



JAIN
DEEMED-TO-BE UNIVERSITY

FACULTY OF
ENGINEERING
AND TECHNOLOGY

Bachelor of Technology
School of computer Science and Engineering
Lab Manual
For
21CSEDSL04:
Data Structures Using C Lab
Prepared and Compiled
By
&

Faculty of Engineering & Technology
Global Campus

45th km NH – 209, Jakkasandra Post, Kanakapura Rd, Bangalore

www.set.jainuniversity.ac.in

Fax STD Code:- 080 Fax:- 2757 7199

CONTENTS

#	TITLE	PAGE NO.
1.	Institute Vision and Mission	1
2.	Department Vision and Mission	2
3.	PEOs	
4.	Program Specific Outcomes (PSO)	
5.	Program Outcomes (PO)	
6.	Mapping of PEOs and POs	
7.	Course Outcome (CO) Statements and CO-PO Mapping	
8.	List of experiment with CO Mapping	
9.	List of Tools used and Ref books	
10.	Aim, Description, Algorithms, Program and Output of each experiment	
11.	Open-Ended Experiments	
12.	Viva questions	
13.	Rubrics for Evaluation (CIA and Semester End Assessment)	

Faculty in-charge(s)

Head of the Department

Institute Vision and Mission

Vision

To be a leading technical institution that offers a transformative education to create leaders and innovators with ethical values to contribute for the sustainable development and economic growth of our nation.

Mission

M1: To impart high standard of engineering education through innovative teaching and research to meet the changing needs of the modern society.

M2: To provide outcome-based education that transforms the students to understand and solve the societal, industrial problems through engineering and technology.

M3: To collaborate with other leading technical institutions and organization to develop globally competitive technocrats and entrepreneurs.

Department Vision and Mission

Vision

To mould student careers in to globally competitive professionals in Information Science and Engineering through technical education, research and services to address the evolving societal needs.

Mission

M1: Impart quality technical education with competence, knowledge, ethics and social responsibility.

M2: Create an environment that promotes research activities through lifelong learning.

M3: Inculcate the best practices that lead to quality placements and personality development.

Program Educational Objectives (PEOs)

A few years after graduation, the Graduates of Computer Science and Engineering will be able

PEO1: A professional Information science Engineer to solve real world problems.

PEO2: A potential graduate to implement the best practices ethically as per industry and societal needs through entrepreneurship

PEO3: Efficient engineer having good communication skills, domain expertise and leadership qualities to acquire further titles in academia and research

Program Specific Outcomes (PSO)

PSO 1: A versatile individual who is dynamic to adopt changes in the trends of Information technology through the concepts of Information retrieval, Security and Analytics

PSO 2: Ability to mine the data in the fields of Management and Information Science with the knowledge gained from programming skills and projects

Program Outcomes

Engineering Graduate's attributes

At the end of the programme, students will be able to	
PO 1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO 2	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO 3	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO 4	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO 5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO 6	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO 7	Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO 8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO 9	Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO 10	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Outcome (CO) Statements and CO-PO Mapping

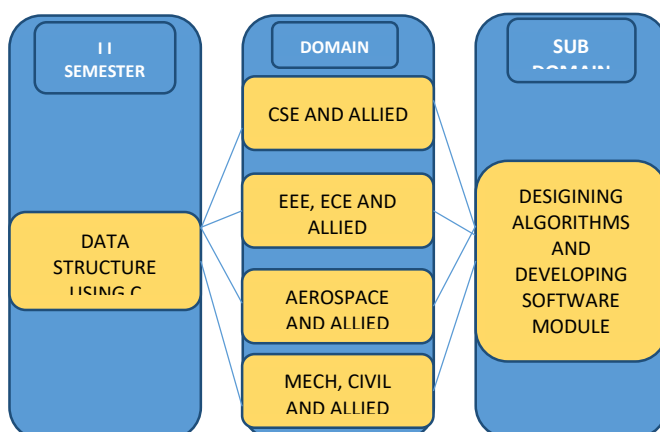
At the end of the course, students will be able to:

Course Outcomes	Description	Bloom's Taxonomy Level
CO1	Illustrate the concepts of Arrays, pointers and functions.	Applying [L3]
CO2	Use suitable recursive method to given problem statements.	Applying [L3]
CO3	Demonstrate the implementation of various operations on stack and queue	Applying [L3]
CO4	Examine the concepts of Linked List and apply various operations on them.	Analyze [L4]
CO5	Demonstrate various binary search tree traversals techniques.	Applying [L3]
CO6	Choose the appropriate data structure to solve the given computational problem.	Analyze [L4]

CO-PO Mapping:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO 10	PO 11	PO 12	PSO1	PSO2
CO1	2	2	-	-	2	-	-	-	-	-	-	1	-	-
CO2	2	2	-	-	2	-	-	-	-	-	-	1	-	-
CO3	2	2	-	-	2	-	-	-	-	-	-	1	-	-
CO4	2	2	-	-	2	-	-	-	-	-	-	1	-	-
CO5	2	2	-	-	2	-	-	-	-	-	-	1	-	-
CO6	2	2	1	1	2	-	-	1	1	1	-	1	-	-

Course Map:



List of Tools used and Ref books

Tools Used: Code Blocks, Turbo c, visual studio.

Code Blocks

Code Blocks is a great tool for beginners who want to get started coding or for those looking for a way to improve their skills. It has features and a user-friendly interface that makes learning coding easy.

CodeBlocks is a cross-platform C++ IDE (Integrated Development Environment) that allows developers to code, debug, build, run and deploy projects. It provides powerful options to customize your development environments, such as source control integration and graphical views of memory and CPU usage.

CodeBlocks also offers support for multiple languages, ranging from C/C++, Java, Python, HTML5, Objective C, PHP It's a free open-source tool.

What is Code::Blocks?

Code: Blocks is an free, open-source, cross-platform Integrated Development Environment or IDE. It's a powerful tool that's made even more useful when used with plugins which further increase its functionality.

Some of the Most Important Features of Code::Blocks

- It's completely open-source and there are no additional or hidden costs.
- It's cross-platform meaning it will run on Linux, Mac, and Windows all the same.
- The platform is written in C++ so no additional libraries are needed.
- High usability as a courtesy of plugins.
- It's has a really quick custom build system.
- A workspace that combines multiple projects.
- The ability to import Dev C++ projects.
- Full breakpoint support, for easier debugging.
- Syntax highlighting.
- Automatically format code to the desired style.

Step 1: Download Code::Blocks

- Go to this website: <http://www.codeblocks.org/downloads>
- Follow the link to "Download the binary release" ([direct link](#))

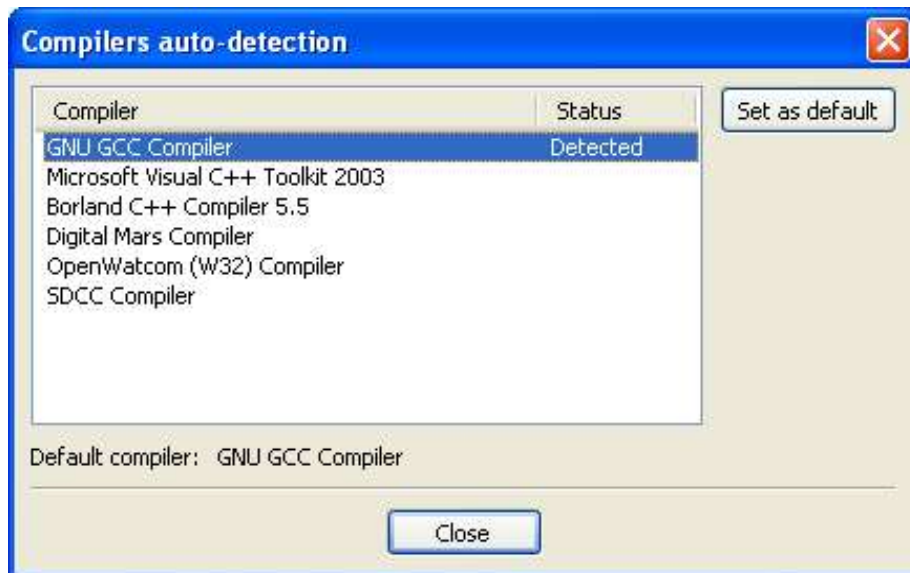
- Go to the Windows 2000 / XP / Vista / 7 section
- Look for the file that includes mingw in the name. (The name as of this writing was codeblocks-10.05mingw-setup.exe; the 10.05 may be different).
- Save the file to your desktop. It is roughly 74 megabytes.

