TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

Subject: C-Programming

Experiment Number: 9

Title: **POINTERS AND DYNAMIC MEMORY ALLOCATION**

Date: 14th August,2021

Submitted By:	Submitted To:
Name: Susheel Thapa	Department of Electronics and Computer Engineering
Program: BCT	
Section: CD	Checked By:
Roll: 077BCT090	

POINTERS AND DYNAMIC MEMORY ALLOCATION

BACKGROUND THEORY

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;

1. A pointer variable can be assigned the address of an ordinary variable(eg. p=&a)
2. 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)
3. An integer quantity can be added to or subtracted from a pointer variable. (eg. p+5, q-1, q++, --p)
4. A pointer variable can be assigned a null value. (eg. p=NULL)
5. One pointer variable can be subtracted from another provided both pointer point to elements of same array.
6. 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:

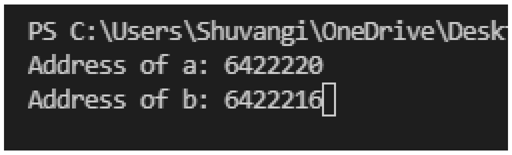
```
void free(void *pointer);
```

1.Run the program, observe the output and comment on that.

SOURCE CODE:

```
#include <stdio.h>

#include <conio.h>

void main()

{

   int a,b;

   printf("Address of a: %u",&a);

   printf("\nAddress of b: %u",&b);

   getch();

}
```

OUTPUT:

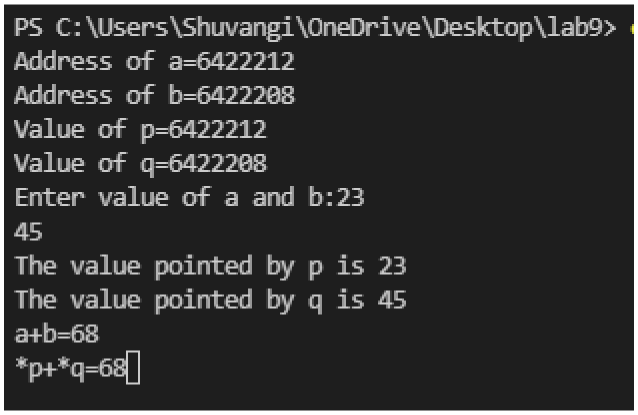


```
PS C:\Users\Shuvangi\OneDrive\Desk
Address of a: 6422220
Address of b: 6422216
```

SOURCE CODE:

```
#include <stdio.h>

#include <conio.h>

void main()

{

    int *p,*q;

    int a,b;

    p=&a;

    q=&b;

    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 value of a and b:");

    scanf("%d%d",&a,&b);

    printf("The value pointed by p is %d\n",*p);

    printf("The value pointed by q is %d\n",*q);

    printf("a+b=%d\n",a+b);

    printf("*p+*q=%d",*p+*q);

    getch();

    }
```

OUTPUT:



```
PS C:\Users\Shuvangi\OneDrive\Desktop\lab9>
Address of a=6422212
Address of b=6422208
Value of p=6422212
Value of q=6422208
Enter value of a and b:23
45
The value pointed by p is 23
The value pointed by q is 45
a+b=68
*p+*q=68
```

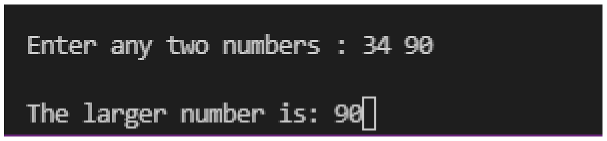
2.WAP to find larger of two numbers using function and pointer. Here pass two numbers from main() to a function that finds the larger. Display the larger one from main() without using return statement.

