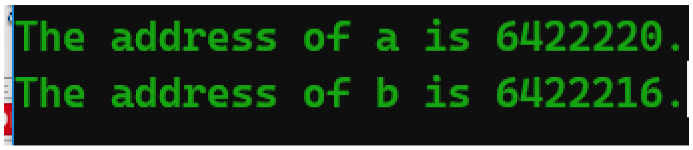
The free() function takes the pointer returned by malloc() and de-allocates the memory. No indication of success or failure is returned. The function prototype is like this:

```
        void free(void *pointer);
```

1.1Run the following program and see the output
Source Code

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

int main()
{
   int a, b;

   system("cls");

   printf("The address of a is %u.\n", &a);
   printf("The address of b is %u.", &b);
   getch();
   return 0;
}
```

Output



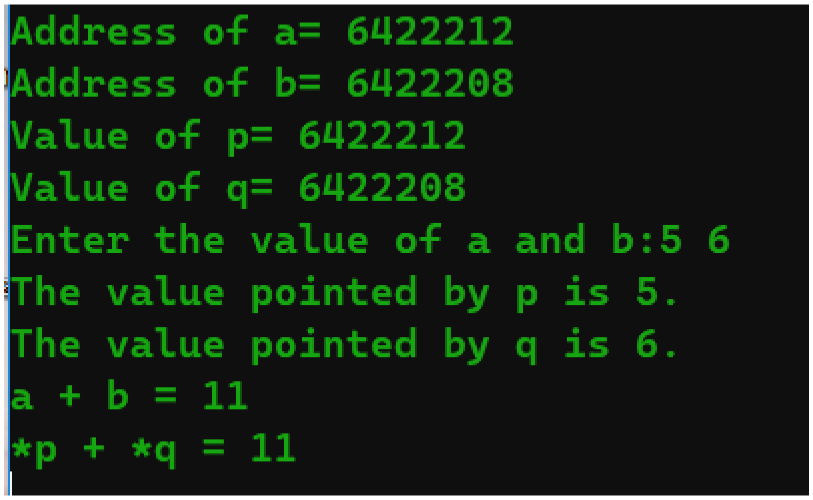
```
The address of a is 6422220.
The address of b is 6422216.
```

1.2Run the following program and see the output
Source Code

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

int main()
{
   int *p, *q; /*Declaration of pointer variable*/
   int a, b;   /*Declaration of variables*/
   p = &a;     /*Using refrencing operator to initialize pointer variable*/
   q = &b;

   system("cls");
```

```
    printf("Address of a= %u\n", &a);
    printf("Address of b= %u\n", &b);

    printf("Value of p= %u\n", p);
    printf("Value of q= %u\n", q);

    printf("Enter the value of a and b:");
    scanf("%d%d", &a, &b);

    printf("The value pointed by p is %d.\n", *p); /*USing derefrencing operator*(*)*/
    printf("The value pointed by q is %d.\n", *q);

    printf("a + b = %d\n", a + b);
    printf("*p + *q = %d\n", *p + *q);/* ----> Pointer expression*/

    getch();
    return 0;
}
```

Output



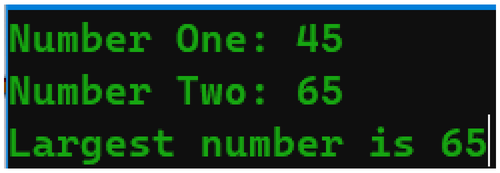
```
Address of a= 6422212
Address of b= 6422208
Value of p= 6422212
Value of q= 6422208
Enter the value of a and b:5 6
The value pointed by p is 5.
The value pointed by q is 6.
a + b = 11
*p + *q = 11
```

2.WAP to find the larger of two number using concept of function and pointer. Here pass two numbers function from main() to a function that finds the larger. Display the larger one from the main() function without return statement.
Source Code:

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

```
void largest(int a, int b, int *large)
{
   "Store the larges value in large variable";

   (a > b) ? (*large = a) : (*large = b);
}

int main()
{
   /*Variable Declaration*/
   int number_one, number_two, large;

   system("cls");

   /*Taking input from user*/
   printf("Number One: ");
   scanf("%d", &number_one);
   printf("Number Two: ");
   scanf("%d", &number_two);

   /*Function call to find largest and store it in variable*/
   largest(number_one, number_two, &large);

   /*Printing the result*/
   printf("Largest number is %d", large);

   getch();
   return 0;
}
```