Step 2: Install Code::Blocks

- Double click the installer.
- Hit next several times. Other setup tutorials will assume you have installed in **C:\Program Files\CodeBlocks** (the default install location), but you may install elsewhere if you like
- Do a Full Installation
- Launch Code::Blocks

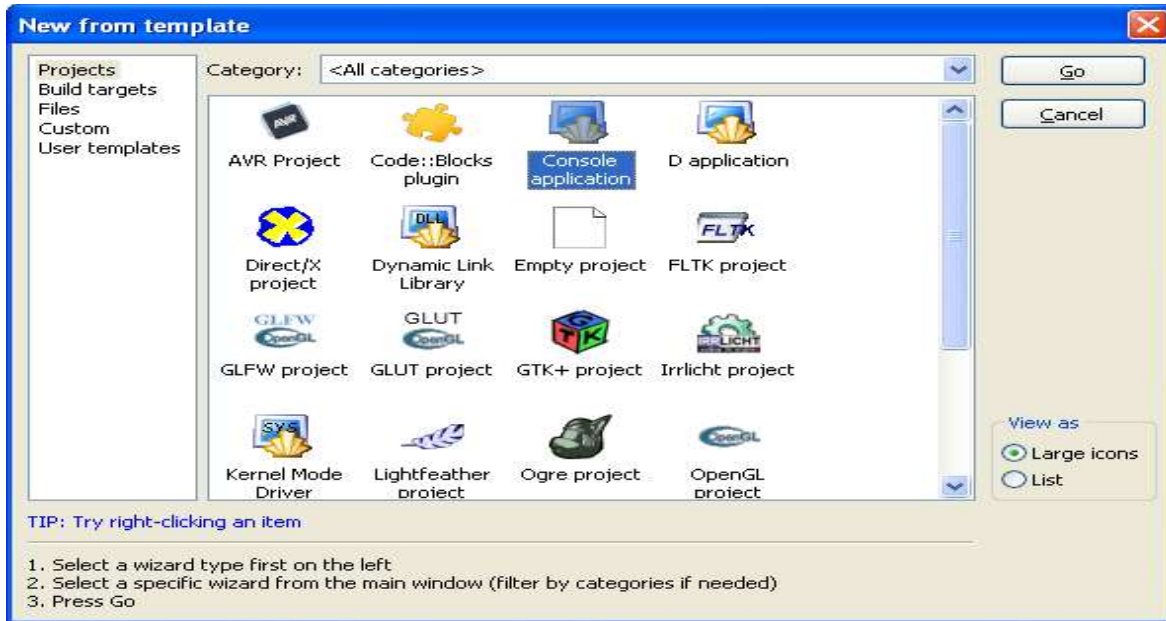
Step 3: Running in Code::Blocks

You will be prompted with a Compilers auto-detection window:



When you get the compiler auto-detection window, just hit OK. Code::Blocks may ask if you want to associate it as the default viewer for C/C++ files--I'd suggest you do. Click on the File menu, and under "New", select "Project..."

The following window will come up:



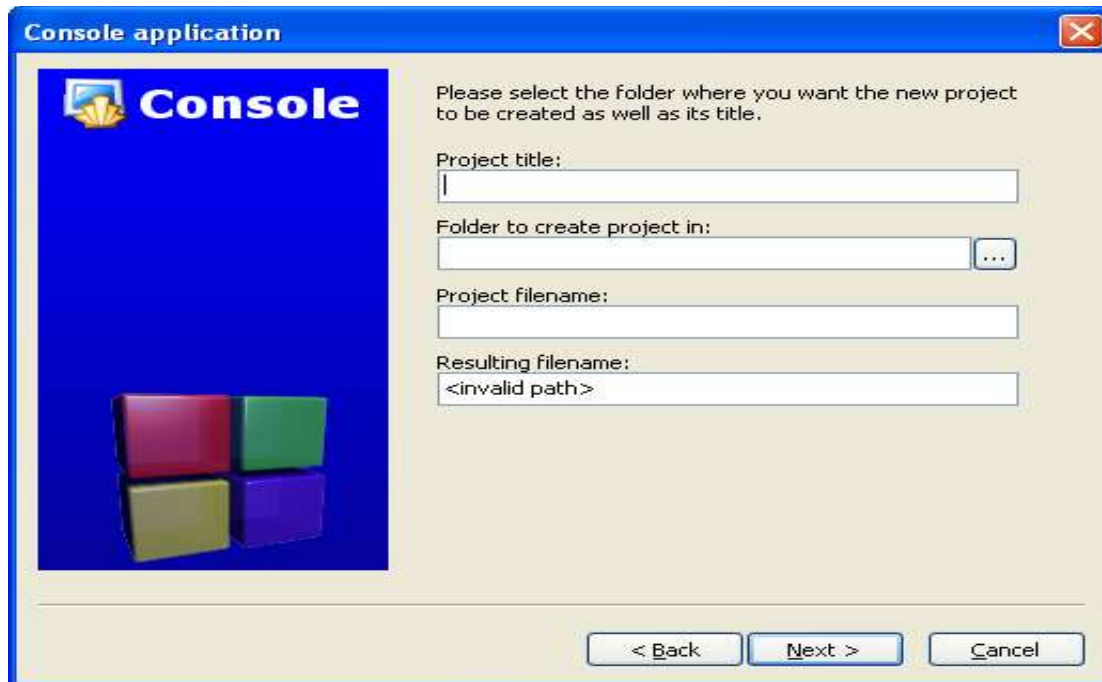
Click on "Console Application" and hit the "Go" button.

Click next until you get to the Language Selection Dialog:



You'll be asked to choose whether you want to use C or C++. If you're not sure, use C++. Otherwise, choose based on the language you are learning. (You can find tutorials here on both [C](#) and [C++](#).)

After clicking "Next", Code::Blocks will then prompt you with where you'd like to save the console application:



I'd recommend you put it in its own folder, as it may create several files (this is especially true if you create other types of projects). You will need to give your project a name, anything will be fine. Clicking "Next" again will prompt you to set up your compiler:

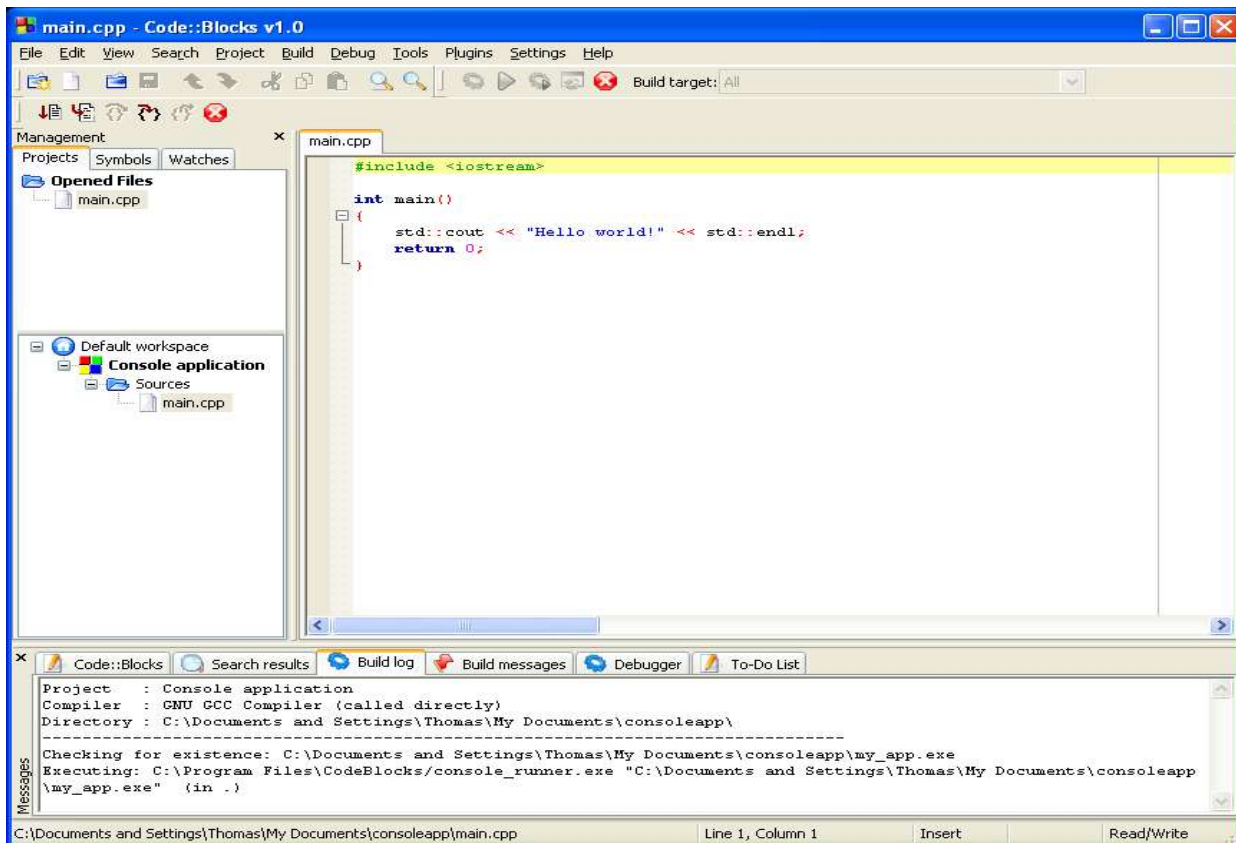


You don't need to do anything here. Just accept the defaults by hitting "Finish".