SOURCE CODE:

```
#include <stdio.h>

#include <conio.h>

void largest(int *,int *);

int main()

{

  int a,b;

  printf("\nEnter any two numbers : ");

  scanf("%d%d",&a,&b);

  largest(&a,&b);

  printf("\nThe larger number is: %d",a);

  getch();

  return 0;

}
```

```
void largest(int *p,int*q){
    if (*p<*q)
    *p=*q;
}
```

OUTPUT:



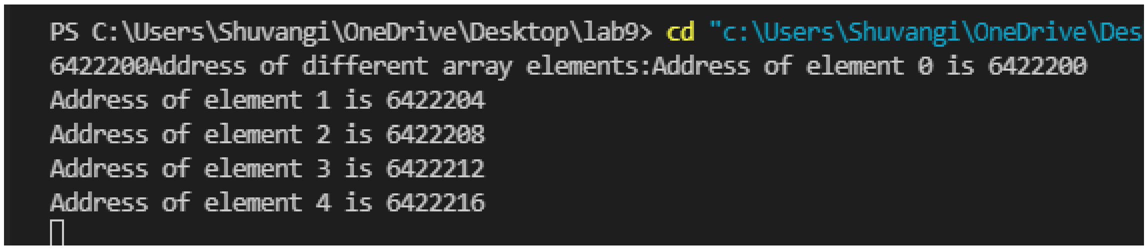
```
Enter any two numbers : 34 90

The larger number is: 90
```

3. **Run the program and observe the output:**

SOURCE CODE:

```
#include <stdio.h>
#include <conio.h>
void main()
{
   float marks[5];
   int i;
   printf("%d",marks);
   printf("Address of different array elements:");
   for (i=0;i<5;i++)
   printf("Address of element %d is %u\n",i,&marks[i]);
   getch();
}
```

OUTPUT:



```
PS C:\Users\Shuvangi\OneDrive\Desktop\lab9> cd "c:\Users\Shuvangi\OneDrive\Des
6422200Address of different array elements:Address of element 0 is 6422200
Address of element 1 is 6422204
Address of element 2 is 6422208
Address of element 3 is 6422212
Address of element 4 is 6422216
```

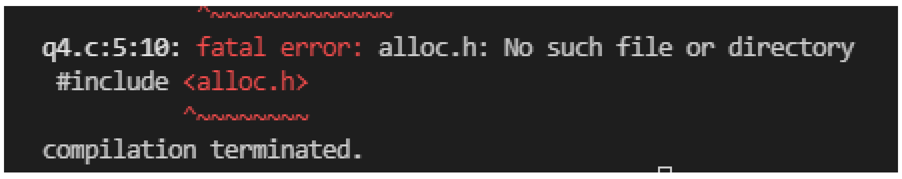
4. This program asks the required size of array to the user and displays the addresses of allocated blocks.

SOURCE CODE:

```
#include <stdio.h>

#include <alloc.h>


void main()

{

   int n,i;

   float *address;

   printf("Enter the number of elements:")

   scanf("%d",&n);

   address=(float *)calloc(n,sizeof(float));

   if (address=NULL)

   {

      printf("Memory can not be allocated");

      exit();

   }
```

```
for (i=0;i<n;i++){
    printf("\nAddress of %d block %d",i,(address+i));
}
free(address);
```

OUTPUT:



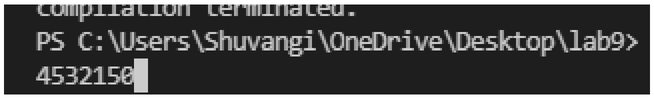
```
q4.c:5:10: fatal error: alloc.h: No such file or directory
 #include <alloc.h>
          ^~~~~~~~~
compilation terminated.
```

5.Solve all the problems of one dimensional array of exercise eight using the concept of dynamic memory allocation.