Output



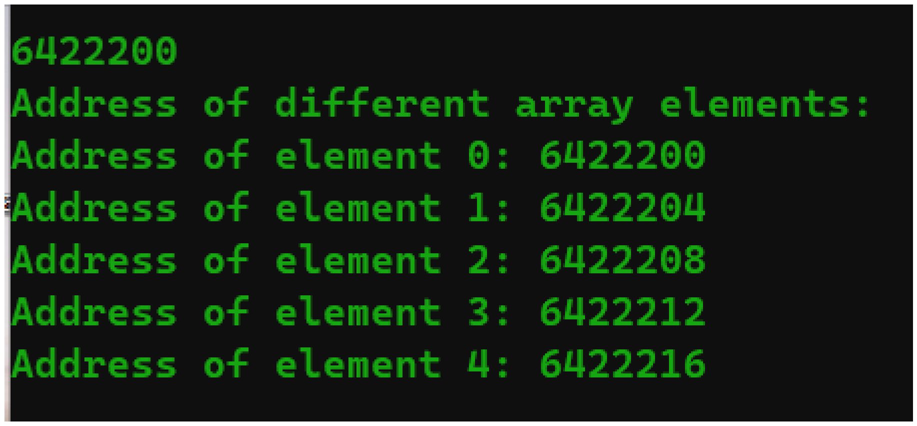
```
Number One: 45
Number Two: 65
Largest number is 65
```

3.Run the following program, observe the output and comment on that
Source Code

```
#include <stdio.h>

void main()
```

```
{
   float marks[5];
   int i;

   printf("%d\n", marks);
   printf("Address of different array elements: \n");
   for (int i = 0; i < 5; i++)
   {
      "Use either one we will obtain same result";
      /*printf("Address of element %d: %u\n", i, &marks[i]);*/
      printf("Address of element %d: %u\n", i, (marks + i));
   }
}
```

Output



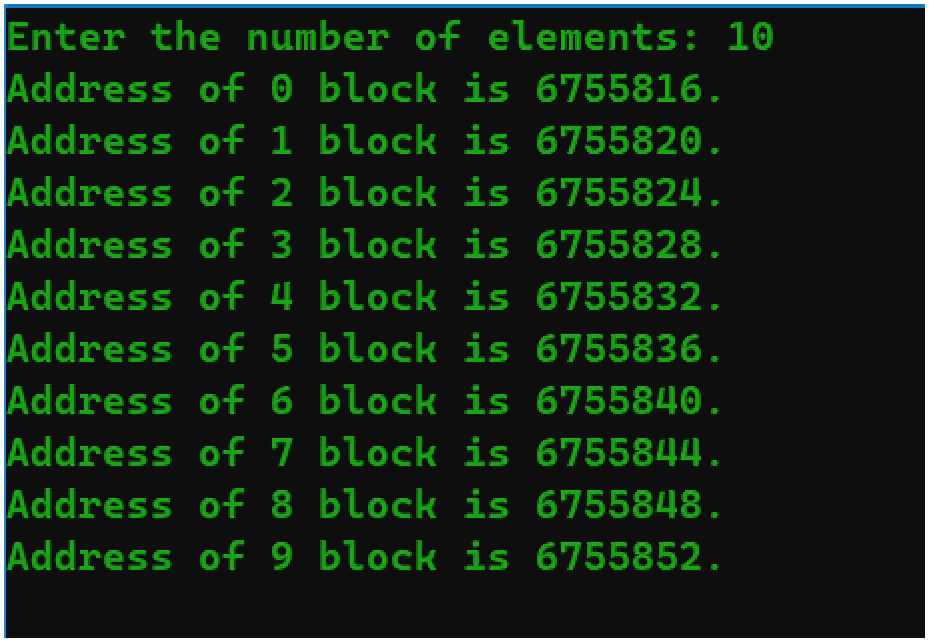
```
6422200
Address of different array elements:
Address of element 0: 6422200
Address of element 1: 6422204
Address of element 2: 6422208
Address of element 3: 6422212
Address of element 4: 6422216
```

4.This program asks the required size of array to the user and displays the address of allocated blocks
Source Code:

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

int main()
{
   /*Variable declaration*/
   int n, i;
   float *address; /*pointer variable declaration*/

   system("cls");

   /*Asking the size of array*/
```

```
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    /*Creating a array with DMA*/
    address = (float *)calloc(n, sizeof(float)); /*Using calloc function to allocate the memory for
n number of float number.*/

    /*Checking if memory is allocated or not*/
    if (address == NULL)
    {
        printf("Memory is not allocated.\n");
        exit(0); /*to exit the program if contents of address is NULL.*/
    }