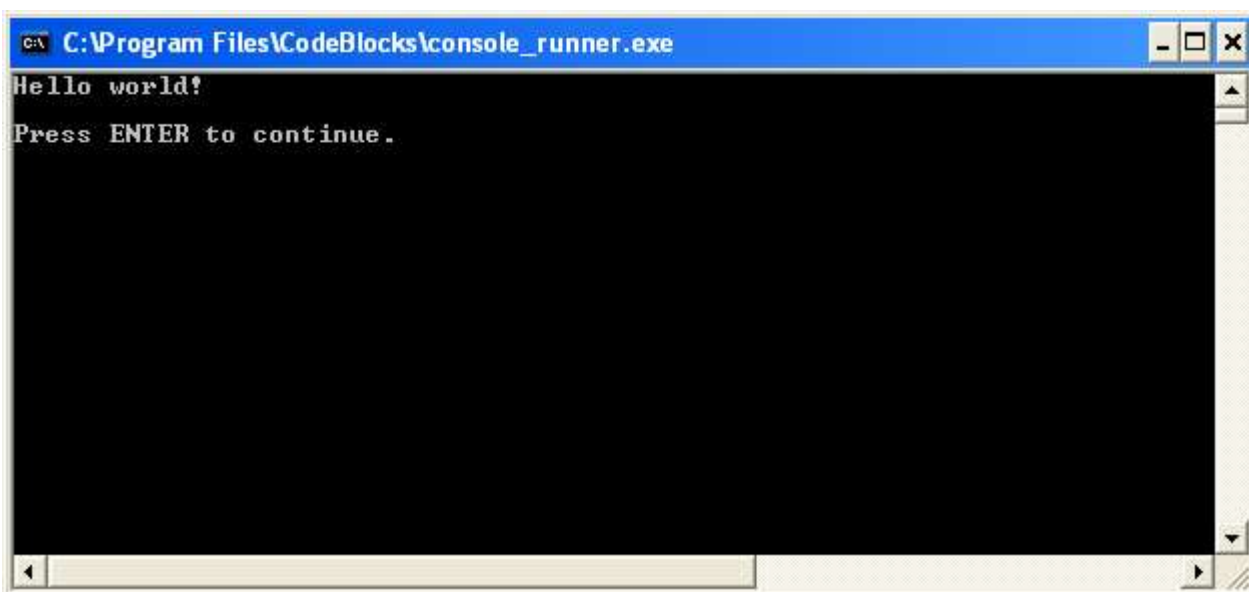
You can now open the main.cpp file on the left:

(You may need to expand the contents of the "Sources" folder if you don't see main.cpp.)

At this point, you will have your main.cpp file, which you can modify if you like.

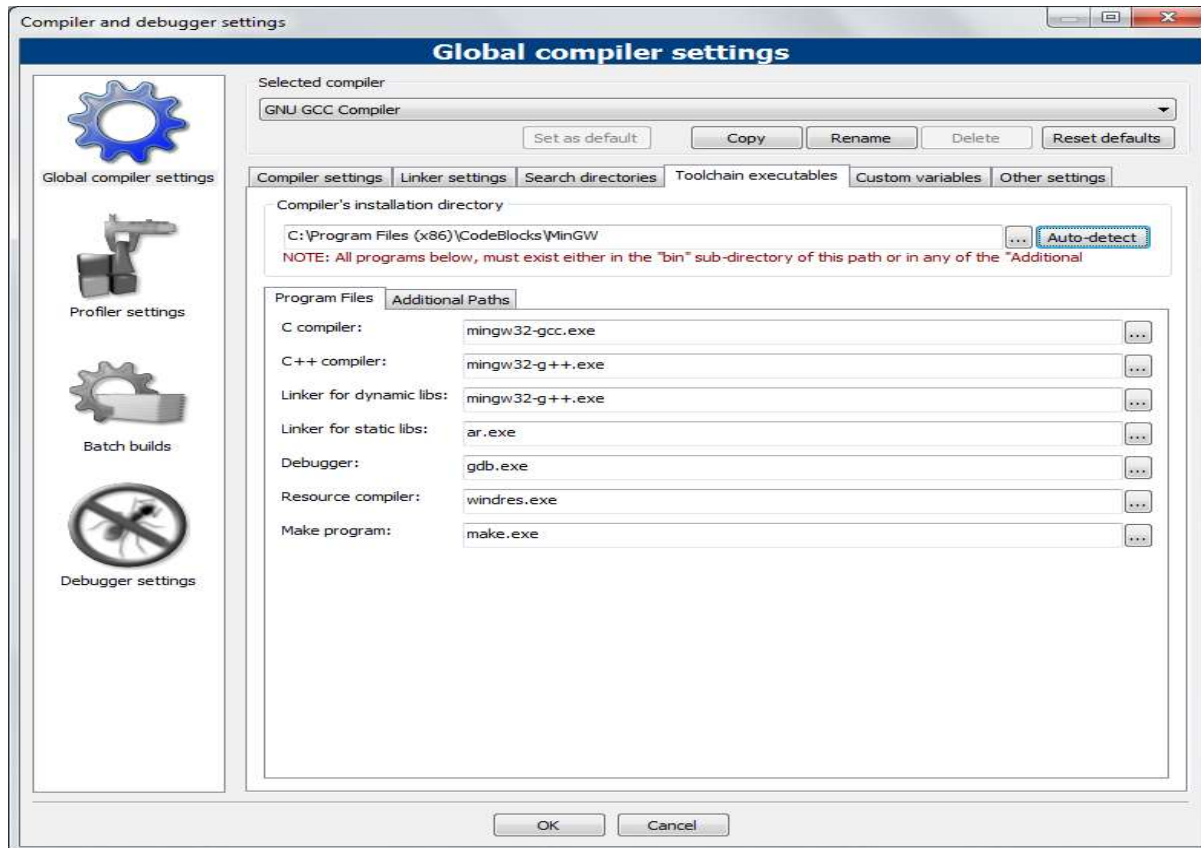


For now, it just says "Hello World!", so we can run it as is. Hit F9, which will first compile it and then run it.



You now have a running program! You can simply edit main.cpp and then hit F9 to compile it and run it again.

Now that you've finished setting your compiler up, it's time to learn to program: [Intro to C++](#) (or if you're learning C, [Intro to C](#)).



Once you've done that, try pressing F9 again to see if you get a running program.

Microsoft Visual Studio 2005

INTRODUCTION:

Microsoft Visual Studio 2005 comes in three versions:

- ◆ No Service Pack - Version 8.0.50727.42 (RTM.050727-4200)
- ◆ Service Pack 1 Beta - Version 8.0.50727.363 (SP.050727-3600)
- Service Pack 1 - Version 8.0.5727.762 (SP.050727-7600)

INSTALLATION:

Create the installation directory (for example, “C:\MSVS”).

By default, the Visual Studio installer will use an installation directory under “Program Files”; however, some products, including many Oracle products, require Visual Studio to be installed in a directory with no spaces in the path.

Insert Microsoft Visual Studio 2005 Professional Edition, Disk 1. When the “Setup” screen appears, click on the “Install Studio 2005” link. When the “Welcome” screen appears, click on “Next”.

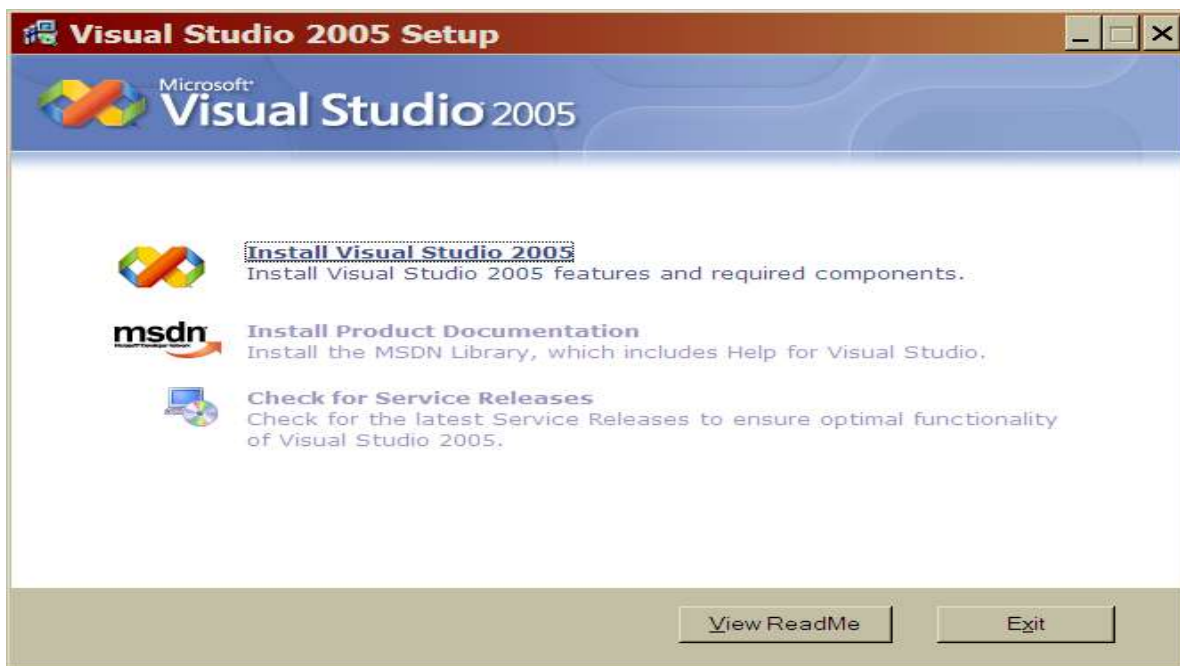


Figure 1 - Select Install



Figure 2 - Welcome

When the “Start Page” screen appears, accept the licence agreement, enter the product key, and an installation name.



