CL1002 Programming Fundamentals

LAB 10 & 11
Nested Structure, Introduction to Pointers, Accessing Arrays using Pointer, Dynamic Memory Management

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Learning Objectives

1. Structures
2. Nested Structures
3. Introduction to Pointers
4. Accessing Array using Pointers
5. Dynamic Memory

1. Structures

Structures are derived data types—they're constructed using objects of other types. Normally, we use structure to store the record or the details of any item or entity. Structure members can be variables of the primitive data types (e.g., int, float, etc.), or aggregates, such as arrays and other structures.

- Keyword struct introduces a structure definition
- The identifier Chocolate is the structure tag, which names the structure definition and is used with struct to declare variables of the structure type—e.g., struct Chocolate kitkat, Mars, Jubilee.
- Variables declared within the braces of the structure definition are the structure's members.
- Members of the same structure type must have unique names, but two different structure types may contain members of the same name without conflict.

1.1   Declaration of Struct

```
struct Choclate{
        char Name[20];
        float Weight;
        int Calories;
        float Price;
        char ExpiryDate[10];
};
```

1.2   Declaration & Initialization of Struct type Variables

You can declare the variables before the semi-colon(;) or using a proper declaration syntax like other variable's in main();

```
struct Choclate{
        char Name[20];
        float Weight;
        int Calories;
        float Price;
        char ExpiryDate[10];
}var1, var2,var3;
```

```
Int main()
{
Struct Choclate Kitkat, Mars, Jubliee, mychoclate[3];
//      OR
struct Choclate myChoclate;
      gets(myChoclate.Name);
      myChoclate.Weight= 20;
      myChoclate.Calories= 500;
      myChoclate.Price= 100;
      strcpy(myChoclate.ExpiryDate,"01-Feb-2021");

//      OR

struct Choclate Jubliee = {"Jubilee",20.50,500,100,"01-Feb-2021"};

}
```

1.3   Declaration & Initialization of Struct type Array

```
Int main()
{
// Array of Struct
struct Choclate myFavChoclates[3];    // It is an array of sruct
int i = 0;
   while(i<3)
        {
            gets(myChoclate[i].Name);
            scanf("%f",&myChoclate.Weight);
            scanf("%d",&myChoclate.Calories);
             scanf("%f",&myChoclate.Price);
            gets(myChoclate[i].ExpiryDate);

            ++i;
        }

// TO print this array of Struct

 i = 0;
   while(i<3)
        {
            puts(myChoclate[i].Name);
            printf("%f",myChoclate.Weight);
            printf("%d",myChoclate.Calories);
             printf("%f",myChoclate.Price);
            puts(myChoclate[i].ExpiryDate);

            ++i;
        }
}
```

2.0 Nested Structures

Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure. The structure variables can be a normal structure variable ,array or a pointer variable to access the data. You can learn below concepts in this section.

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

struct UniversityDetails
{
    int UniversityRanking;
    char UniversityName[90];
};

struct student_detail
{
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct UniversityDetails data;
};

int main()
{
    struct student_detail std_data = {1, "Arif", 80.5, 285,
                                "National University of Computer & Emerging Sciences"};
    printf(" Id is: %d \n", std_data.id);
    printf(" Name is: %s \n", std_data.name);
    printf(" Percentage is: %f \n\n", std_data.percentage);

