CC-112L

Programming Fundamentals

Laboratory 08

Pointers – I

Version: 1.0.0

Release Date: 05-09-2022

Department of Information Technology
University of the Punjab
Lahore, Pakistan

Contents:

Learning Objectives:

- Pointer Variable Definition
- Pointer Variable Initialization
- Pointer Operators
- Passing Arguments to Function by Reference
- sizeof Operator
- Pointer Arithmetic

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Teacher Assistants	Usman Ali	bitf19m007@pucit.edu.pk
	Saad Rahman	bsef19m021@pucit.edu.pk

Background and Overview:

Pointers in C:

Pointers are special variables that are used to store memory addresses rather than values.

Pointer Operators:

C++ provides two pointer operators, which are Address of Operator **“&”** and Indirection Operator **“*”**.

Pass by Value:

Passing by value refers to passing the actual value of the variable into the parameter of calling function.

Pass by Reference:

Passing by reference refers to a method of passing the address of an argument in the parameter of the calling function.

sizeof Operator:

The sizeof operator is a common operator in C. It is a compile-time unary operator and used to compute the size of its operand. It returns the size of a variable. It can be applied to any data type, pointer type variables.

© https://www.tutorialspoint.com/sizeof-operator-in-c

Activities:

Pre-Lab Activities:

Pointer Variable Definition & Initialization:

Declaring Pointers:

The following statement defines the variable countPtr as an int *, which is a pointer to an integer type variable

int *countPtr;

This definition is read as **"countPtr is a pointer to int"** or **"countPtr points to a variable of type int."**

The **"*"** indicates that the variable is a pointer.

Initializing Pointer:

A pointer may be initialized to NULL, 0 or a memory address of a variable.

- A pointer with NULL value points to nothing
- Pointer initialized to 0 is same as NULL

Pointer Operators:

The Address (&) Operator:

The address operator (&) returns the address of its operand. For example, given the following definition of y

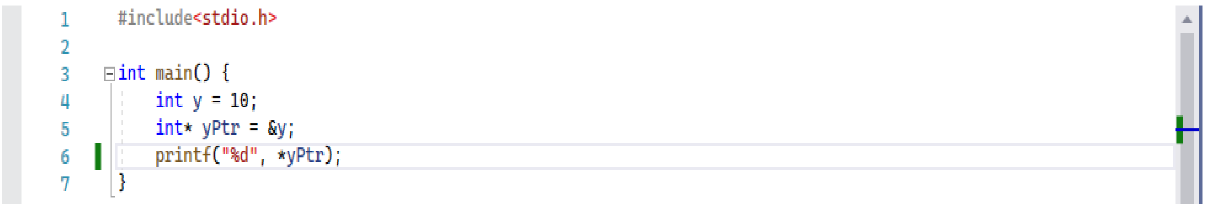
int y = 5;

the statement **"int *yPtr = &y;"** initializes pointer variable yPtr with variable y's address in memory. yPtr is then said to "pointer to" y.

The Indirection (*) Operator:

Indirection operator (*) is used with a pointer operand to get the value of the variable to which the pointer points. For example, the following program prints 10, which is the value of a variable y.

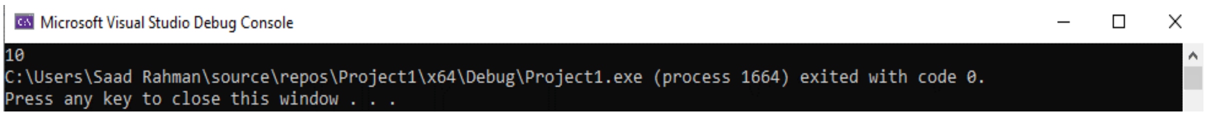
Using * in this manner is called **"dereferencing a pointer"**.



```c
#include<stdio.h>

int main() {
    int y = 10;
    int* yPtr = &y;
    printf("%d", *yPtr);
}
```

Fig 01. (Pointer Operators)

Following is the output of the above program.

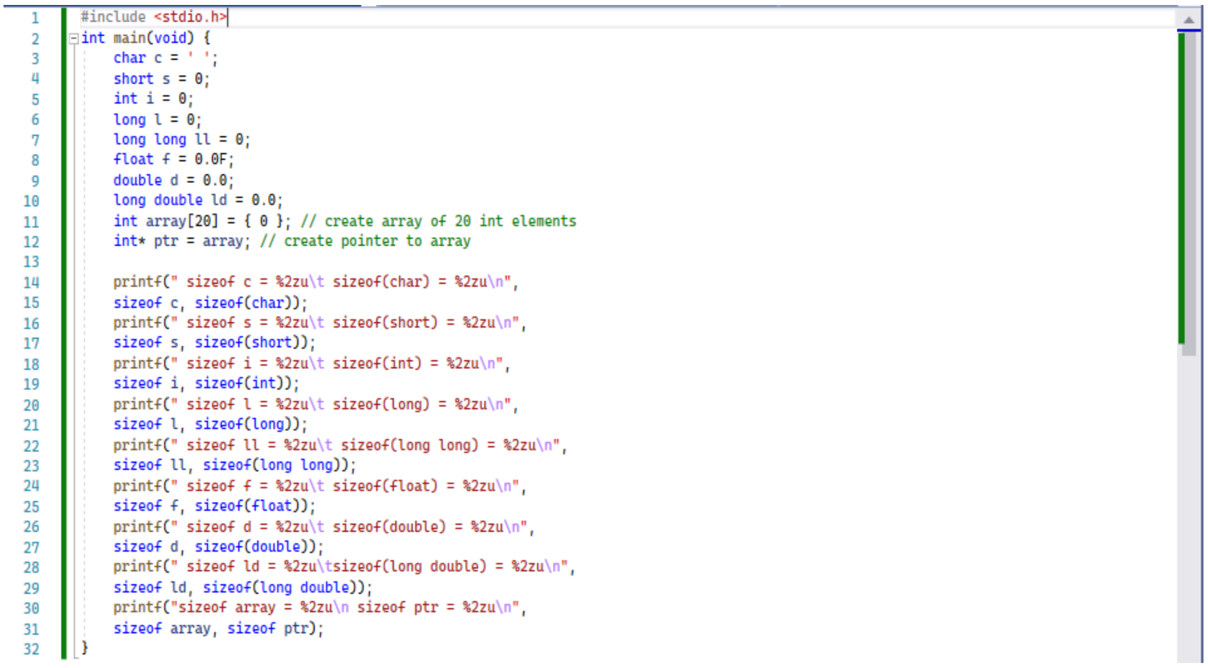


```
10
C:\Users\Saad Rahman\source\repos\Project1\x64\Debug\Project1.exe (process 1664) exited with code 0.
Press any key to close this window . . .
```