Figure 3 - Enter Product Key

Click on “Next”. When the “Options Page” screen appears, select the appropriate installation features ratio button, and enter the installation product path (for example, “C:\MSVS”).

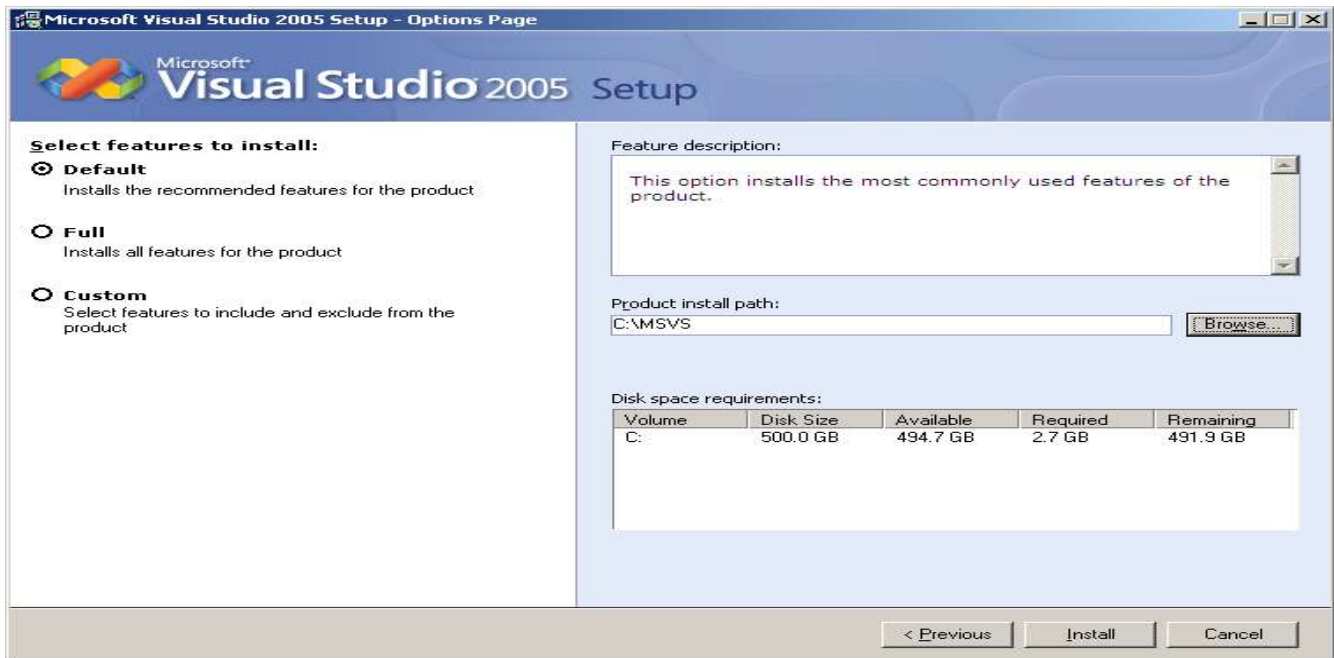


Figure 4 - Installation Path

Click on “Install”, and wait while the installation proceeds.

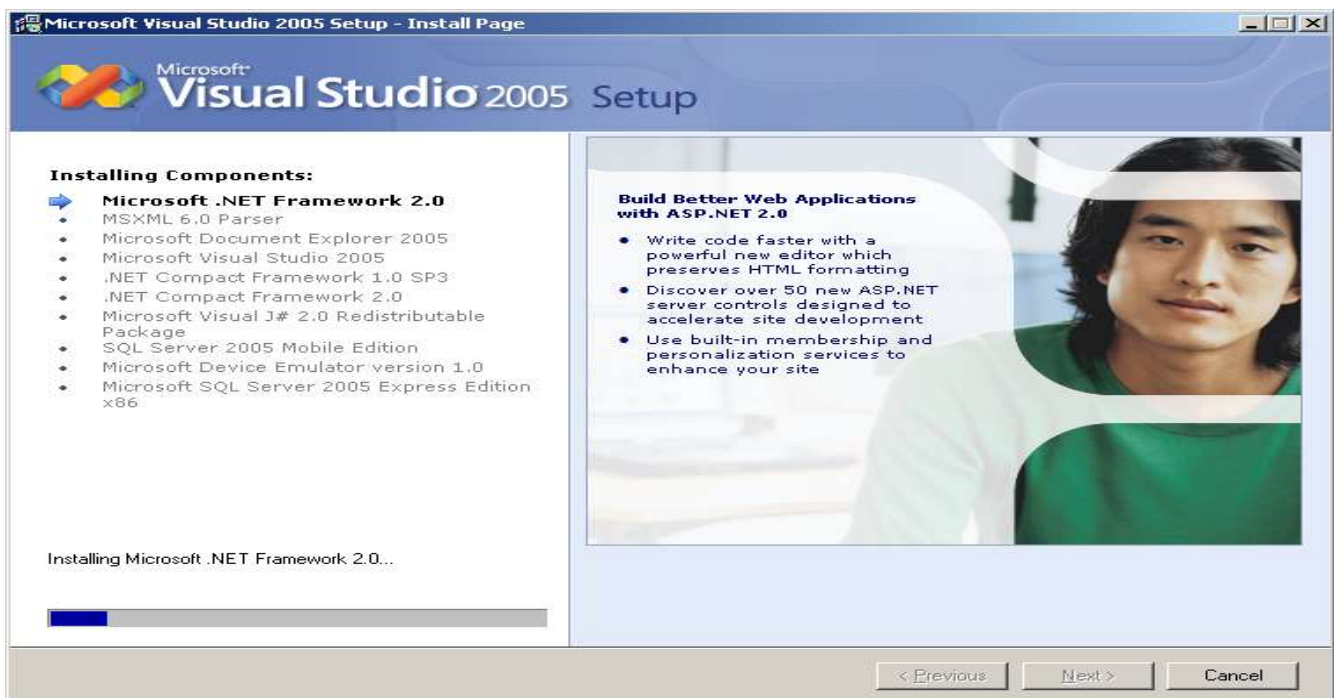


Figure 5 - Installation Progress ...

When requested, insert Microsoft Visual Studio 2005 Professional Edition, Disk 2, and click “OK” for the installation to continue (a default installation will take about 25 minutes).

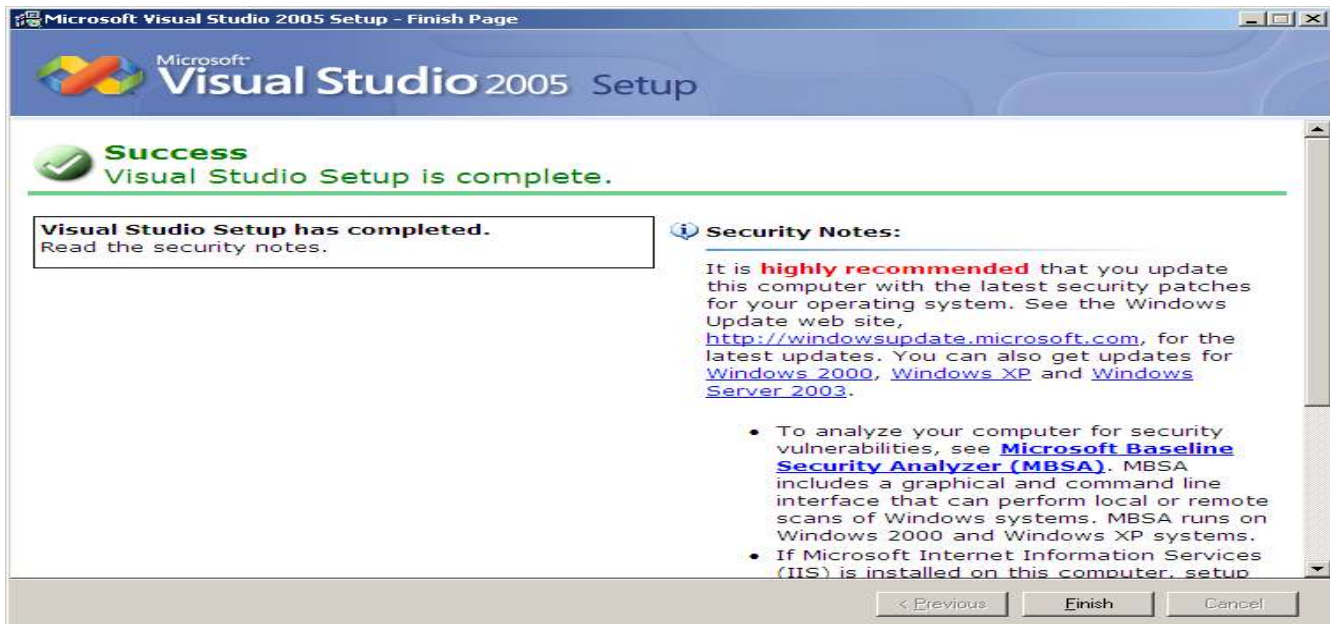


Figure 6 - Installation Completed

When the installation has finished the “Finish Page” appears. Click “Finish”. When the “Setup” screen reappears, click “Exit”.