    printf(" University Ranking is: %d \n",
                std_data.data.UniversityRanking);
    printf(" University Name is: %s \n",
                std_data.data.UniversityName);
    return 0;
}
```
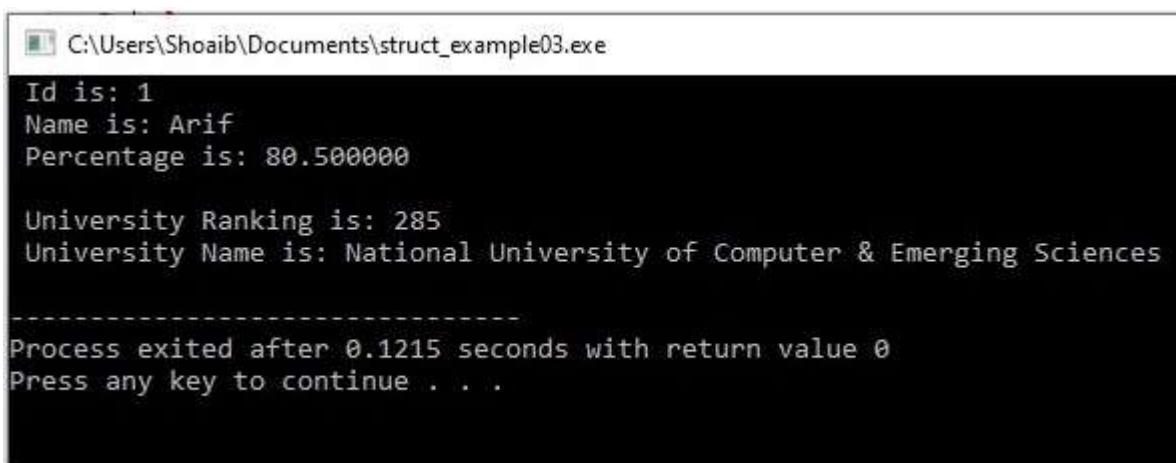
OUTPUT:

```
 C:\Users\Shoaib\Documents\struct_example03.exe

Id is: 1
Name is: Arif
Percentage is: 80.500000

University Ranking is: 285
University Name is: National University of Computer & Emerging Sciences

--------------------------------
Process exited after 0.1215 seconds with return value 0
Press any key to continue . . .
```

Another example of Nested Structure:

Sample Code:

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

struct Type{
    char TypeName[20];          // Mini, Sedan, Sports, Luxary, SUV
};

struct Car{
    char CarName[20];
    char make[15];
    char model[15];
    char color[10];
    int seats;
    int engine;          // 1800 cc
    int price;

    struct Type CarType;
};

int main()
{
    struct Car myCar;

    puts("----------------------- Example: Nested Structure -----------------------");

    puts("Enter the Name of your Car: ");
    gets(myCar.CarName);
    puts("Enter the type of your Car {Mini, Sedan, Sports, Luxary, SUV}: ");
    gets(myCar.CarType.TypeName);

    puts("Enter the Color of your Car: ");
    gets(myCar.color);
    puts("Enter the make of your Car: ");
    gets(myCar.make);
    puts("Enter the model of your Car: ");
    gets(myCar.model);
    printf("\nEnter the seats of your Car: ");
    scanf("%d",&myCar.seats);
    printf("\nEnter the engine cpacity (cc) of your Car: ");
    scanf("%d",&myCar.engine);
    printf("\nEnter the price of your Car: ");
    scanf("%d",&myCar.price);

    puts("\n\n----------------------- Print  -----------------------");

    printf("\nCarName: %s",myCar.CarName);
    printf("\nCarType: %s",myCar.CarType.TypeName);
    printf("\nColor: %s",myCar.color);
    printf("\nMake: %s",myCar.make);
    printf("\nModel: %s",myCar.model);
    printf("\nSeats: %d",myCar.seats);
    printf("\nEngine (cc): %d",myCar.engine);
    printf("\nPrice: %d", myCar.price);


    return 0;
```

OUTPUT:

C:\Users\Shoaib\Documents\struct_example02.exe

```
------------------------ Example: Nested Structure ------------------------
Enter the Name of your Car:
Picanto 2.0
Enter the type of your Car {Mini, Sedan, Sports, Luxary, SUV}:
Mini
Enter the Color of your Car:
White
Enter the make of your Car:
KIA
Enter the model of your Car:
Picanto

Enter the seats of your Car: 4

Enter the engine cpacity (cc) of your Car: 1300

Enter the price of your Car: 120000



------------------------ Print ------------------------

CarName: Picanto 2.0
CarType: Mini
Color: White
Make: KIA
Model: Picanto
Seats: 4
Engine (cc): 1300
Price: 120000
--------------------------------
Process exited after 54.59 seconds with return value 0
Press any key to continue . . .
```
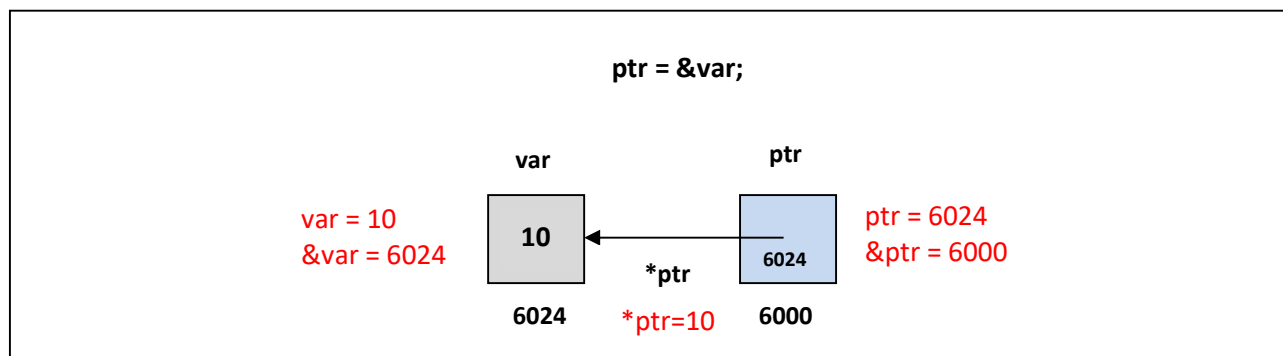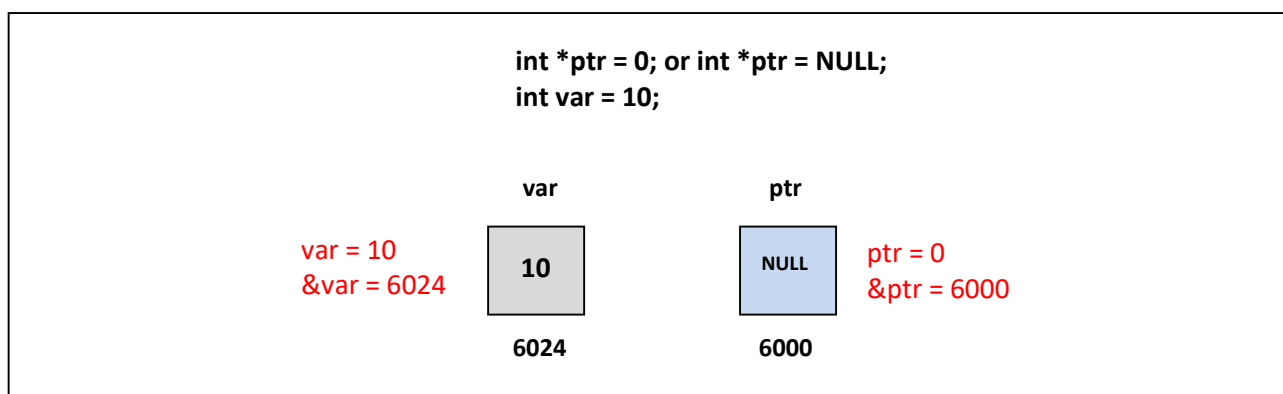
3.0 Introduction to Pointers

Pointer is a variable whose value is a memory address. Normally, a variable directly contains a specific value. A pointer contains the memory address of a variable that, in turn, contains a specific value. In this sense, a variable name directly references a value, and a pointer indirectly references a value.

3.1 Pointer Declaration & Initialization

Syntax: type * variable;

Code: int *ptr = 0; // Pointer Declaration
                int var= 10;
                ptr = &var;   // Pointer Initialization

The value of the pointer variable ptr is a memory address. A data item whose address is stored in this variable must be of the specified type.

---

int *ptr = 0; or int *ptr = NULL;
int var = 10;

	var		ptr	
var = 10 &var = 6024	**10** 6024		**NULL** 6000	ptr = 0 &ptr = 6000

---

ptr = &var;

	var		ptr	
var = 10 &var = 6024	**10** 6024	← *ptr *ptr=10	**6024** 6000	ptr = 6024 &ptr = 6000