    /*Printing the address of allocated blocks*/
    for (i = 0; i < n; i++)
    {
        printf("Address of %d block is %d.\n", i, (address + i));
    }

    free(address); /*To deallocate memory*/

    getch();
    return 0;
}
```

Output

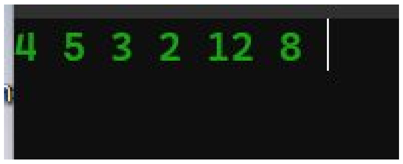


```
Enter the number of elements: 10
Address of 0 block is 6755816.
Address of 1 block is 6755820.
Address of 2 block is 6755824.
Address of 3 block is 6755828.
Address of 4 block is 6755832.
Address of 5 block is 6755836.
Address of 6 block is 6755840.
Address of 7 block is 6755844.
Address of 8 block is 6755848.
Address of 9 block is 6755852.
```

5.1
Source Code

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

void main()
{

   int i, num[6] = {4, 5, 3, 2, 12, 8};
   int *ptr;
   ptr = num;

   system("cls");

   for (int i = 0; i < 6; i++)
   {
      printf("%d ", *(ptr + i));
   }

   getch();
}
```

Output



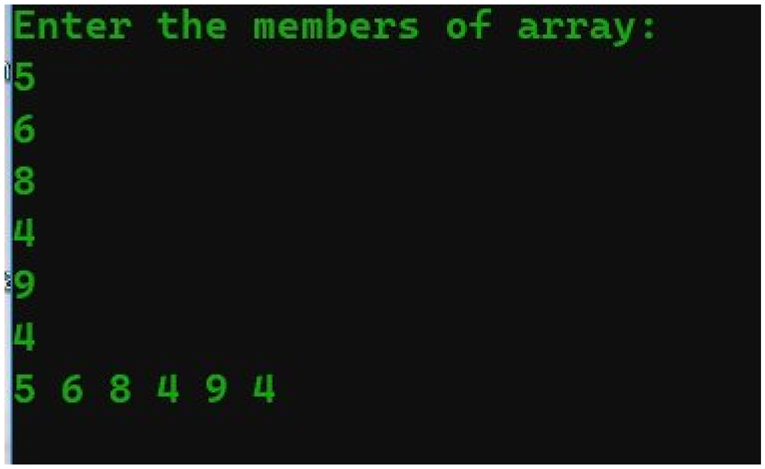
```
4 5 3 2 12 8
```

5.2
Source Code

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

int main()
{
   int i, num[6];
   int *ptr;
```

```
    ptr = num;

    system("cls");

    printf("Enter the members of array: \n");
    for (int i = 0; i < 6; i++)
    {
        scanf("%d", (ptr + i));
    }
    for (int i = 0; i < 6; i++)
    {
        printf("%d ", *(ptr + i));
    }

    getch();
    return 0;
}
```

Output



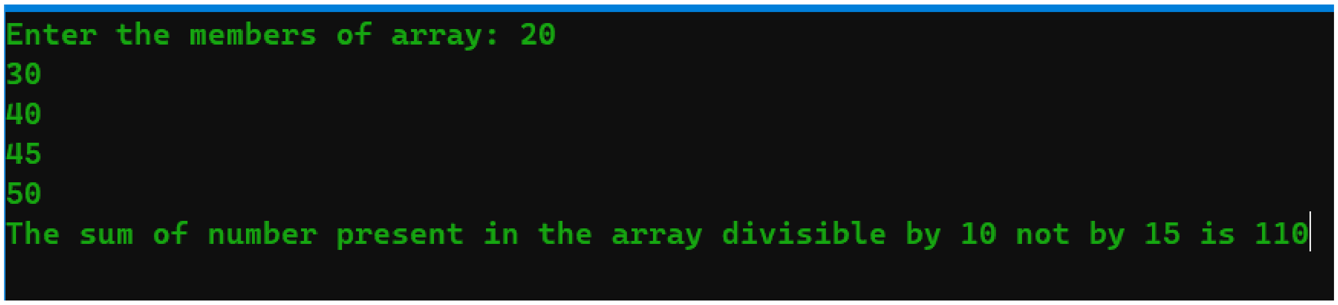
```
Enter the members of array:
5
6
8
4
9
4
5 6 8 4 9 4
```

5.3
Source Code

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

```
#include <conio.h>
#include <stdlib.h>
int main()
{
   int i, num[5], sum = 0;
   int *ptr;

   ptr = num;

   system("cls");

   printf("Enter the members of array: ");
   for (int i = 0; i < 5; i++)
   {
      scanf("%d", (ptr + i));
   }
   for (int i = 0; i < 5; i++)
   {
      (*(ptr + i) % 10 == 0 && *(ptr + i) % 15 != 0) ? (sum = *(ptr + i) + sum) : (sum =
sum);
   }

   printf("The sum of number present in the array divisible by 10 not by 15 is %d", sum);