Figure 7 - Exit Setup

UPGRADE: Navigate to the Microsoft Windows Update site, to update Visual Studio to the latest service pack, and to apply the latest set of security patches.

Text Books:

1. “Data Structure using C”, Aaron M. Tenenbaum, Yedidiah Langsam & Moshe J. Augenstein, Pearson Education/PHI, 2006.
2. “Data structures and program design in C”, 2nd Edition, Robert Kruse.
3. “Data Structures and Algorithm Analysis in C++”, Mark Allen Weiss, Third Edition, Pearson Education, 2006

Reference Books:

1. “Introduction to Data Structures in C”, Kamthane:. Pearson Education 2005.
2. “An Introduction to Data Structures with Application ”, Trembley and Sorenson, 2nd Edition, MCrawHill, 2004.
3. “Understanding Pointers In C”, 4th edition, Yashavant P. Kanetkar
4. “Data structures: A pseducode approach with C”, 2nd edition Richard F Gillberg & Behrouz A Frouzan.

List of exercise with CO Mapping

Lab	Details	COs
1.	Design a menu driven Program for the following Array operations: a. Creating an Array of N Integer Elements. b. Inserting an Element at a given valid Position. c. Deleting an Element at a given valid Position. d. Support the program with functions for each of the operations.	CO1
2.	Design a Program to perform the following operations on Two-Dimensional arrays. a. Accept matrix. b. Find the transpose of a matrix. c. To find the norm of a matrix.	CO1
3.	Design and Implement following Program to demonstrate the pointer concepts: a. The use of pointer operators and expressions. b. To swap two numbers using functions. c. Different memory allocation functions.	CO1
4.	Design, Develop and Implement a menu driven Program for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX). a. push(int): Push an Element on to Stack. b. x=pop(): Pop an Element from Stack. c. Demonstrate Empty, Overflow and Underflow situations on Stack. d. Display the status of Stack (Display the top element and position in the stack).	CO3
5.	Design, Develop and Implement a Program for the following using Recursive functions. a. To find the factorial of a number. b. To find the GCD of two numbers. c. To find X^N where X is real number and N is an integer number. d. To Solve Towers of Hanoi problem.	CO2
6.	Design, Develop and Implement a menu driven Program for the following operations on Linear QUEUE of Characters. (Array Implementation of Queue with maximum size MAX). a. Insert an Element on to Linear QUEUE. b. Delete an Element from Linear QUEUE. c. Demonstrate Overflow and Underflow situations on Linear QUEUE. d. Display the status of Linear QUEUE.	CO3

7.	Design, Develop and Implement a menu driven Program for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX). a. Insert an Element on to Circular QUEUE. b. Delete an Element from Circular QUEUE. c. Demonstrate Overflow and Underflow situations on Circular QUEUE. d. Display the status of Circular QUEUE.	CO3
8.	Design, Develop and Implement a menu driven Program for the following operations on Singly Linked List (SLL) for integer data. (<i>Students will be asked to demonstrate various combinations of insertions and deletions</i>). a. Creation of N nodes. b. Insertion [beginning, end, position]. c. Deletion [beginning, end, position]. d. Search for key. e. Display [traverse].	CO4
9.	Design, Develop and Implement a menu driven Program for the following operations on Doubly Linked List (DLL) for integer data. (<i>Students will be asked to demonstrate various combinations of insertions and deletions</i>). a. Creation of N nodes. b. Insertion [beginning, end, position]. c. Deletion [beginning, end, position]. d. Search for key. e. Display [traverse].	CO4
10.	Construct Binary Search Tree (BST) of N nodes and perform the following operations: a. Insert new node into the BST. b. Traverse the BST in Inorder, Preorder and Post Order.	CO5

Rubrics for Evaluation (CIA and Semester End Assessment)

Rubrics used for continuous evaluation in every lab session

Rubrics	Marks Allocated (100)	High	Medium	Low
Procedure	20	For given concept, algorithm and pseudo code is designed	For given concept, partial algorithm and pseudo code is designed	For given concept, low preparation before implementation but understood the concept
		15 to 20	10 to 15	0 to 10
Conduction	40	For given concept, implementation successfully done.	For given concept, implementation partially done.	For given concept, implementation still in process towards the goal.
		30 to 40	20 to 30	0 to 20
Calculation, Results, Graph	15	Desired output achieved and validation is processed.	Desired output partially achieved.	Desired output is yet to achieve.
		10 to 15	5 to 10	0 to 5
Viva/Oral	15	Student answered all the viva voce questions which includes analytical skills	Student answered to all viva voce questions	Student answered to partial viva voce questions
		10 to 15	5 to 10	0 to 5
Record Writing	10	completed record was submitted on time	Record was submitted late Late submission	Record was submitted but incomplete
		8 to 10	5 to 8	0 to 5

Assessment and Evaluation

CA (Continuous Assessment) : Every exercise is evaluated for 100 marks

Test1(Lab Internal-1) : Conducted for 100 marks (Open Ended exercise) in the middle of the semester

Test1 (Lab Internal-1) : Conducted for 100 marks (Open Ended exercise) towards end of the semester

Semester End Test : Conducted for 100 marks at the end of the semester

Rubrics used for continuous evaluation lab internals

Rubrics	Marks Allocated (100)	High	Medium	Low
Write up	30	Student is able to analyze the problem statement, design the algorithm and pseudo code with expected logic and approach.	Student is able to analyze the problem statement, but partially design the algorithm and pseudo code with expected logic and approach.	Student is able to partially analyze the problem statement, but partially design the algorithm and pseudo code with expected logic and approach.
		20 to 30	10 to 20	0 to 10
Execution / Output	50	For given concept, implementation successfully done.	For given concept, implementation partially done.	For given concept, implementation still in process towards the goal.
		40 to 50	30 to 40	0 to 30
Viva Voce	20	Student answered all the viva voce questions which includes analytical skills	Student answered to all viva voce questions	Student answered to partial viva voce questions
		15 to 20	10 to 15	0 to 10

Rubrics for Semester End Assessment (Total 100 marks)

Exercise write Up : 35 marks

Results : 35 marks

Viva-Voce : 30 mark

EXERCISE - 01

Design a menu driven program for the following Array operations

- a. **Creating Array of N Integer elements.**
- b. **Inserting an element at a given valid position.**
- c. **Deleting an element at a given valid position.**
- d. **Support the program with functions for each of the above operations.**

Array : An array is defined as finite ordered collection of homogenous data, stored in contiguous memory locations.

- finite means data range must be defined.
- ordered means data must be stored in continuous memory addresses.
- homogenous means data must be of similar data type.

Algorithm:

- Step 1: Start.
- Step 2: Read N value.
- Step 3: Read Array of N integer elements
- Step 4: Print array of N integer elements.
- Step 5: Insert an element at given valid position in an array.
- Step 6: Delete an element at given valid position from an array.
- Step 7: Stop.

Program:

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

int a[20];
int n,val,i,pos;
/*Function Prototype*/
void create();
void display();
void insert();
void delete();
int main()
{
    int choice;
    while(1)
    {
        printf("\n\n-----MENU \n");
        printf("1.CREATE\n");
```

```

printf("2.DISPLAY\n");
printf("3.INSERT\n");
printf("4.DELETE\n");
printf("5.EXIT\n");
printf(" ");
printf("\nEnter YOUR CHOICE:\t");
scanf("%d",&choice);
switch(choice)
{
    case 1: create(); break;
    case 2: display(); break;
    case 3: insert(); break;
    case 4: delete(); break;
    case 5: exit(0); break;
    default: printf("\nInvalid choice:\n");
    break;
}
}
return 0;
}
//creating an array
void create()
{
    printf("\nEnter the size of the array elements:\t");
    scanf("%d",&n);
    printf("\nEnter the elements for the array:\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
}
//displaying an array elements
void display()
{
    int i;
    printf("\nThe array elements are:\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",a[i]);
    }
}
}

```


//inserting an element into an array

void insert()

```
{
    printf("\nEnter the position for the new element(Position starts from 0 to n-1):\t");
    scanf("%d",&pos);
    printf("\nEnter the element to be inserted :\t");
    scanf("%d",&val);
    for(i=n-1;i>=pos;i--)
    {
        a[i+1]=a[i];
    }
    a[pos]=val;
    n=n+1;
}
```

//deleting an array element

void delete()