Fig 02. (Pointer Operators)

sizeof Operator:

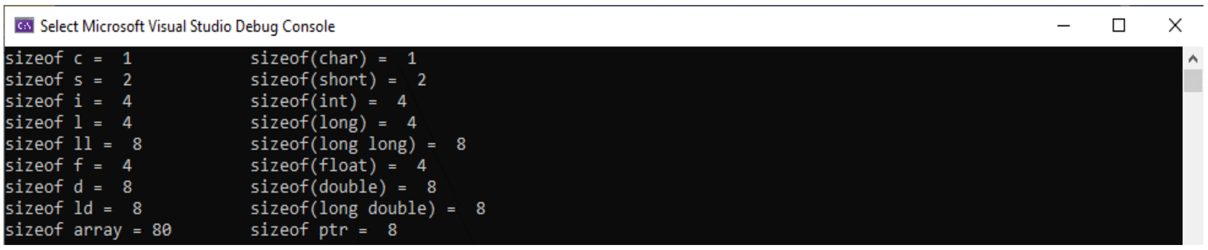
C provides the unary operator sizeof to determine an objects or type's size in **"bytes"**. This operator is applied at compilation time unless its operand is a variable-length array. Following program shows the **sizeof** the standard types, array and pointer.



```c
#include <stdio.h>
int main(void) {
    char c = ' ';
    short s = 0;
    int i = 0;
    long l = 0;
    long long ll = 0;
    float f = 0.0F;
    double d = 0.0;
    long double ld = 0.0;
    int array[20] = { 0 }; // create array of 20 int elements
    int* ptr = array; // create pointer to array

    printf(" sizeof c = %2zu\t sizeof(char) = %2zu\n",
    sizeof c, sizeof(char));
    printf(" sizeof s = %2zu\t sizeof(short) = %2zu\n",
    sizeof s, sizeof(short));
    printf(" sizeof i = %2zu\t sizeof(int) = %2zu\n",
    sizeof i, sizeof(int));
    printf(" sizeof l = %2zu\t sizeof(long) = %2zu\n",
    sizeof l, sizeof(long));
    printf(" sizeof ll = %2zu\t sizeof(long long) = %2zu\n",
    sizeof ll, sizeof(long long));
    printf(" sizeof f = %2zu\t sizeof(float) = %2zu\n",
    sizeof f, sizeof(float));
    printf(" sizeof d = %2zu\t sizeof(double) = %2zu\n",
    sizeof d, sizeof(double));
    printf(" sizeof ld = %2zu\tsizeof(long double) = %2zu\n",
    sizeof ld, sizeof(long double));
    printf("sizeof array = %2zu\n sizeof ptr = %2zu\n",
    sizeof array, sizeof ptr);
}
```

Fig. 03 (sizeof Operator)

The output of the above program is as follows.



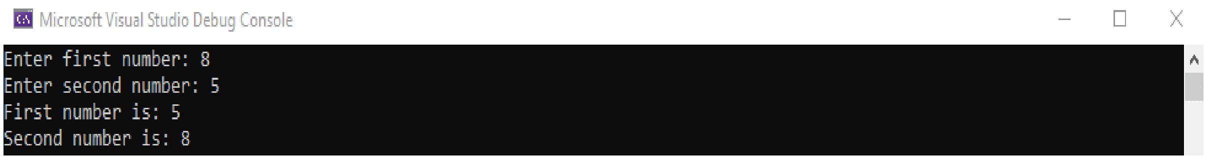
```
sizeof c =  1           sizeof(char) =  1
sizeof s =  2           sizeof(short) =  2
sizeof i =  4           sizeof(int) =  4
sizeof l =  4           sizeof(long) =  4
sizeof ll =  8          sizeof(long long) =  8
sizeof f =  4           sizeof(float) =  4
sizeof d =  8           sizeof(double) =  8
sizeof ld =  8          sizeof(long double) =  8
sizeof array = 80       sizeof ptr =  8
```

Fig. 04 (sizeof Operator)

Task 01: Swap two integers **[Estimated 30 minutes / 20 marks]**

Create a C program that:

- Takes values of two integers as an input from user
- Swap (interchange) the values of integers using pointer
- Display the integer values after swapping



```
Enter first number: 8
Enter second number: 5
First number is: 5
Second number is: 8
```

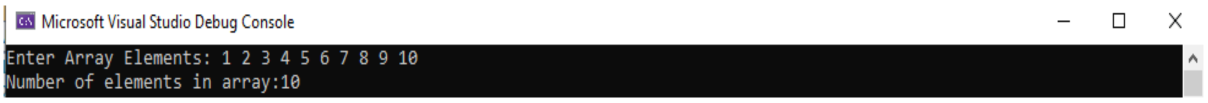
Fig. 05 (Pre-Lab Task)

- Submit **".c"** file named your **"Roll No"** on Google Classroom

Task 02: Elements in Array **[Estimated 15 minutes / 10 marks]**

Create a C program that:

- Declare an integer array of size 10
- Initialize the array by taking input from the user
- Display the no of elements in the array using **"sizeof Operator"**



```
Enter Array Elements: 1 2 3 4 5 6 7 8 9 10
Number of elements in array:10
```

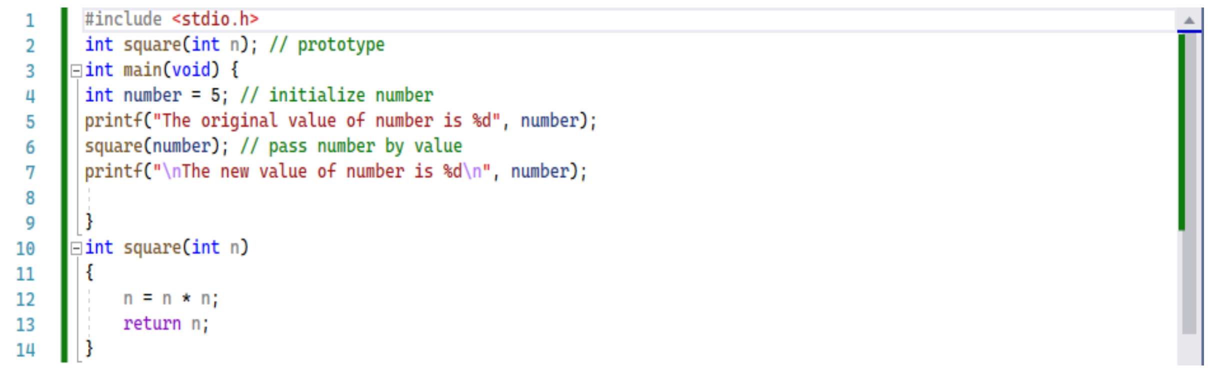
Fig. 06 (Pre-Lab Task)

- Submit **".c"** file named your **"Roll No"** on Google Classroom

In-Lab Activities:

Pass by Value:

Passing by value refers to passing the actual value of the variable into the parameter of calling function. An example is shown in the following program.

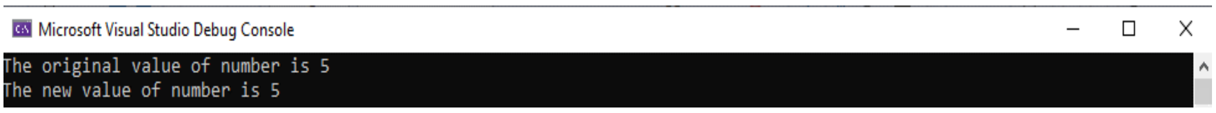


```
1  #include <stdio.h>
2  int square(int n); // prototype
3  int main(void) {
4  int number = 5; // initialize number
5  printf("The original value of number is %d", number);
6  square(number); // pass number by value
7  printf("\nThe new value of number is %d\n", number);
8  
9  }
10 int square(int n)
11 {
12     n = n * n;
13     return n;
14 }
```

Fig. 07 (Pass by Value)

Execution of above code is as follows:

- In line 4, variable **"number"** is initialized to 5
- In line 5, the value of variable **"number"** is displayed
- In line 6, square function is called, in which argument is passed by value
- Line 10 to line 13, function returns the square of the number
- In line 7, the value of variable **"number"** is displayed after the execution of the square function

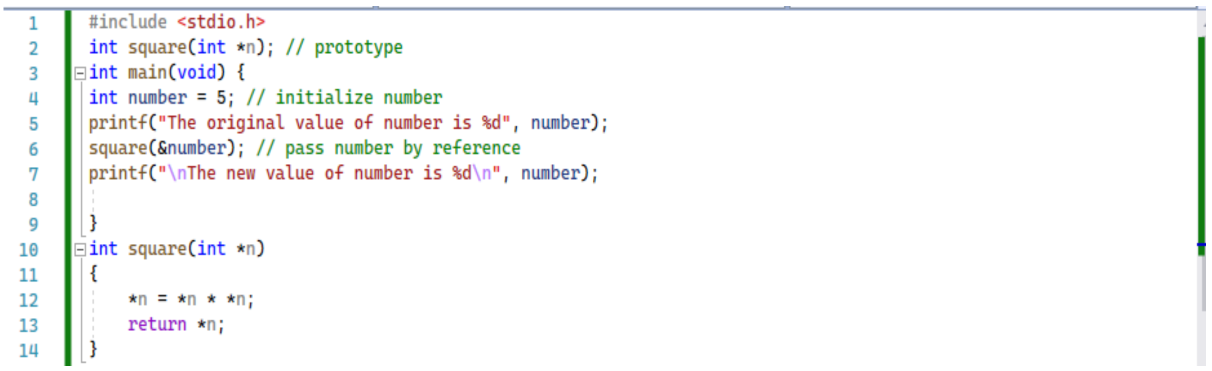


```
The original value of number is 5
The new value of number is 5
```

Fig. 08 (Pass by Value)

Pass by Reference:

Passing by value refers to passing the actual value of the variable into the parameter of calling function. An example is shown in the following program.