   getch();
   return 0;
}
```

Output



```
Enter the members of array: 20
30
40
45
50
The sum of number present in the array divisible by 10 not by 15 is 110
```

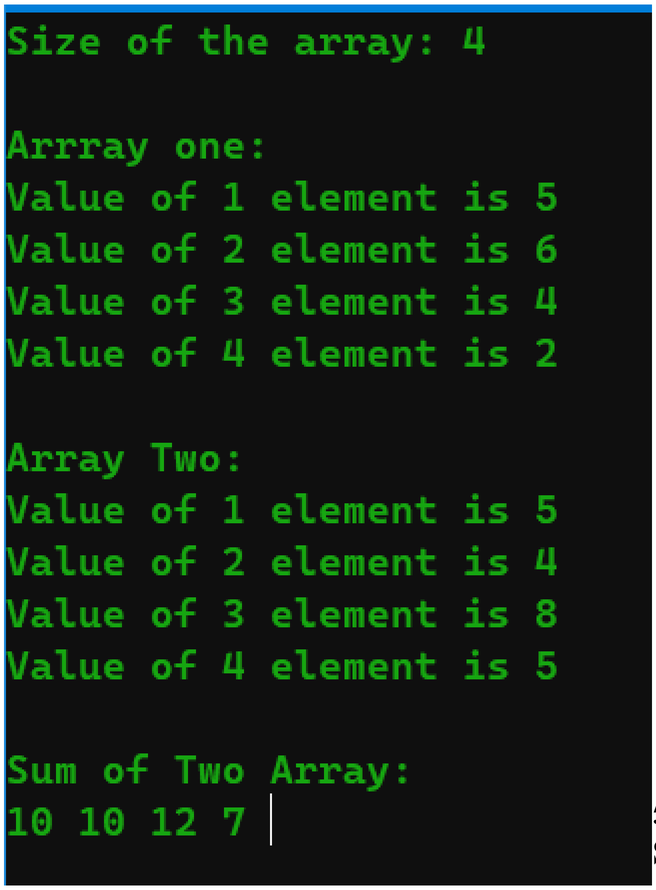
5.4Write a program to add the elements at the corresponding position of two array of size n. Read value of n from user
Source Code

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

```
#include <conio.h>
#include <stdlib.h>

int main()
{
   /*Variable and pointer declaration*/
   int number1[100], number2[100], number[100], size;
   int *one, *two, *zero;

   /*Pointer initialization*/
   one = number1;
   two = number2;
   zero = number;

   system("cls");

   printf("Size of the array: ");
   scanf("%d", &size);

   /*Taking input*/
   printf("\nArrray one:\n");
   for (int i = 0; i < size; i++)
   {
      printf("Value of %d element is ", i + 1);
      scanf("%d", (one + i));
   }

   printf("\nArray Two:\n");
   for (int i = 0; i < size; i++)
   {
      printf("Value of %d element is ", i + 1);
      scanf("%d", (two + i));
   }

   /*Calculating sum*/
   for (int i = 0; i < size; i++)
   {
       *zero = *one + *two;
      zero++;
      one++;
      two++;
   }

   //Taking pointer base address to base ko array base address
   zero -= size;
```

```
    one -= size;
    two -= size;

    /*Printing result*/
    printf("\nSum of Two Array:\n");
    for (int i = 0; i < size; i++)
    {
        printf("%d ", *(zero + i));
    }

    getch();
    return 0;
}
```

Output:

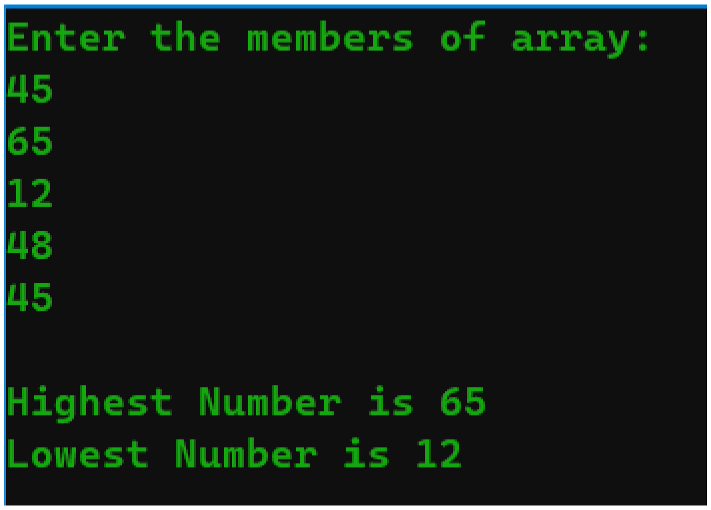


```
Size of the array: 4

Arrray one:
Value of 1 element is 5
Value of 2 element is 6
Value of 3 element is 4
Value of 4 element is 2

Array Two:
Value of 1 element is 5
Value of 2 element is 4
Value of 3 element is 8
Value of 4 element is 5