1.Source code:

```
#include <stdio.h>

#include <conio.h>

void main()

{

        int i, num[6]={4,5,3,2,15};

        for (i=0;i<6;i++){

        printf("%d",*(num+i));

        }

        getch();

}
```

OUTPUT:

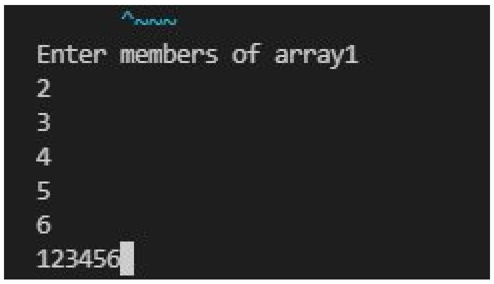


```
compilation terminated.
PS C:\Users\Shuvangi\OneDrive\Desktop\lab9>
4532150
```

2. SOURCE CODE:

```
#include <stdio.h>

#include <conio.h>

void main()

{

   int i,num[6];

   printf("Enter members of array");

   for (i=0;i<6;i++)

   scanf("%d",(num+i)); //doesn't need ampersand

   for (i=0;i<6;i++)

   printf("%d",*(num+i));

   getch();

   return 0;

}
```

OUTPUT:

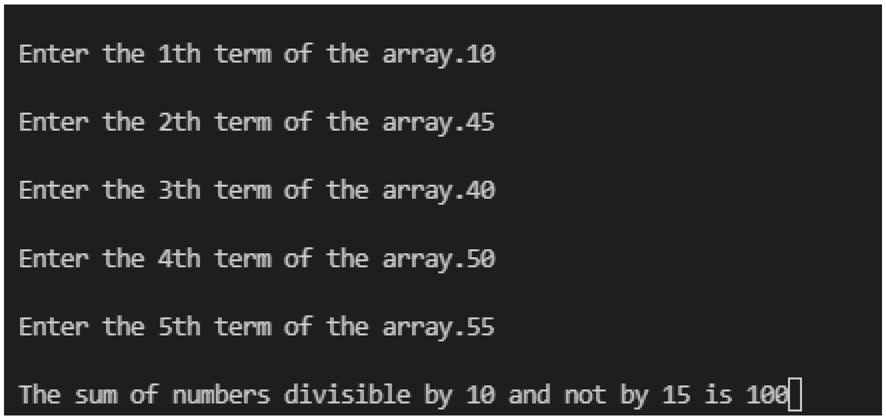


```
        ^~~~
Enter members of array1
2
3
4
5
6
123456
```

3. SOURCE CODE:

```
#include <stdio.h>
#include <conio.h>
int main(){
    int i,arr[5],sum=0;
    for (i=0;i<5;i++){
        printf("\nEnter the %dth term of the array.",i+1);
        scanf("%d",(arr+i));
    }
    for (i=0;i<5;i++){
        if ((*(arr+i)%10==0) && (*(arr+i)%15!=0)){
            sum+=*(arr+i);
        }
    }
    printf("\nThe sum of numbers divisible by 10 and not by 15 is %d",sum);
    getch();
    return 0;
}
```

OUTPUT:

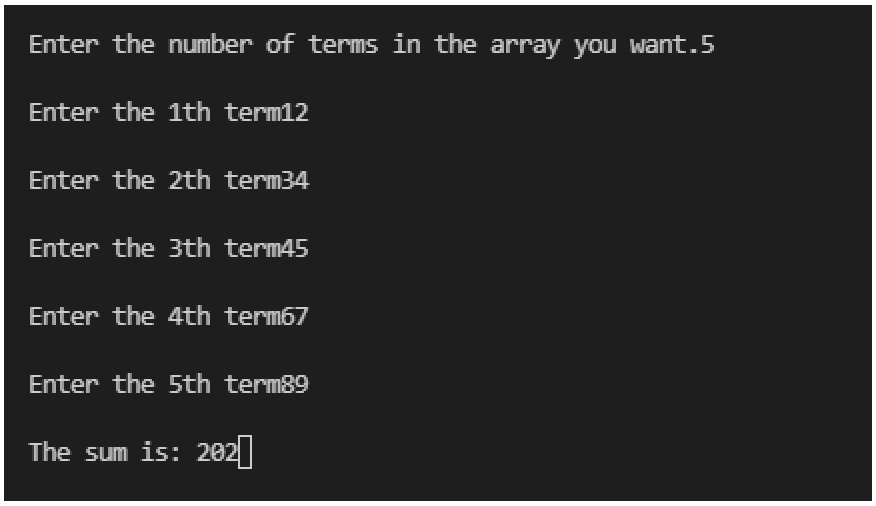


```
Enter the 1th term of the array.10

Enter the 2th term of the array.45

Enter the 3th term of the array.40

Enter the 4th term of the array.50

Enter the 5th term of the array.55

The sum of numbers divisible by 10 and not by 15 is 100
```