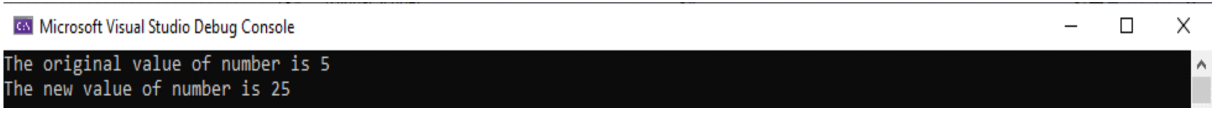
```
1  #include <stdio.h>
2  int square(int *n); // prototype
3  int main(void) {
4  int number = 5; // initialize number
5  printf("The original value of number is %d", number);
6  square(&number); // pass number by reference
7  printf("\nThe new value of number is %d\n", number);
8  
9  }
10 int square(int *n)
11 {
12     *n = *n * *n;
13     return *n;
14 }
```

Fig. 09 (Pass by Reference)

Execution of above code is as follows:

- In line 4, variable **"number"** is initialized to 5
- In line 5, the value of variable **"number"** is displayed

- In line 6, square function is called, in which argument is passed by reference
- Line 10 to line 13, function returns the square of the number
- In line 7, the value of variable **"number"** is displayed after the execution of the square function



```
The original value of number is 5
The new value of number is 25
```

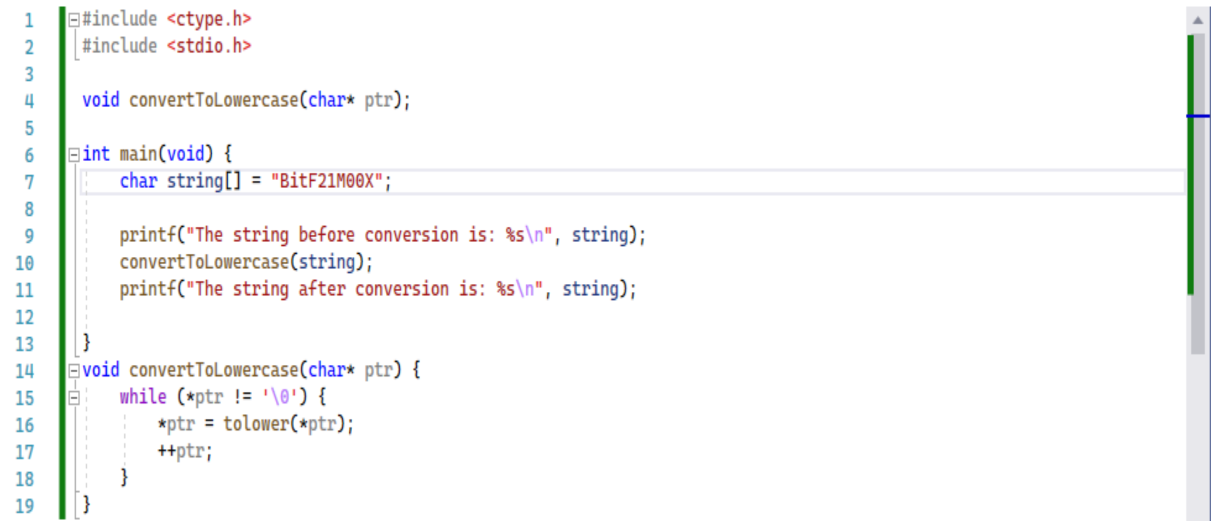
Fig. 10 (Pass by Reference)

const Qualifier with Pointers:

Non-Constant Pointer to Non-Constant Data:

The highest level of data access is granted by a non-constant pointer to non-constant data. The data can be modified through the dereferenced pointer, and the pointer can be modified to point to other data items.

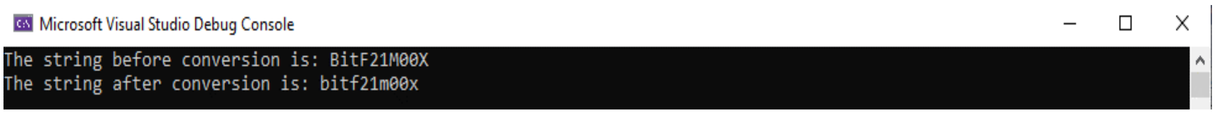
Following is an example in which a function uses such a pointer to receive a string argument, then convert each character to lowercase in the string.