Sum of Two Array:
10 10 12 7
```

5.5
Source Code

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

int main()
{
    int i, num[5], lowest, highest;
    int *ptr;

    ptr = &num[0];

    system("cls");

    printf("Enter the members of array: \n");
    for (int i = 0; i < 5; i++)
    {
        scanf("%d", (ptr + i));
    }

    lowest = num[0];
    highest = num[0];

    for (int i = 1; i < 5; i++)
    {
        (lowest < *(ptr + i)) ? (lowest = lowest) : (lowest = *(ptr + i));
        (highest > *(ptr + i)) ? (highest = highest) : (highest = *(ptr + i));
    }

    printf("\nHighest Number is %d\n", highest);
    printf("Lowest Number is %d", lowest);

    getch();
    return 0;
}
```

Output:



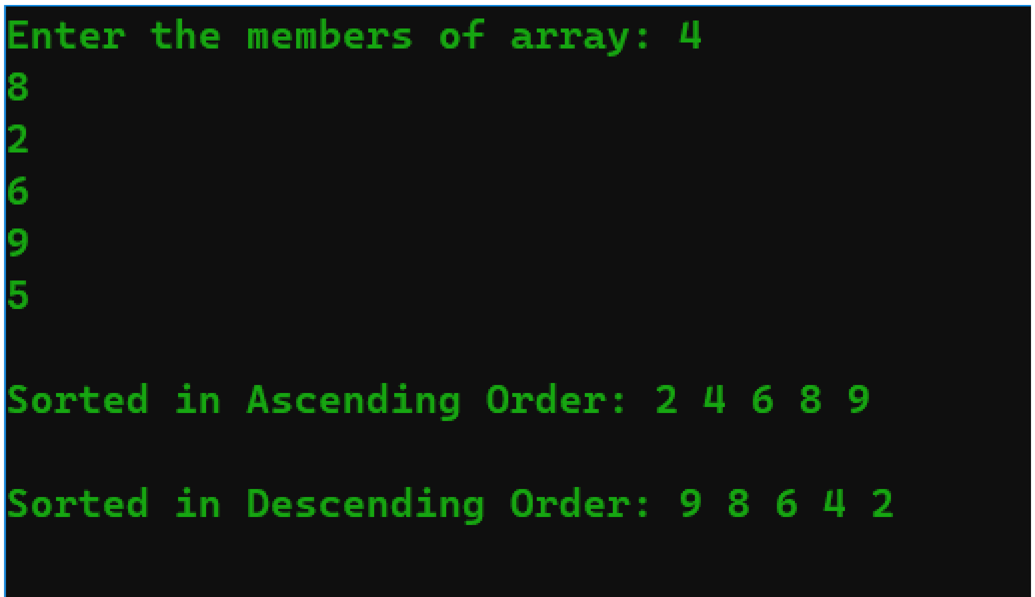
```
Enter the members of array:
45
65
12
48
45

Highest Number is 65
Lowest Number is 12
```

5.6Write a program to read the element of array in main() pass it to fucntion to sort the array in ascending/descending order. Display the sorted array from main().
Source Code

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

void sortArray(int *num, char sort[5])
{
   int temp;
   for (int i = 0; i < 5; i++)
   {
      for (int j = 0; j < 5; j++)
      {
         if (strcmp(sort, "Ascend") == 0)
         {
            if (*(num + i) > *(num + j))
            {
            }
            else
            {
               temp = *(num + i);
               *(num + i) = *(num + j);
               *(num + j) = temp;
            }
         }
         else if (strcmp(sort, "Descend") == 0)
         {
            if (*(num + i) < *(num + j))
            {
            }
            else
            {
               temp = *(num + i);
               *(num + i) = *(num + j);
               *(num + j) = temp;
            }
         }
      }
   }
}

int main()
```

```
{
   int i, num[5], lowest, highest;
   int *num_ptr;

   num_ptr = &num[0];
   system("cls");

   printf("Enter the members of array: ");

   for (int i = 0; i < 5; i++)
   {
      scanf("%d ", (num_ptr + i));
   }

   sortArray(num, "Ascend");
   printf("\nSorted in Ascending Order: ");

   for (int i = 0; i < 5; i++)
   {
      printf("%d ", *(num + i));
   }

   sortArray(num, "Descend");
   printf("\n\nSorted in Descending Order: ");