Sample Code:
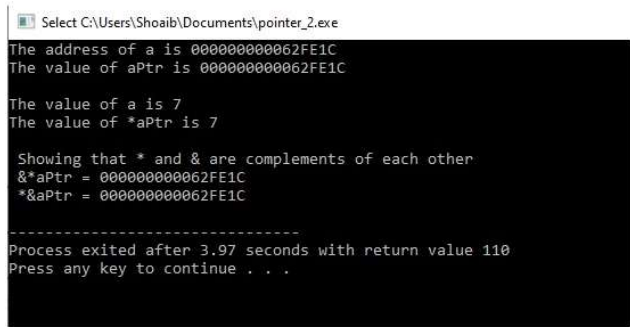
```c
#include <stdio.h>

int main()
{
int a; // a is an integer
int *aPtr; // aPtr is a pointer to an integer

a = 7;
aPtr = &a; // set aPtr to the address of a

printf( "The address of a is %p \nThe value of aPtr is %p", &a, aPtr);

printf( "\n\nThe value of a is %d \nThe value of *aPtr is %d", a, *aPtr);

printf( "\n\n Showing that * and & are complements of each other \n &*aPtr = %p \n *&aPtr = %p\n", &*aPtr,*&aPtr );
} // end main
```

```
Select C:\Users\Shoaib\Documents\pointer_2.exe
The address of a is 000000000062FE1C
The value of aPtr is 000000000062FE1C

The value of a is 7
The value of *aPtr is 7

 Showing that * and & are complements of each other
 &*aPtr = 000000000062FE1C
 *&aPtr = 000000000062FE1C

------------------------------
Process exited after 3.97 seconds with return value 110
Press any key to continue . . .
```

3.2 POINTER ARITHMETICS

- A limited set of arithmetic operations may be performed on pointers. A pointer may be incremented (++) or decremented (--), an integer may be added to a pointer (+ or +=), an integer may be subtracted from a pointer (- or -=) and one pointer may be subtracted from another.
- When an integer is added to or subtracted from a pointer, the pointer is incremented or decremented by that integer times the size of the object to which the pointer refers.
- Two pointers to elements of the same array may be subtracted from one another to determine the number of elements between them.

4.0 Accessing Array using Pointers

Arrays and pointers are intimately related in C and often may be used interchangeably.
- An array name can be thought of as a constant pointer.
- Pointers can be used to do any operation involving array subscripting.
- When a pointer points to the beginning of an array, adding an offset to the pointer indicates which element of the array should be referenced, and the offset value is identical to the array subscript. This is referred to as pointer/offset notation.
- An array name can be treated as a pointer and used in pointer arithmetic expressions that do not attempt to modify the address of the pointer.
- Pointers can be subscripted exactly as arrays can. This is referred to as pointer/subscript notation.