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

void convertToLowercase(char* ptr);

int main(void) {
    char string[] = "BitF21M00X";

    printf("The string before conversion is: %s\n", string);
    convertToLowercase(string);
    printf("The string after conversion is: %s\n", string);

}
void convertToLowercase(char* ptr) {
    while (*ptr != '\0') {
        *ptr = tolower(*ptr);
        ++ptr;
    }
}
```

Fig. 11 (Non-Constant Pointer to Non-Constant Data)

The output of the above program is:



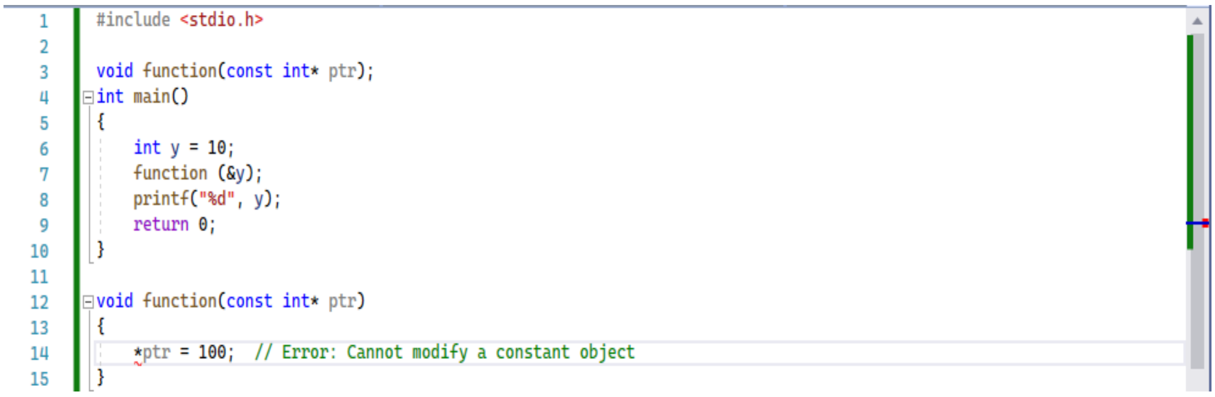
```
The string before conversion is: BitF21M00X
The string after conversion is: bitf21m00x
```

Fig. 12 (Non-Constant Pointer to Non-Constant Data)

Non-Constant Pointer to Constant Data:

A non-constant pointer to constant data can be modified to point to any data item of the appropriate type, but the data to which it points cannot be modified. A function receives such a pointer to process an array argument's elements without modifying them.

In the following example **"ptr"** cannot be used to modify the value of the variable to which it points.



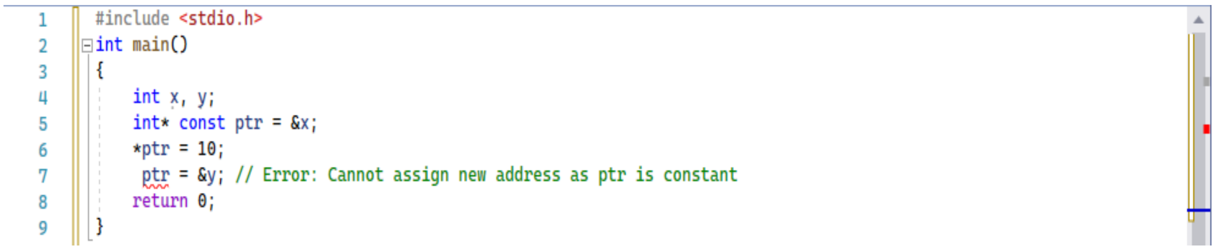
```c
#include <stdio.h>

void function(const int* ptr);
int main()
{
    int y = 10;
    function (&y);
    printf("%d", y);
    return 0;
}

void function(const int* ptr)
{
    *ptr = 100;  // Error: Cannot modify a constant object
}
```

Fig. 13 (Non-Constant Pointer to Constant Data)

Constant Pointer to Non-Constant Data:

A constant pointer to non-constant data always points to the same memory location (cannot be modified due to constant pointer) and the data at that location cannot be modified through the pointer.

In the following example, new address cannot be assigned to **"ptr"** as it is constant.



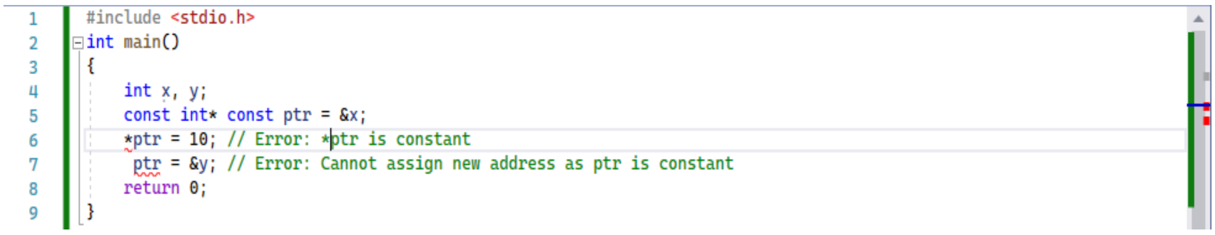
```c
#include <stdio.h>
int main()
{
    int x, y;
    int* const ptr = &x;
    *ptr = 10;
     ptr = &y; // Error: Cannot assign new address as ptr is constant
    return 0;
}
```

Fig. 14 (Constant Pointer to Non-Constant Data)

Constant Pointer to Constant Data:

A constant pointer to non-constant data always points to the same memory location (cannot be modified due to constant pointer), but the data at that location can be modified through the pointer.

In the following example, new address cannot be assigned to **"ptr"** as it is constant and **"ptr"** cannot be used to modify the value of the variable to which it points.