   for (int i = 0; i < 5; i++)
   {
      printf("%d ", *(num + i));
   }
   getch();
   return 0;
}
```

Output:

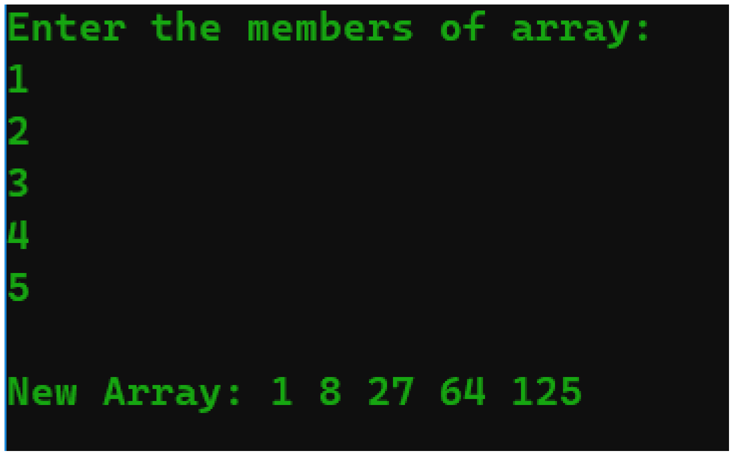


```
Enter the members of array: 4
8
2
6
9
5

Sorted in Ascending Order: 2 4 6 8 9

Sorted in Descending Order: 9 8 6 4 2
```

5.7
Source Code

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

int main()
{
   int i, num[5];
   int *num_ptr;

   num_ptr = &num[0];

   system("cls");

   printf("Enter the members of array: \n");

   for (int i = 0; i < 5; i++)
   {
      scanf("%d", (num_ptr + i));
   }

   for (int i = 0; i < 5; i++)
   {
       *(num + i) = *(num + i) * *(num + i) * *(num + i);
   }

   printf("\nNew Array: ");
   for (int i = 0; i < 5; i++)
   {
      printf("%d ", *(num + i));
   }

   getch();
   return 0;
}
```

Output:

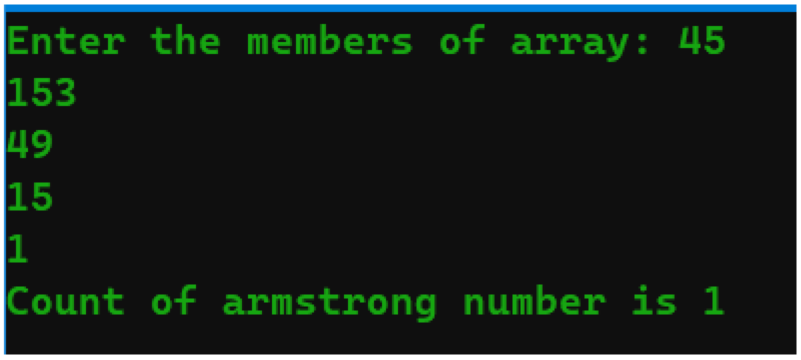


```
Enter the members of array:
1
2
3
4
5

New Array: 1 8 27 64 125
```

5.8
Source Code:

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

int armstrongArray(int *num)
{
    int count = 0, armstrong = 0, n, rem;

    for (int i = 0; i < 5; i++)
    {
        n = *(num + i);
        if (n == 0)
        {
            count++;
        }
        else
        {

            while (n != 0)
            {
                rem = n % 10;
                armstrong = n * n * n + armstrong;
                if (*(num + i) == armstrong)
                {
                    count++;
                }
                n = n / 10;
            }
            armstrong = 0;
        }
    }

    return count;
}

int main()
{
    unsigned int num[5];
    int *num_ptr;

    num_ptr = &num[0];
    system("cls");

    printf("Enter the members of array: ");

    for (int i = 0; i < 5; i++)
    {
        scanf("%d", (num_ptr + i));
    }

    printf("Count of armstrong number is %d", armstrongArray(num));
    getch();
    return 0;
}
```

Output:



```
Enter the members of array: 45
153
49
15
1
Count of armstrong number is 1
```

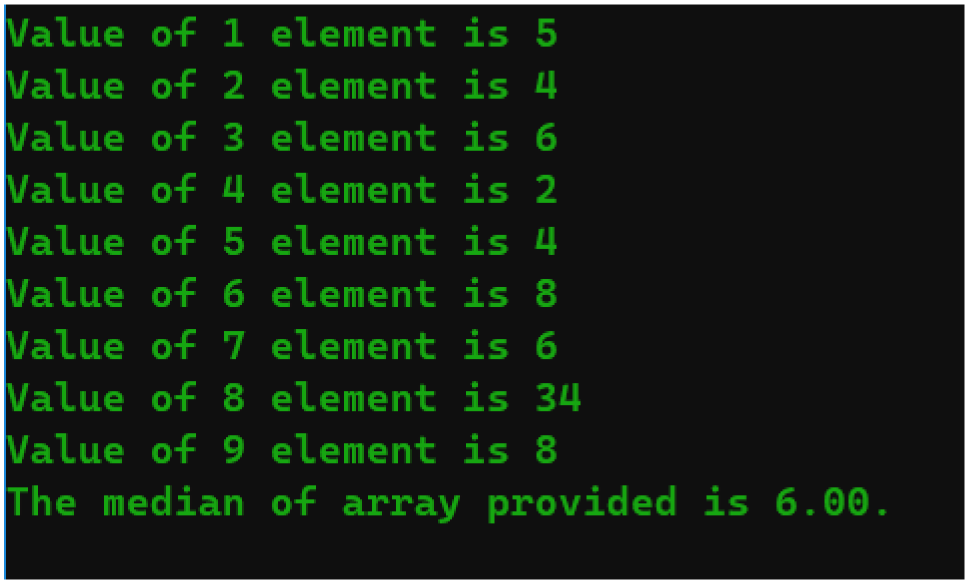
5.9.1
Source Code

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