```
{
    printf("\nEnter the position of the element to be deleted(position starts from 0 to n-1):\t");
    scanf("%d",&pos);
    val=a[pos];
    for(i=pos;i<n-1;i++)
    {
        a[i]=a[i+1];
    }
    n=n-1;
    printf("\nThe deleted element is =%d",val);
}
```

OUTPUT:-

```
-----MENU
1.CREATE
2.DISPLAY
3.INSERT
4.DELETE
5.EXIT

ENTER YOUR CHOICE:      1

Enter the size of the array elements:  6

Enter the elements for the array:
11
22
44
55
66
77

-----MENU
1.CREATE
2.DISPLAY
3.INSERT
4.DELETE
5.EXIT

ENTER YOUR CHOICE:      2

The array elements are:
11      22      44      55      66      77

-----MENU
1.CREATE
2.DISPLAY
3.INSERT
4.DELETE
5.EXIT

ENTER YOUR CHOICE:
```

-----MENU

- 1.CREATE
- 2.DISPLAY
- 3.INSERT
- 4.DELETE
- 5.EXIT

ENTER YOUR CHOICE: 3

Enter the position for the new element(Position starts from 0 to n-1): 2

Enter the element to be inserted : 33

-----MENU

- 1.CREATE
- 2.DISPLAY
- 3.INSERT
- 4.DELETE
- 5.EXIT

ENTER YOUR CHOICE: 3

Enter the position for the new element(Position starts from 0 to n-1): 4

Enter the element to be inserted : 99

-----MENU

- 1.CREATE
- 2.DISPLAY
- 3.INSERT
- 4.DELETE
- 5.EXIT

ENTER YOUR CHOICE: 2

The array elements are:

11 22 33 44 99 55 66 77

-----MENU

```
ENTER YOUR CHOICE:      4

Enter the position of the element to be deleted(position starts from 0 to n-1): 5

The deleted element is =55
```

```
-----MENU
```

- 1.CREATE
- 2.DISPLAY
- 3.INSERT
- 4.DELETE
- 5.EXIT

```
ENTER YOUR CHOICE:      2
```

```
The array elements are:
```

```
11      22      33      44      99      66      77
```

```
-----MENU
```

- 1.CREATE
- 2.DISPLAY
- 3.INSERT
- 4.DELETE
- 5.EXIT

```
ENTER YOUR CHOICE:      5
```

```
Process returned 0 (0x0)   execution time : 199.960 s
Press any key to continue.
```

EXERCISE - 02

Design a Program to perform the following operations on Two-Dimensional arrays.

A. Accept matrix.

B. Find the transpose of a matrix.

C. To find the norm of a matrix.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
{
    int a[20];
    int n,m,val,i,j;
    int arr[20][20];
    /*Function Prototype*/
    void create();
    void display();
    void transpose();
    void norm();
}

int main()
{
    int choice;
    while(1)
    {
        printf("\n\n-----MENU \n");
        printf("1.CREATE THE MATRIX\n");
        printf("2.DISPLAY THE MATRIX\n");
        printf("3.TRANSPOSE OF THE MATRIX\n");
        printf("4.NORM OF THE MATRIX\n");
        printf("5.EXIT\n");
        printf(" ");
        printf("\nENTER YOUR CHOICE:\t");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: create(); break;
            case 2: display(); break;
            case 3: transpose(); break;
            case 4: norm(); break;
```

```

        case 5: exit(0); break;
        default: printf("\nInvalid choice:\n");
        break;
    }
}
return 0;
}

```

```

void create()
{
    //int m,n;          //Matrix Size Declaration
    printf("Enter the number of rows and column: \n");
    scanf("%d %d",&m,&n); //Matrix Size Initialization
    //int arr[10][10];   //Matrix Size Declaration
    printf("\nEnter the elements of the matrix: \n");
    for(i=0;i<m;i++) //Matrix Initialization
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&arr[i][j]);
        }
    }
}

```

```

void display()
{
    printf("\nThe elements in the matrix are: \n");
    for(i=0;i<m;i++) //Print the matrix
    {
        for(j=0;j<n;j++)
        {
            printf("%d\t",arr[i][j]);
        }
        printf("\n");
    }
}

```

```

void transpose()
{

```

```

int brr[10][10];    //Transpose Matrix Declaration
for(i=0;i<m;i++)    //Transpose Matrix initialization
{
    for( j=0;j<n;j++)
    {
        brr[j][i]=arr[i][j];    //Store elements in the transpose matrix
    }
}
printf("\nAfter transpose the elements are...\n");
for(i=0;i<m;i++)    //Print the transpose matrix
{
    for(j=0;j<n;j++)
    {
        printf("%d ",brr[i][j]);
    }
    printf("\n");
}
}

```

```

void norm()
{
    int sum1=0,sum=0,a=0;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            a = arr[i][j] * arr[i][j];
            sum1 = sum1 + a;
        }
    }
    int normal = sqrt(sum1);
    printf("The normal of the given matrix is = %d\n", normal);
    return 0;
void exit()
{
    exit(0);
}
}

```

OUTPUT:-

```
D:\prajwal\newproject\dspgm2\bin\Debug\dspgm2.exe

-----MENU
1.CREATE THE MATRIX
2.DISPLAY THE MATRIX
3.TRANSPOSE OF THE MATRIX
4.NORM OF THE MATRIX
5.EXIT

ENTER YOUR CHOICE: 1
Enter the number of rows and column:
3
3

Enter the elements of the matrix:
1
1
1
2
2
2
3
3
3

-----MENU
1.CREATE THE MATRIX
2.DISPLAY THE MATRIX
3.TRANSPOSE OF THE MATRIX
4.NORM OF THE MATRIX
5.EXIT

ENTER YOUR CHOICE: 2

The elements in the matrix are:
1 1 1
2 2 2
3 3 3

-----MENU
1.CREATE THE MATRIX
2.DISPLAY THE MATRIX
3.TRANSPOSE OF THE MATRIX
4.NORM OF THE MATRIX
5.EXIT

ENTER YOUR CHOICE: .
```

30°C
Partly sunny

08-04-2022 11:30

ENTER YOUR CHOICE: 3

After transpose the elements are...

1 2 3

1 2 3

1 2 3

-----MENU

1.CREATE THE MATRIX

2.DISPLAY THE MATRIX

3.TRANSPOSE OF THE MATRIX

4.NORM OF THE MATRIX

5.EXIT

ENTER YOUR CHOICE: 4

The normal of the given matrix is = 6

-----MENU

1.CREATE THE MATRIX

2.DISPLAY THE MATRIX

3.TRANSPOSE OF THE MATRIX

4.NORM OF THE MATRIX

5.EXIT

ENTER YOUR CHOICE: 5

Process returned 0 (0x0) execution time : 130.333 s

Press any key to continue.

EXERCISE - 03

Design and Implement following Program to demonstrate the pointer concepts:

- a. The use of pointer operators and expressions.
- b. To swap two numbers using functions.
- c. Different memory allocation functions.

a). The use of pointer operators and expressions.

Aim:

To implement the use of pointer operators and expression using C Program .

Algorithm:

1. Create two variables a and b.
2. Initialize the value for a and b variables.
3. Create two pointer variable to hold the address of the variable a and b.
4. Assign the address of variable a to ptr_a and address of variable b to ptr_b as follows
ptr_a = &a;
ptr_b = &b;
5. Perform the arithmetic operations on pointer variable using pointer operator and store the output of each operations in the different variables like add,sub,div,mul and mod.
6. Print the values which are stored in different variables.


Program:

```
#include <stdio.h>
int main()
{
    int a = 20, b = 10;
    int add, sub, div, mul, mod;
    int *ptr_a, *ptr_b;
    ptr_a = &a;
    ptr_b = &b;

    add = *ptr_a + *ptr_b;
    sub = *ptr_a - *ptr_b;
    mul = *ptr_a * *ptr_b;
    div = *ptr_a / *ptr_b;
    mod = *ptr_a % *ptr_b;
```

```
printf("Addition = %d\n", add);  
printf("Subtraction = %d\n", sub);  
printf("Multiplication = %d\n", mul);  
printf("Division = %d\n", div);  
printf("Modulo = %d\n", mod);  
return 0;  
}
```

OUTPUT:-

 c:\users\admin\documents\visual studio 2005\projects\wedwqf\debug\wedw

```
Addition = 30  
Subtraction = 10  
Multiplication = 200  
Division = 2  
Modulo = 0
```

B) swap two numbers using functions.

Aim:

To implement C program to Swap two numbers using functions and Pointers.

Algorithm:

1. Input two numbers from the user and store them in number1 and number2.
2. Print the value of number1 and number2 before swapping.
3. Pass the address of variable number1 and number2 to function swap().
4. Inside function swap() we take a local variable temp.
5. Take 2 pointer variables *n1 and *n2. Pointer variable n1 holds the address of number1 and pointer variable n2 holds the address of number2.
6. Use the following logic to swap:

```
temp = *n1;  
*n1 = *n2;  
*n2 = temp;
```
7. Print the value of number1 and number2 after swapping, since directly changing the value present at particular address, the value gets reflected throughout the program and not just inside particular function.