```c
#include <stdio.h>
int main()
{
    int x, y;
    const int* const ptr = &x;
    *ptr = 10; // Error: *ptr is constant
     ptr = &y; // Error: Cannot assign new address as ptr is constant
    return 0;
}
```

Fig. 15 (Constant Pointer to Constant Data)

Pointer Arithmetic:

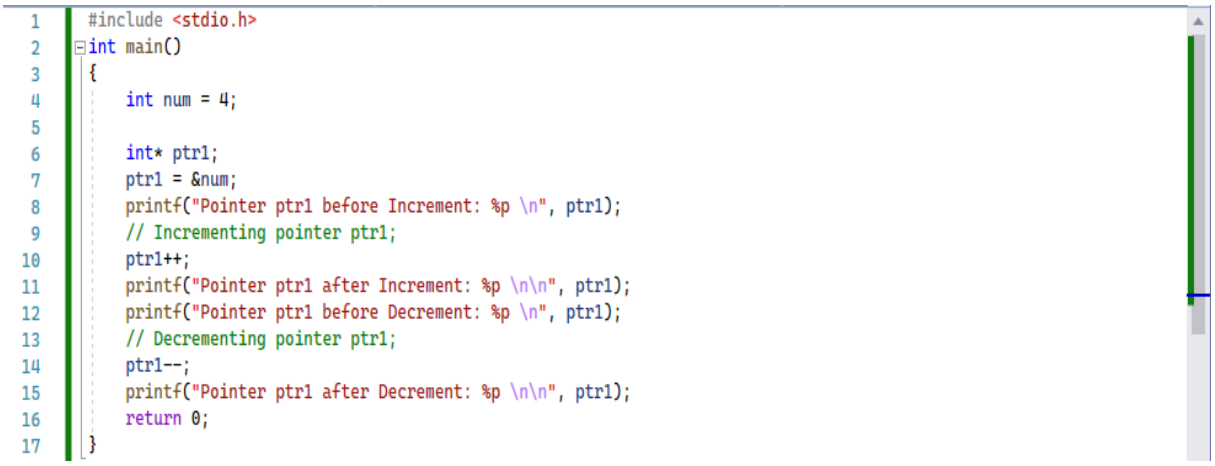
The following arithmetic operations are allowed for pointers:

- Incrementing (++)
- Decrementing (--)
- Adding integer to pointer (+ or +=)
- Subtracting integer from pointer (- or -=)

- Subtracting one pointer from another

Increment/Decrement Pointer:

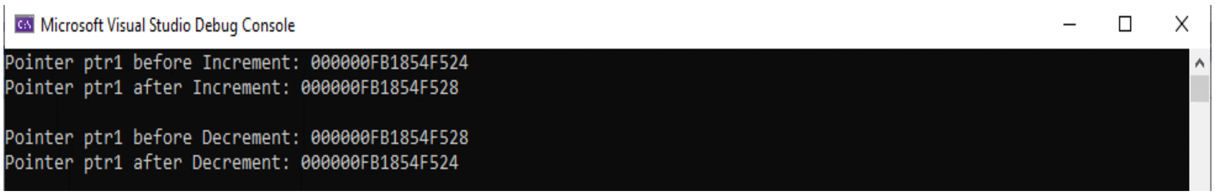
When a pointer is incremented, it points to the next memory location, similarly when it is decremented it points to previous memory location.



```c
#include <stdio.h>
int main()
{
    int num = 4;

    int* ptr1;
    ptr1 = &num;
    printf("Pointer ptr1 before Increment: %p \n", ptr1);
    // Incrementing pointer ptr1;
    ptr1++;
    printf("Pointer ptr1 after Increment: %p \n\n", ptr1);
    printf("Pointer ptr1 before Decrement: %p \n", ptr1);
    // Decrementing pointer ptr1;
    ptr1--;
    printf("Pointer ptr1 after Decrement: %p \n\n", ptr1);
    return 0;
}
```

Fig. 16 (Increment/Decrement Pointer)

The output of the above program is following:



```
Pointer ptr1 before Increment: 000000FB1854F524
Pointer ptr1 after Increment: 000000FB1854F528