```
#include <conio.h>
#include <stdlib.h>

#define ARRAY_LENGTH 9

int main()
{
   int median[ARRAY_LENGTH];
   int *ptr_median;
   float result;

   ptr_median = &median[0];

   system("cls");

   for (int i = 0; i < ARRAY_LENGTH; i++)
   {
      printf("Value of %d element is ", i + 1);
      scanf("%d", (ptr_median + i));
   }

   for (int i = 0; i < ARRAY_LENGTH; i++)
   {
      for (int j = 0; j < ARRAY_LENGTH; j++)
      {
         if (*(ptr_median + i) < *(ptr_median + j))
         {
            *(ptr_median + i) = *(ptr_median + i) + *(ptr_median + j) - (*(ptr_median + j) =
*(ptr_median + i));
         }
      }
   }

   float position_of_array = (ARRAY_LENGTH + 1) / 2.0;

   if (position_of_array - (int)position_of_array == 0.0)
   {
      result = *(ptr_median + (int)position_of_array - 1);
   }
   else
   {
      result = *(ptr_median + (int)position_of_array) + *(ptr_median + (int)position_of_array
- 1) / 2.0;
   }
   printf("The median of array provided is %.2f.", result);
```

```
    getch();
    return 0;
}
```

Output:

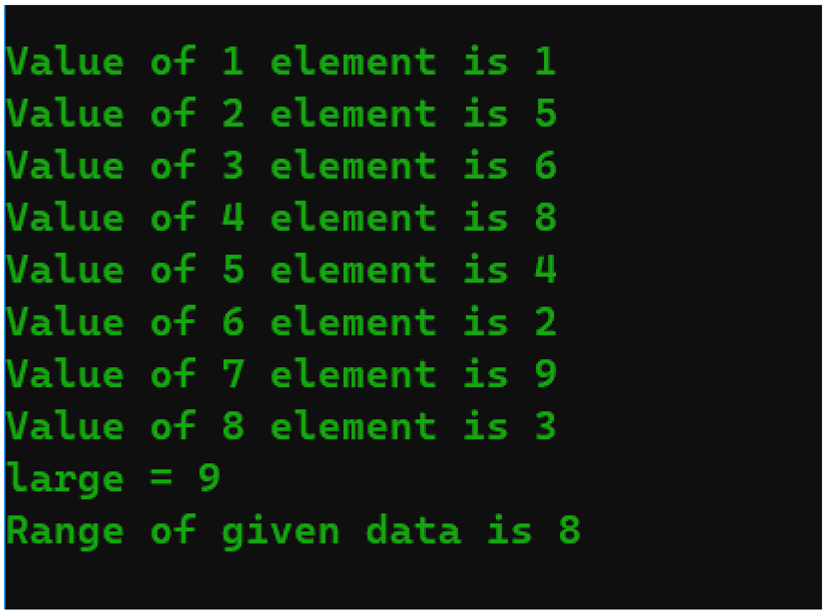


```
Value of 1 element is 5
Value of 2 element is 4
Value of 3 element is 6
Value of 4 element is 2
Value of 5 element is 4
Value of 6 element is 8
Value of 7 element is 6
Value of 8 element is 34
Value of 9 element is 8
The median of array provided is 6.00.
```

5.9.2Program to compute range of one d array
Source Code

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

#define ARRAY_LENGTH 8

int main()
{
    int input[ARRAY_LENGTH], small, large, range;
    int *ptr_input;

    ptr_input = &input[0];

    for (int i = 0; i < ARRAY_LENGTH; i++)
    {
        printf("Value of %d element is ", i + 1);
        scanf("%d", (ptr_input + i));
    }

    small = *(ptr_input);
    large = *(ptr_input);

    for (int i = 1; i < ARRAY_LENGTH; i++)
    {
        if (small > *(ptr_input + i))
        {
            small = *(ptr_input + i);
            printf("small = %d\n", small);
        }
    }
    for (int i = 1; i < ARRAY_LENGTH; i++)
    {
        if (large < *(ptr_input + i))
        {
            large = *(ptr_input + i);
        }
    }
    printf("large = %d\n", large);

    range = (large - small);

    printf("Range of given data is %d", range);

    getch();
    return 0;
}
```