Sample Code:

```c
#include <stdio.h>

int main()
{
        int *ptr = NULL; // pointer variable of type "pointer to int" / null pointer
        int intVariable1 = 10; // Declare an integer variable and initialize it with 10

        // Use address-of operator & to assign memory address of intVariable1 to a pointer
        ptr = &intVariable1;
        // Pointer ptr now holds a memory address of intVariable

        // Print out associated memory addresses and their values
        printf("The memory address allocated to ptr at the time of its creation:%d \n",&ptr);
        printf("\nptr is pointing to memory address or value contained in ptr:%d\n",ptr);
        printf("\nThe memory address allocated to intVariable at the time of its creation:%d\n",&intVariable1);
        printf("The value contained in intVariable:%d\n" ,intVariable1);
        printf("\nptr is pointing to the value:%d\n" , *ptr);

        int array[3] = {1,2,3},offset,i; //Initialize an array of three elements
        printf("\nThe number of bytes in the array is %d\n", sizeof(array));

        ptr = array; // Assign memory address of arr to pointer
        printf("\nThe number of bytes in the ptr is %d\n", sizeof(*ptr));

    printf("\nThe total number of elements in the array is %d\n", sizeof(array)/sizeof(*ptr));

        // Print out associated memory addresses and their values
        printf("\nThe memory address allocated to array at the time of it's creation:%d\n", array);
        printf("\nptr is now pointing to memory address array[0] or value now contained in ptr:%d\n",ptr);
        printf("\nThe value at array[0]:%d\n",*ptr);

        ptr++; //Adds 4 to the value(address) contained in ptr i.e to address of array[0] and now contains the address of array[1]
        printf("\nptr is now pointing to memory address array[1] or value now contained in ptr:%d\n",ptr);
        printf("\nThe value at array[1]:%d\n",*ptr);

        ptr--; //Subtracts 4 from the value(address) contained in ptr i.e from address of array[1] and now contains the address of array[0]
        printf("\nptr is now pointing to memory address array[0] or value now contained in ptr:%d\n",ptr);
        printf("\nThe value at array[0]:%d\n",*ptr);

        ptr = ptr+2; //Adds 4 to the value(address) contained in ptr i.e to address of array[0] and now contains the address of array[2]
        printf("\nptr is now pointing to memory address array[2] or value now contained in ptr:%d\n",ptr);
        printf("\nThe value at array[2]:%d\n",*ptr);

        ptr = ptr - 2;

        // Displaying array using array subscript notation
        printf( "\nArray printed with:\nArray subscript notation\n" );
        for ( i = 0; i < 3; ++i )
        {
                printf( "array[ %d ] = %d\n", i,array[ i ]);
        }

        // Displaying array using array name and pointer/offset notation
        printf( "\nPointer/offset notation where the pointer is the array name\n" );
        for ( offset = 0; offset < 3; ++offset )
        {
                printf( "*( array + %d ) = %d\n", offset, *(array + offset) );
        }

        // Displaying array using ptr and pointer/offset notation
        printf( "\nPointer/offset notation\n" );
```

```
            for ( offset = 0; offset < 3; ++offset )
            {
                    printf( "*( ptr + %d ) = %d\n", offset, *(ptr + offset) );
            }

            // Displaying array using ptr and array subscript notation
            printf( "\nPointer subscript notation\n" );
            for ( i = 0; i < 3; ++i )
            {
                    printf( "ptr[ %d ] = %d\n", i,ptr[ i ]);
            }

            return 0;
}
```

```
■ E:\Solat\ITC\Lab10\test.exe                                           □ □ ⊠

The memory address allocated to ptr at the time of it's creation:2293312

ptr is pointing to memory address or value contained in ptr:2293308

The memory address allocated to intVariable at the time of it's creation:2293308
The value contained in intVariable:10

ptr is pointing to the value:10

The number of bytes in the array is 12

The number of bytes in the ptr is 4

The total number of elements in the array is 3

The memory address allocated to array at the time of it's creation:2293296

ptr is now pointing to memory address array[0] or value now contained in ptr:2293296

The value at array[0]:1

ptr is now pointing to memory address array[1] or value now contained in ptr:2293300

The value at array[1]:2

ptr is now pointing to memory address array[0] or value now contained in ptr:2293296

The value at array[1]:1

ptr is now pointing to memory address array[2] or value now contained in ptr:2293304

The value at array[1]:3

Array printed with:
Array subscript notation
array[ 0 ] = 1
array[ 1 ] = 2
array[ 2 ] = 3

Pointer/offset notation where the pointer is the array name
*( array + 0 ) = 1
*( array + 1 ) = 2
*( array + 2 ) = 3

Pointer/offset notation
*( ptr + 0 ) = 1
*( ptr + 1 ) = 2
*( ptr + 2 ) = 3

Pointer subscript notation
ptr[ 0 ] = 1
ptr[ 1 ] = 2
ptr[ 2 ] = 3

------------------------------------
Process exited after 0.06422 seconds with return value 0
Press any key to continue . . .
```

Pointers to Structures

You can define pointers to structures in very similar way as you define pointer to any other variable as follows –

```
struct Books *struct_pointer;
```

Now, you can store the address of a structure variable in the above defined pointer variable. To find the address of a structure variable, place the & operator before the structure's name as follows –

```
struct_pointer = &Book1;
```

To access the members of a structure using a pointer to that structure, you must use the -> operator as follows –

```
struct_pointer->title;
```

Let us re-write above example using structure pointer, hope this will be easy for you to understand the concept –

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

void printBook( struct Books *book );

struct Books {
   char title[50];
   char author[50];
   char subject[100];
   int book_id;
};
int main() {
   struct Books Book1;        // Declare Book1 of type Book
   struct Books Book2;        // Declare Book2 of type Book

   // Book 1 specification
   strcpy( Book1.title, "Learn C Programming");
   strcpy( Book1.author, "Ahmad");
   strcpy( Book1.subject, "C Programming");
   Book1.book_id = 6495407;

   // Book 2 specification
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Ali");
   strcpy( Book2.subject, "Telecom");
   Book2.book_id = 6495700;

   // Print Book1 info, passing address of structure
   printBook( &Book1 );