4. SOURCE CODE:

```
#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

int main()

{

    int *arr,n,sum=0; //setting up a pointer from where to allocate

    printf("\nEnter the number of terms in the array you want.");

    scanf("%d",&n);

    arr=(int*)malloc(n*sizeof(int));

    for (int i=0;i<n;i++){

        printf("\nEnter the %dth term",i+1);

        scanf("%d",arr+i); // no need to use ampersand since its address anyway

    }

    //to add corresponding terms;

    for (int i=0;i<(n/2);i++){ //running the loop upto midpoint or near midpoint

    sum+=(*(arr+i)+*(arr+n-i-1));


    }

    printf("\nThe sum is: %d",sum);

    getch();

    return 0;

}
```

OUTPUT:

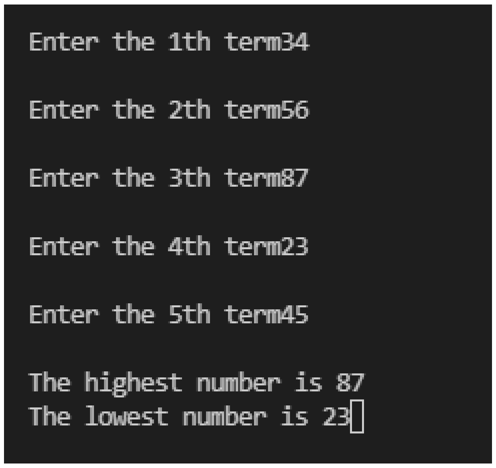

```
Enter the number of terms in the array you want.5

Enter the 1th term12

Enter the 2th term34

Enter the 3th term45

Enter the 4th term67

Enter the 5th term89

The sum is: 202
```

5. SOURCE CODE:

```
#include <stdio.h>

#include <conio.h>

int main()

{

  int arr[5],i,high,low;

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

     printf("\nEnter the %dth term",i+1);

     scanf("%d",arr+i);

  }

  high=*arr;

  low=*arr;

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

     if (*(arr+i)>high)

     high=*(arr+i);
```

```
        if (*(arr+i)<low)

        low=*(arr+i);

    }

    printf("\nThe highest number is %d\nThe lowest number is %d",high,low);



    getch();

    return 0;

}
```

OUTPUT:

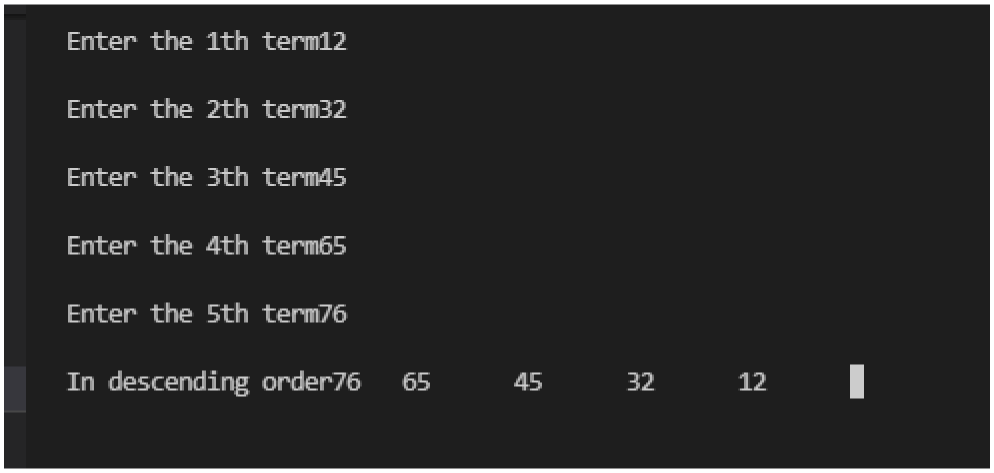


```
Enter the 1th term34

Enter the 2th term56

Enter the 3th term87

Enter the 4th term23

Enter the 5th term45

The highest number is 87
The lowest number is 23
```

6. SOURCE CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

int func(int *arr,int n){

        int temp;

        for (int i=0;i<n;i++)

        for(int j=i+1;j<n;j++){


        if (*(arr+j)>*(arr+i)){

        temp=*(arr+i);

        *(arr+i)=*(arr+j);

        *(arr+j)=temp;


        }

        }

}

int main()

{

        int *arr,n;

        printf("\nEnter the number of terms that you want to enter.");

        scanf("%d",&n);

        arr=(int*)malloc(n*sizeof(int));

        for (int i=0;i<n;i++){

        printf("\nEnter the %dth term",i+1);

        scanf("%d",arr+i);
        }
        func(arr,n);
        printf("\nIn descending order");
        for (int i=0;i<n;i++){
        printf("%d\t",*(arr+i));
        }
        getch();
        return 0;
}
```

OUTPUT:

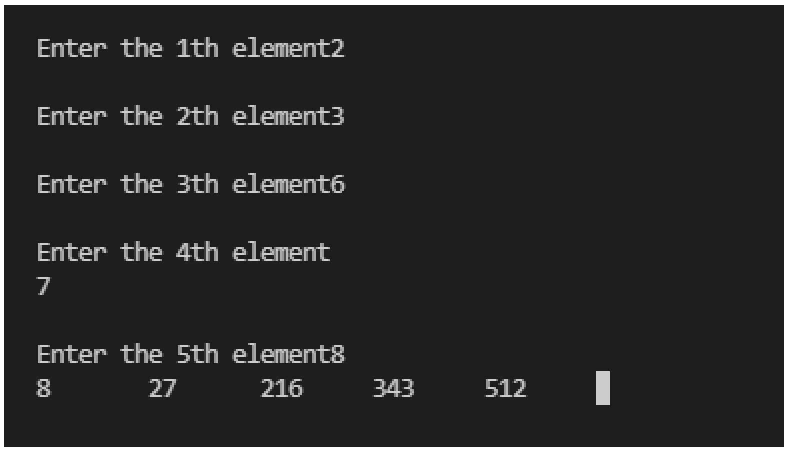


```
Enter the 1th term12

Enter the 2th term32

Enter the 3th term45

Enter the 4th term65

Enter the 5th term76

In descending order76   65      45      32      12
```

7. SOURCE CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <math.h>

int main()

{

   int *arr,n;

   printf("\nEnter the number of terms you want to add");

   scanf("%d",&n);

   arr=(int*)malloc(n*sizeof(int));

   for (int i=0;i<n;i++){

       printf("\nEnter the %dth element",i+1);

       scanf("%d",arr+i);

   }

for (int i=0;i<n;i++){

   *(arr+i)=pow(*(arr+i),3);

   printf("%d\t",*(arr+i));


}

   getch();

   return 0;

}
```

OUTPUT:



```
Enter the 1th element2

Enter the 2th element3

Enter the 3th element6

Enter the 4th element
7

Enter the 5th element8
8       27      216     343     512
```

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.