Output:



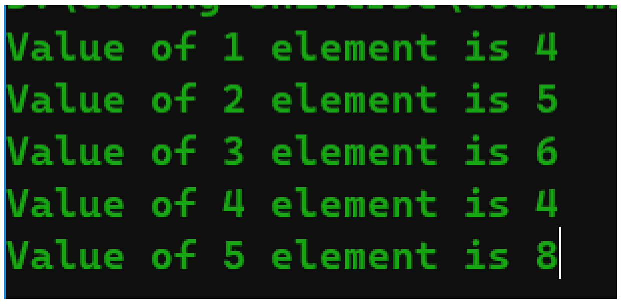
```
Value of 1 element is 1
Value of 2 element is 5
Value of 3 element is 6
Value of 4 element is 8
Value of 5 element is 4
Value of 6 element is 2
Value of 7 element is 9
Value of 8 element is 3
large = 9
Range of given data is 8
```

5.9.3
Source Code

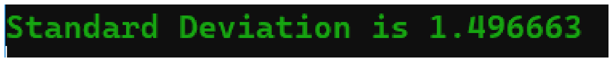
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>

#define ARRAY_LENGTH 5

int main()
{
   int input[ARRAY_LENGTH], *ptr_input, sum = 0;
   float sd, diverse, mean = 0.0;

   ptr_input = &input[0];

   for (int i = 0; i < ARRAY_LENGTH; i++)
   {
      printf("Value of %d element is ", i + 1);
      scanf("%d", (ptr_input + i));
   }

   system("cls");

   for (int i = 0; i < ARRAY_LENGTH; i++)
   {
      sum = sum + *(ptr_input + i);
   }
   mean = sum / (float)ARRAY_LENGTH;

   for (int i = 0; i < ARRAY_LENGTH; i++)
   {
      diverse = (mean - *(ptr_input + i)) * (mean - *(ptr_input + i)) + diverse;
   }

   sd = pow(diverse / ARRAY_LENGTH, 0.5);

   printf("Standard Deviation is %f\n", sd);
```

```
    getch();
    return 0;
}
```

Output:



```
Value of 1 element is 4
Value of 2 element is 5
Value of 3 element is 6
Value of 4 element is 4
Value of 5 element is 8
```

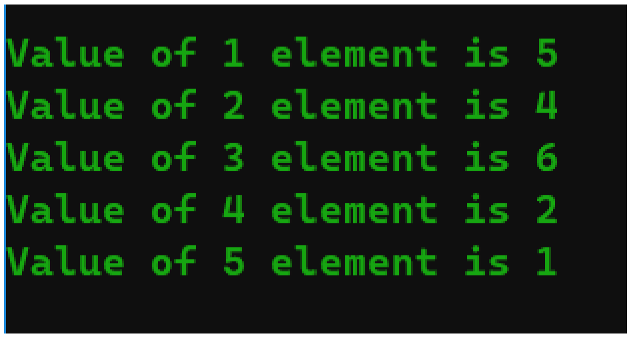


```
Standard Deviation is 1.496663
```

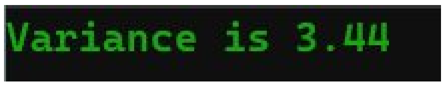
5.9.4
Source Code

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

#define ARRAY_LENGTH 5

int main()
{
    float variance;
    int input[10];
    float mean = 0.0;
    int sum = 0;

    int *ptr_input;

    ptr_input = &input[0];

    for (int i = 0; i < ARRAY_LENGTH; i++)
    {
        printf("Value of %d element is ", i + 1);
        scanf("%d", (ptr_input + i));
    }
    system("cls");
    for (int i = 0; i < ARRAY_LENGTH; i++)
    {
        sum = sum + *(ptr_input + i);
    }
    mean = sum / (float)ARRAY_LENGTH;

    for (int i = 0; i < ARRAY_LENGTH; i++)
    {
        variance = (mean - *(ptr_input + i)) * (mean - *(ptr_input + i)) + variance;
    }

    variance = variance / ARRAY_LENGTH;
    printf("Variance is %.2f", variance);

    getch();
    return 0;
}
```

Output



```
Value of 1 element is 5
Value of 2 element is 4
Value of 3 element is 6
Value of 4 element is 2
Value of 5 element is 1
```



```
Variance is 3.44
```

ANALYSIS AND DISCUSSION:

In exercise 9 of C programming, we understood about pointers, C dynamic memory allocation and their importance. The usage of pointers and functions like malloc, calloc, realloc and free were also discussed.

CONCLUSION:

Hence, the focused objective to understand the importance of pointers and dynamic memory allocation was achieved successfully.