   // Print Book1 info, passing address of structure
   printBook( &Book2 );

   return 0;
}
```

```c
// This function accept pointer to structure as parameter.
void printBook( struct Books *book ) {
    printf("Book title : %s", book->title)
    printf("Book author : %s",book->author);
    printf("Book subject : %s",book->subject);
    printf("Book id : %s",book->book_id);
}
```
When the above code is compiled and executed, it produces the following result –
```
Book title : Learn C Programming
Book author : Ahmad
Book subject : C Programming
Book id : 6495407
Book title : Telecom Billing
Book author : Ali
Book subject : Telecom
Book id : 6495700
```

5.0 Dynamic Memory

The process of allocating memory during program execution is called dynamic memory allocation. The ability for a program to obtain more memory space at execution time to hold new nodes, and to release space no longer needed is known as dynamic memory management.

5.1 Importance of Dynamic memory

Many times, it is not known in advance how much memory will be needed to store particular information in a defined variable and the size of required memory can be determined at run time. For example, we may want to hold someone's name, but we do not know how long their name is until they enter it. Or we may want to read in a number of records from disk, but we don't know in advance how many records there are. Or we may be creating a game, with a variable number of monsters (that changes over time as some monsters die and new ones are spawned) trying to kill the player.

C language offers 4 dynamic memory allocation functions. They are,

Function	Syntax
malloc ()	malloc (number *sizeof(int));
calloc ()	calloc (number, sizeof(int));
realloc ()	realloc (pointer_name, number * sizeof(int));
free ()	free (pointer_name);

Malloc()

- malloc function is used to allocate space in memory during the execution of the program.
- malloc does not initialize the memory allocated during execution. It carries garbage value.
- Malloc function returns null pointer if it couldn't able to allocate requested amount of memory.

Calloc()

- calloc function is also like malloc function. But calloc initializes the allocated memory to zero. But, malloc doesn't.

Realloc()

- realloc function modifies the allocated memory size by malloc and calloc functions to new size.
- If enough space doesn't exist in memory of current block to extend, new block is allocated for the full size of reallocation, then copies the existing data to new block and then frees the old block.

Free()

- free function frees the allocated memory by malloc, calloc, realloc functions and returns the memory to the system.

Malloc & free Sample Code:

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

int main() {
  int n, i, *ptr, sum = 0;

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

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

  // if memory cannot be allocated
  if(ptr == NULL) {
    printf("Error! memory not allocated.");
    exit(0);
  }

  printf("Enter elements: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  }

  printf("Sum = %d", sum);
  // deallocating the memory
  free(ptr);

  return 0;
}
```

C:\Users\Shoaib\Documents\malloc_example_02.exe

```
Enter number of elements: 5
Enter elements: 2
1
2
3
1
Sum = 9
-------------------------------
Process exited after 42.24 seconds with return value 0
Press any key to continue . . .
```

Calloc & free Sample Code:

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

int main() {
  int n, i, *ptr, sum = 0;
  printf("Enter number of elements: ");
  scanf("%d", &n);

  ptr = (int*) calloc(n, sizeof(int));
  if(ptr == NULL) {
    printf("Error! memory not allocated.");
    exit(0);
  }

  printf("Enter elements: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  }

  printf("Sum = %d", sum);
  free(ptr);
  return 0;
}
```

```
C:\Users\Shoaib\Documents\calloc_example_02.exe

Enter number of elements: 3
Enter elements: 6
9
2
Sum = 17
----------------------------------
Process exited after 21.82 seconds with return value 0
Press any key to continue . . .
```

Realloc Sample Code:

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

int main() {
  int *ptr, i , n1, n2;
  printf("Enter size: ");
  scanf("%d", &n1);

  ptr = (int*) malloc(n1 * sizeof(int));

  printf("Addresses of previously allocated memory:\n");
  for(i = 0; i < n1; ++i)
    printf("%pc\n",ptr + i);

  printf("\nEnter the new size: ");
  scanf("%d", &n2);

  // rellocating the memory
  ptr = realloc(ptr, n2 * sizeof(int));

  printf("Addresses of newly allocated memory:\n");
  for(i = 0; i < n2; ++i)
    printf("%pc\n", ptr + i);

  free(ptr);

  return 0;
}
```

```
 C:\Users\Shoaib\Documents\realloc_01.exe
Enter size: 2
Addresses of previously allocated memory:
0000000000C11400c
0000000000C11404c

Enter the new size: 5
Addresses of newly allocated memory:
0000000000C11400c
0000000000C11404c
0000000000C11408c
0000000000C1140Cc
0000000000C11410c

--------------------------------
Process exited after 5.863 seconds with return value 0
Press any key to continue . . .
```