Pointer ptr1 before Decrement: 000000FB1854F528
Pointer ptr1 after Decrement: 000000FB1854F524
```

Fig. 17 (Increment/Decrement Pointer)

Task 01: Boundary Values [30 minutes / 20 marks]

Write a C program which:

- Take number of data values as input from user with each value
- Declare and initialize two pointers which points to largest and smallest value
- Display the values of the pointers as shown in the table below

Input	Output
5 10 7 8 9 12	Largest Value is: 12 Smallest Value is: 7
3 100 87 98	Largest Value is: 100 Smallest Value is: 87

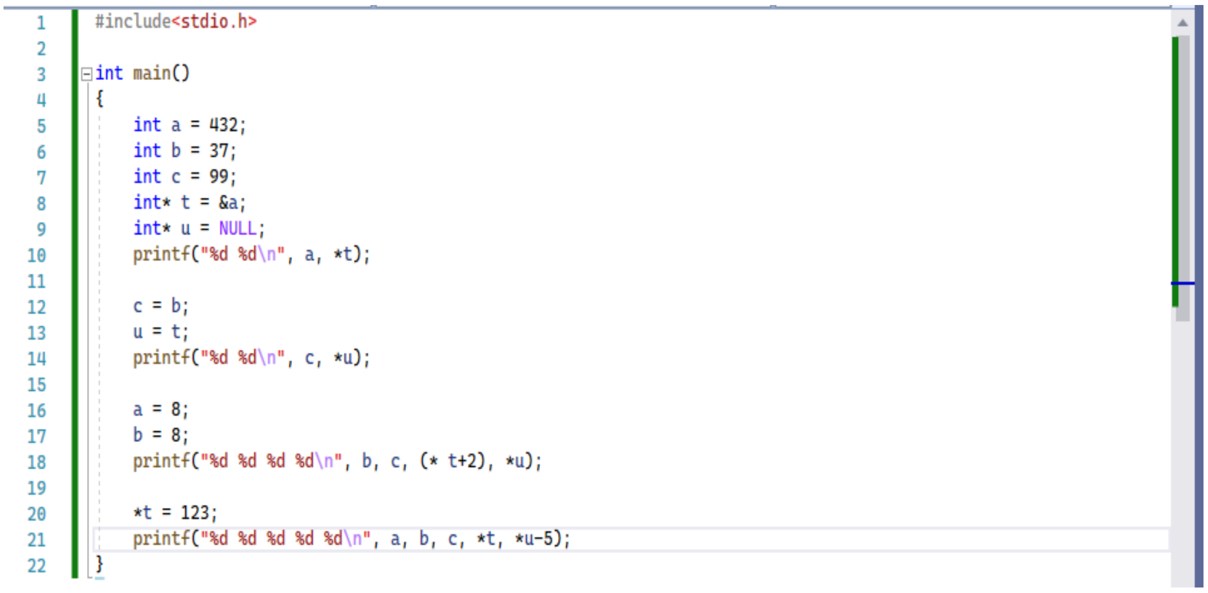
Task 02: Average [20 minutes / 10 marks]

Write a C program which:

- Takes three numbers as average from the user
- Calculates the average using pointers
- Display the average on the Console

Task 03: Dry Run [30 minutes / 20 marks]

Trace the following C program and fill the table below.



```
1   #include<stdio.h>
2
3   int main()
4   {
5       int a = 432;
6       int b = 37;
7       int c = 99;
8       int* t = &a;
9       int* u = NULL;
10      printf("%d %d\n", a, *t);
11
12      c = b;
13      u = t;
14      printf("%d %d\n", c, *u);
15
16      a = 8;
17      b = 8;
18      printf("%d %d %d %d\n", b, c, (* t+2), *u);
19
20      *t = 123;
21      printf("%d %d %d %d %d\n", a, b, c, *t, *u-5);
22  }
```

Fig. 18 (In-Lab Task)

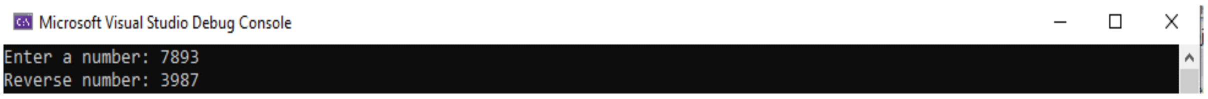
Line No	Output
10	
14	
18	
21	

Task 04: Reverse Number **[20 minutes / 20 marks]**

Create a program in C which

- Takes a number as input from user
- Defines a function ReverseOrder
- Pass the number to function using pass by reference
- Display the reversed number

Sample Output:

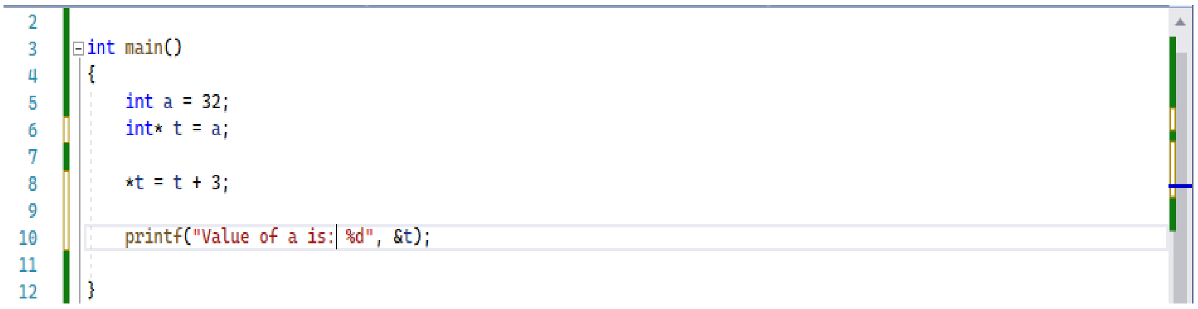


```
Enter a number: 7893
Reverse number: 3987
```

Fig. 19 (In-Lab Task)

Task 05: Find the Error **[20 minutes / 10 marks]**

Find out the error in the following program:



```

int main()
{
    int a = 32;
    int* t = a;

    *t = t + 3;

    printf("Value of a is: %d", &t);

}
```

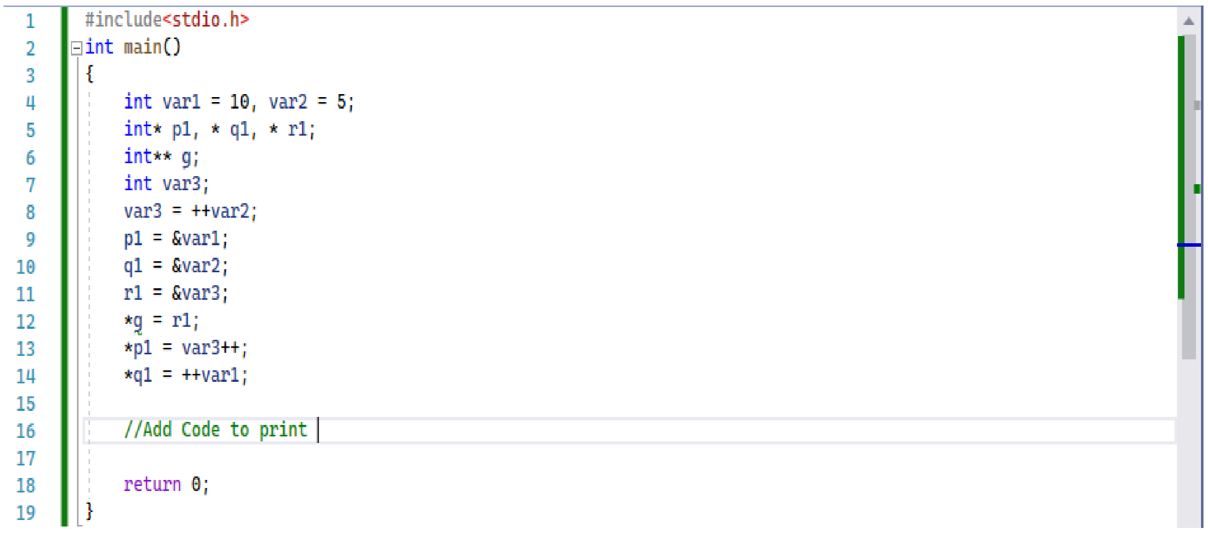
Fig. 20 (In-Lab Task)

Post-Lab Activities:

Task 01: Pointer to Pointer and Addresses **[Estimated 60 minutes / 50 marks]**

Consider the following code:

s



```c
#include<stdio.h>
int main()
{
    int var1 = 10, var2 = 5;
    int* p1, * q1, * r1;
    int** g;
    int var3;
    var3 = ++var2;
    p1 = &var1;
    q1 = &var2;
    r1 = &var3;
    *g = r1;
    *p1 = var3++;
    *q1 = ++var1;

    //Add Code to print

    return 0;
}
```

Fig. 21 (Post-Lab Task)

Print the following things and fill in the table.

Address of variable var1:	
Address of variable var2:	
Address of variable var3:	
Address of pointer variable p1:	
Address of pointer variable q1:	
Address of pointer variable r1:	
Value at location pointed by variable p1:	
Value at location pointed by variable q1:	
Value at location pointed by variable r1:	
Value at location pointed by variable g:	
Value at location pointed by the pointer that is pointed by g:	
Address of location pointed by the pointer that is pointed by g:	
Address of pointer variable g:	

You need to submit following things on Google Classroom:

- Complete code .c file
- Above table filled (Share MS Word file)
- Memory map diagram (Share MS Word file) E.g.

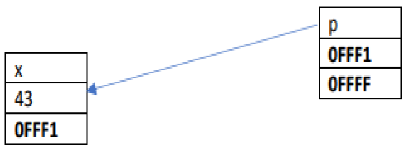


Fig. 22 (Post-Lab Task)

Task 02: Reverse Order **[Estimated 60 minutes / 30 marks]**

Write a C program in which:

- Takes a string as an input from the user
- Make a function named **"reverseOrder"** that reverses the order of the string
- Display the reversed string

You are not allowed to use the array subscript [] notation.

Input	**Output**
This is reverse program	program reverse is This
My Roll No is 100	100 is No Roll My

Submissions:

- For Pre-Lab Activity:
  - Submit the .c file on Google Classroom and name it to your roll no.
- For In-Lab Activity:
  - Save the files on your PC.
  - TA's will evaluate the tasks offline.
- For Post-Lab Activity:
  - Submit the .c file on Google Classroom and name it to your roll no.

Evaluations Metric:

- All the lab tasks will be evaluated offline by TA's

• **Division of Pre-Lab marks:**	**[30 marks]**
▪ Swap two integers	[20 marks]
▪ Elements in Array	[10 marks]
• **Division of In-Lab marks:**	**[80 marks]**
▪ Task 01: Boundary Values	[20 marks]
▪ Task 02: Average	[10 marks]
▪ Task 03: Dry Run	[20 marks]
▪ Task 04: Reverse Number	[20 marks]
▪ Task 05: Find the error	[10 marks]
• **Division of Post-Lab marks:**	**[80 marks]**
▪ Task01: Pointer to pointer and addresses	[50 marks]
▪ Task02: Reverse Order	[30 marks]

References and Additional Material:

- C Pointers
  https://www.programiz.com/c-programming/c-pointers

- C Pointers and Functions
  https://www.programiz.com/c-programming/c-pointer-functions

Lab Time Activity Simulation Log:

• Slot – 01 – 00:00 – 00:15:	Class Settlement
• Slot – 02 – 00:15 – 00:30:	In-Lab Task
• Slot – 03 – 00:30 – 00:45:	In-Lab Task
• Slot – 04 – 00:45 – 01:00:	In-Lab Task
• Slot – 05 – 01:00 – 01:15:	In-Lab Task
• Slot – 06 – 01:15 – 01:30:	In-Lab Task
• Slot – 07 – 01:30 – 01:45:	In-Lab Task
• Slot – 08 – 01:45 – 02:00:	In-Lab Task
• Slot – 09 – 02:00 – 02:15:	In-Lab Task
• Slot – 10 – 02:15 – 02:30:	In-Lab Task
• Slot – 11 – 02:30 – 02:45:	In-Lab Task
• Slot – 12 – 02:45 – 03:00:	Discussion on Post-Lab Task