5.2 Difference between static memory allocation and dynamic memory allocation in C

Static memory allocation	Dynamic memory allocation
In static memory allocation, memory is allocated while writing the C program. Actually, user requested memory will be allocated at compile time.	In dynamic memory allocation, memory is allocated while executing the program. That means at run time.
Memory size can't be modified while execution. Example: array	Memory size can be modified while execution. Example: Linked list

Exercises:

Task 1:

Write a program that calculates the sum of all the elements in array using pointers
(**Note: Generate the array via user)**

Task 2:

Write a program to implement the exchange or swap the values of 3 variables{a,b,c}. To implement this, you need to write a function called swaped().

                        void swaped(int *aPtr, int *bPtr, int *cPtr);

   such that; b ----> temp
           a ----> b
           c ---->   a
           temp -> a

Task 3:

Consider there are two structures Employee (depended structure) and another structure called Organization(Outer structure). The structure Organization has the data members like organisation_name,organisation_number. The Employee structure is nested inside the structure Organization and it has the data members like employee_id, name, salary.
org.emp.employee_id;
org.emp.name;
org.emp.salary;
org.organisation_name;
org.organisation_number;
Here, org is the structure variable of the outer structure Organization and emp is
the structure variable of the inner structure Employee.

Output the following data using above structure
The size of structure organisation : 123 //This is the size of the organization building in sq-ft
Organisation Name : NU-Fast
Organisation Number : NUFAST123ABC
Employee id : 127
Employee name : Linus Sebastian
Employee Salary : 400000

Task 4:
Let us work on the menu of a library. Create a structure containing book information like accession number, name of author, book title and flag to know whether book is issued or not.
Create a menu in which the following can be done.
1 - Display book information
2 - Add a new book
3 - Display all the books in the library of a particular author
4 - Display the number of books of a particular title
5 - Display the total number of books in the library
6 - Issue a book
(If we issue a book, then its number gets decreased by 1 and if we add a book, its number gets increased by 1)

Task 5:
You need to implement the the following 2 struct.
struct Student{}; struct Register{};
Student contains attribute StudentId, FirstName, LastName, cellno, email.
Register contains attribute CourseId, CourseName.
Now you need to inherit the Register struct in Student struct. It means that student struct holds the variable of Register struct variable. After that you need to take input for 5 students and then print them [Hint: Declare array of struct Student std[5]; for 5 students]

Task 6:
You are transporting some boxes through a tunnel, where each box is a parallelepiped, and is characterized by its length, width and height.
The height of the tunnel is 41 feet, and the width can be assumed to be infinite. A box can be carried through the tunnel only if its height is strictly less than the tunnel's height. Find the volume of each box that can be successfully transported to the other end of the tunnel. Note: Boxes cannot be rotated.
Sample Input 0
4
5 5 5
1 2 40
10 5 41
7 2 42
Sample Output 0
125
80
Explanation: The first box is low, only 5 feet tall, so it can pass through the tunnel and its volume is 5*5*5=125. The second box is sufficiently low, its volume is 1*2*40=80. The third box is exactly 41 feet tall so it cannot pass. The same can be said about the fourth box.
(**Note: You are only allowed to use structures. For the initial inputs can be handled via loops or embedded in the code**)

Task 7:
Write a program that implements the function(WordCount).
        int WordCount(char *Text, int *size);

Task 8:
Write a program that takes N no of character in an array. Now you need to correct the array and convert it into the Sentence Case. To correct that, you need to pass that array to a function using pointer.
        void SentenceCase(char *Text, int *size);
In the end you need to print the array in Main() to check.
Sentence Case: You capitalize just the first letter of the first word in sentence. The rest of the words and letters in the title should not be capitalized. Also the first letter after the fullstop(.)

Task 9:
Write a C Program to Find Largest Number in an array Using Dynamic Memory Allocation.
Note: The array must be dynamically resized and initially the array size will be taken by the user and then resized by the user
Example arr[5]={1,2,3,4,5}
//After resizing
Arr[7]={1,2,3,4,5,6,7}
Take the same array and resize it.