Program:

```
#include<stdio.h>

#include<conio.h>

void swap(int *, int *);

int main()
{
    int numbr1, numbr2;

    clrscr();

    printf("\n\n Enter any two Integer numbers:\n ");

    scanf("%d %d", &numbr1, &numbr2);

    printf("\n\n Before swapping :The Given Numbers are %d %d ", numbr1, numbr2);
```

```
    swap(&numbr1, &numbr2);

    printf("\n\n After Swapping: The Swapped Numbers are %d %d", numbr1, numbr2);

    getch();

    return 0;

}

void swap(int *n1, int *n2)

{

    int temp;

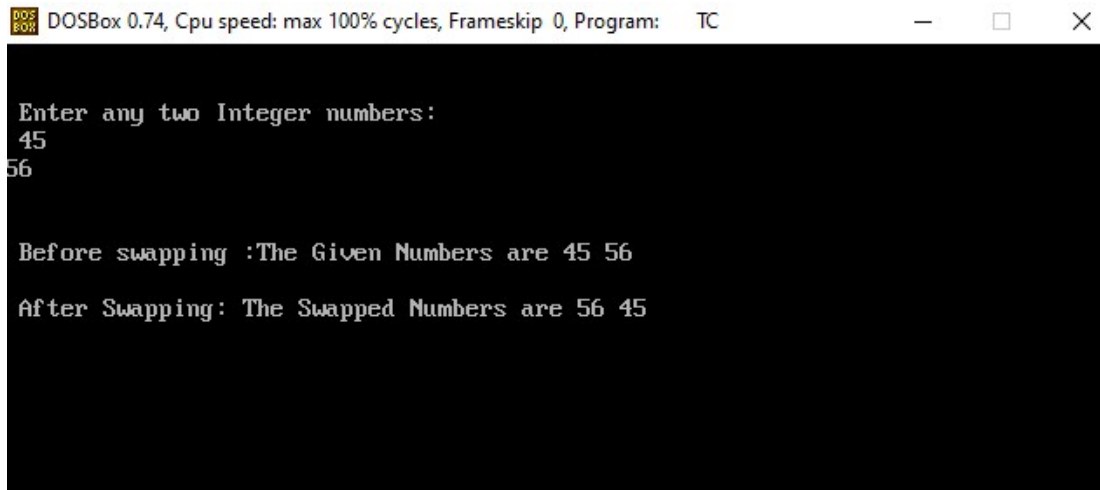
    temp = *n1;

    *n1 = *n2;

    *n2 = temp;

}
```

OUTPUT:-



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

Enter any two Integer numbers:
45
56

Before swapping :The Given Numbers are 45 56
After Swapping: The Swapped Numbers are 56 45
```

C. Different memory allocation functions.

The concept of dynamic memory allocation in c language enables the C programmer to allocate memory at runtime. Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.

1. malloc()
2. calloc()
3. realloc()
4. free()

Difference between static memory allocation and dynamic memory allocation.

static memory allocation	dynamic memory allocation
memory is allocated at compile time.	memory is allocated at run time.
memory can't be increased while executing program.	memory can be increased while executing program.
used in array.	used in linked list.

Methods used for dynamic memory allocation.

malloc()	allocates single block of requested memory.
calloc()	allocates multiple block of requested memory.
realloc()	reallocates the memory occupied by malloc() or calloc() functions.
free()	frees the dynamically allocated memory.

Aim:

To implement summation of array elements using different dynamic memory allocation using malloc(),calloc(),realloc() function and free() to deallocate .

1). malloc() functon in c:

- The malloc() function allocates single block of requested memory.
- It doesn't initialize memory at execution time, so it has garbage value initially.
- It returns NULL if memory is not sufficient.
- The syntax of malloc() function is given below:

```
ptr=(cast-type*)malloc(byte-size)
```

Algorithm:

Step 1. Input number of elements as n

Step 2.[dynamically allocate memory to declare a array]

```
Ptr=(int *) malloc(n*sizeof(int));
```

```
For(i=0 to n-1)
```

```
Input to a[i]
```

```
End for
```

Step 3. [input data and Compute sum]

```
For(i=0 to n-1]
```

```
Input to ptr+i
```

```
Sum+=*(ptr+i)
```

```
End for
```

Step 4. Output sum

Step 5. Free(ptr)

Step 6.Stop

Program:

```
#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(ptr==NULL)
```

```
{
```

```
printf("Sorry! unable to allocate memory");
```

```
exit(0);
```

```
}
```

```
printf("Enter elements of array: ");
```

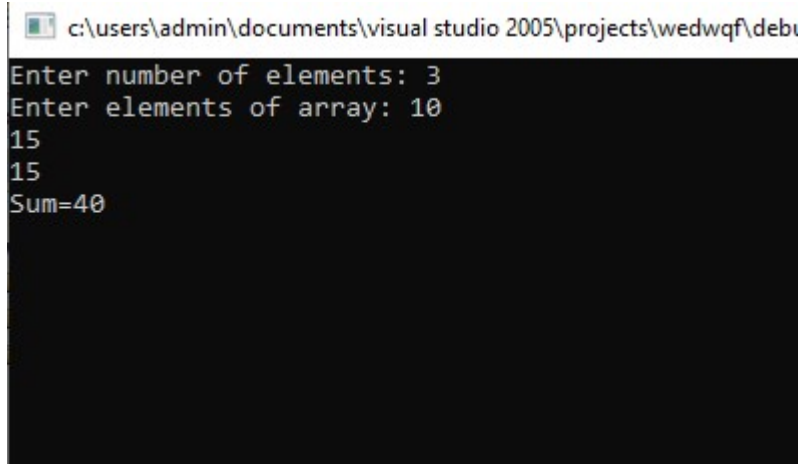
```
for(i=0;i<n;++i)
```

```

    {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
    return 0;
}

```

OUTPUT:-



```

c:\users\admin\documents\visual studio 2005\projects\wedwqf\debu
Enter number of elements: 3
Enter elements of array: 10
15
15
Sum=40

```

2.calloc() function in C:

- The calloc() function allocates multiple block of requested memory.
- It initially initialize all bytes to zero.
- It returns NULL if memory is not sufficient.
- The syntax of calloc() function is given below:

```
ptr=(cast-type*)calloc(number, byte-size)
```

Algorithm:

Step 1. Input number of elements as n

Step 2.[dynamically allocate memory to declare a array]

```
ptr=(int*)calloc(n,sizeof(int));
```

```
For(i=0 to n-1)
```

```
Input to a[i]
```

```
End for
```

Step 3. [input data and Compute sum]

```
For(i=0 to n-1]
```

```
Input to ptr+i
```

```
Sum+=*(ptr+i)
```

```
End for
```


Step 4. Output sum

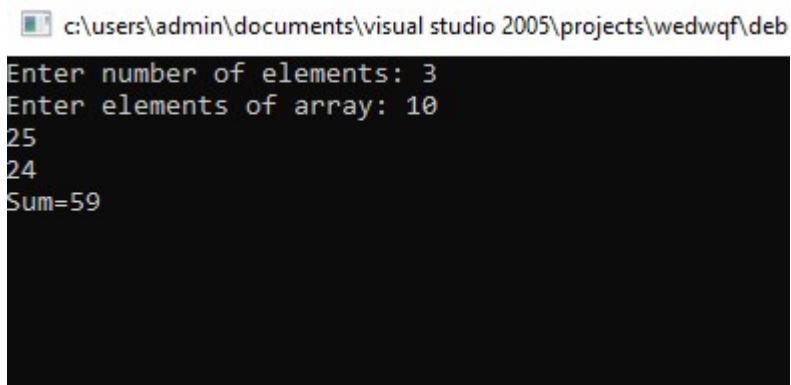
Step 5. Free(ptr)

Step 6. Stop

Program:

```
#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("Sorry! unable to allocate memory");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
    return 0;
}
```

OUTPUT:-



```
c:\users\admin\documents\visual studio 2005\projects\wedwqf\deb
Enter number of elements: 3
Enter elements of array: 10
25
24
Sum=59
```

Result:

Thus the program has been implemented for the pointer and output has been verified successfully.

EXPERIMENT – 04

Design, Develop and Implement a menu driven Program for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX).

- a. **push(int): Push an Element on to Stack.**
- b. **x=pop(): Pop an Element from Stack.**
- c. **Demonstrate Empty, Overflow and Underflow situations on Stack.**
- d. **Display the status of Stack (Display the top element and position in the stack).**

Aim:

To write a C program to implement Design, Develop and Implement a menu driven Program for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX).

Algorithm:

1. Create a Stack[] with MAX size as your wish.
2. Write function for all the basic operations of stack - PUSH(),POP()and DISPLAY().
3. By using Switch case, select push() operation to insert element in the stack.
 - Step 1: Check whether stack is FULL. (top ==SIZE-1)
 - Step 2: If it is FULL, then display "Stack is FULL!!! Insertion is not possible!!!" and terminate the function.
 - Step 3: If it is NOT FULL, then increment top value by one (top++)and set stack[top]to value (stack[top] = value).
4. Similarly, By using Switch case, select pop() operation to delete element from the stack.
 - Step 1: Check whether stack is EMPTY. (top == -1)
 - Step 2: If it is EMPTY, then display "Stack is EMPTY!!!Deletion is not possible!!!" and terminate the function.
 - Step 3: If it is NOT EMPTY, then delete stack[top] and decrement top value by one(top--).
5. Similarly, By using Switch case, select display() operation to display all elementfrom the stack.
 - Step 1: Check whether stack is EMPTY. (top ==-1)
 - Step 2: If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.
 - Step 3: If it is NOT EMPTY, then define a variable 'i' and initialize with top.

Display stack[i] value and decrement i value by one (i--).

6. Step 3: Repeat above step until i value becomes'0'.Close the program.

Program:

```
#include<stdio.h>
int stack[100],choice,n,top,x,i;void
push(void);
void pop(void); void
display(void);
int main()
{
    top=-1;
    printf("\n Enter the size of STACK[MAX=100]:");
    scanf("%d",&n);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t.");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
    do
    {
        printf("\n Enter the Choice:");
        scanf("%d",&choice); switch(choice)
        {
            case 1:
                {
                    push(); break;
                }
            case 2:
                {
                    pop();break;
                }
            case 3:
                {
                    display();break;
                }
            case 4:
                {
                    printf("\n\t EXIT POINT ");
                    break;
                }
        }
    }
}
```

```

        }
        default:
        {
            printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
        }
    }
}
while(choice!=4);
return 0;
}

void push()
{
    if(top>=n-1)
    {
        printf("\n\tSTACK is over flow");
    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}

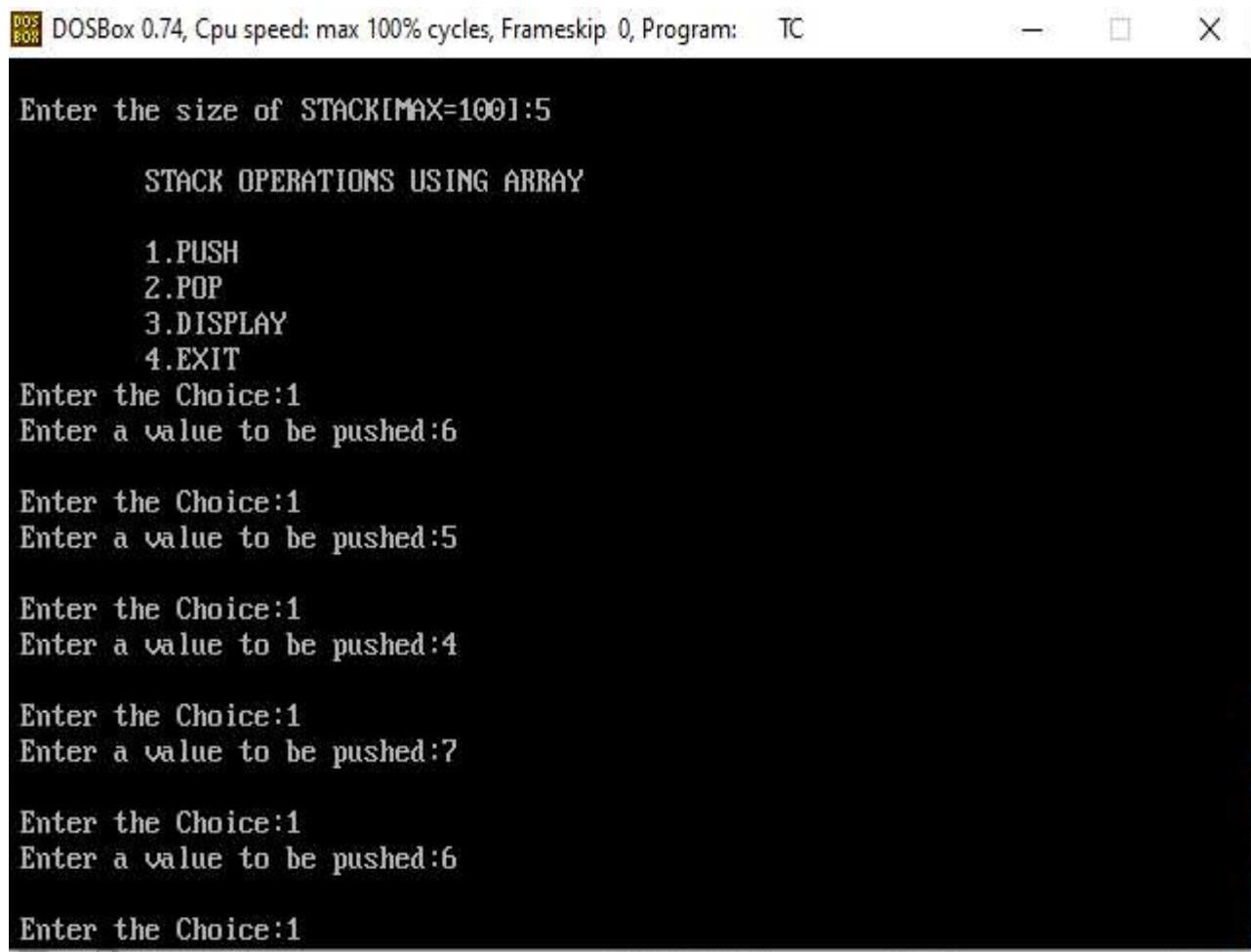
void pop()
{
    if(top<=-1)
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}

void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK \n");
        for(i=top; i>=0; i--)

```

```
    printf("\n%d",stack[i]);  
    printf("\n Press Next Choice");  
}  
else  
{  
    printf("\n The STACK is empty");  
}  
}
```

OUTPUT:-



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the size of STACK[MAX=100]:5  
  
    STACK OPERATIONS USING ARRAY  
  
    1.PUSH  
    2.POP  
    3.DISPLAY  
    4.EXIT  
Enter the Choice:1  
Enter a value to be pushed:6  
  
Enter the Choice:1  
Enter a value to be pushed:5  
  
Enter the Choice:1  
Enter a value to be pushed:4  
  
Enter the Choice:1  
Enter a value to be pushed:7  
  
Enter the Choice:1  
Enter a value to be pushed:6  
  
Enter the Choice:1
```

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

STACK is over flow
Enter the Choice:3

The elements in STACK
6
7
4
5
6
Press Next Choice
Enter the Choice:2

The popped elements is 6
Enter the Choice:2

The popped elements is 7
Enter the Choice:2

The popped elements is 4
Enter the Choice:2

The popped elements is 5
Enter the Choice:
```

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
6
Press Next Choice
Enter the Choice:2

The popped elements is 6
Enter the Choice:2

Stack is under flow
Enter the Choice:3

The STACK is empty
Enter the Choice:1
Enter a value to be pushed:6

Enter the Choice:1
Enter a value to be pushed:2

Enter the Choice:3

The elements in STACK
2
6
Press Next Choice
Enter the Choice:
```

Result:

Thus to implement Design, Develop and Implement a menu driven Program for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX). program was created and executed successfully.

EXERCISE – 05

Design, Develop and Implement a Program for the following using Recursive functions.

- a. To find the factorial of a number.
- b. To find the GCD of two numbers.
- c. To find X^N where X is real number and N is an integer number.
- d. To Solve Towers of Hanoi problem.

a) To find the factorial of a number.

Algorithm:

Step 1: Start

Step 2: Declare Variable n, fact, i

Step 3: Read number from User

Step 4: Initialize Variable fact=1 and i=1

Step 5: Repeat Until $i \leq \text{number}$

5.1 fact=fact*i

5.2 i=i+1

Step 6: Print fact

Step 7: Stop

Program:

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

```
int main()
```

```
{
```

```
    int n, i;
```

```
    unsigned long long fact = 1;
```

```
    printf("Enter an integer: ");
```

```
    scanf("%d", &n);
```

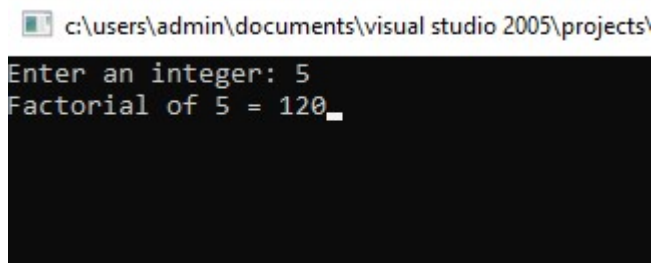
```
    // shows error if the user enters a negative integer
```

```
    if (n < 0)
```

```
        printf("Error! Factorial of a negative number doesn't exist.");
```

```
else
{
    for (i = 1; i <= n; i++)
    {
        fact=fact* i;
    }
    printf("Factorial of %d = %llu", n, fact);
}
getch();
return 0;
}
```

OUTPUT:-



```
c:\users\admin\documents\visual studio 2005\projects\
Enter an integer: 5
Factorial of 5 = 120_
```

b)To find the GCD of two numbers.

Algorithm:

Step 1: Start

Step 2: Declare variable n1, n2, gcd=1, i=1

Step 3: Input n1 and n2

Step 4: Repeat until $i \leq n1$ and $i \leq n2$

Step 4.1: If $n1 \% i == 0$ && $n2 \% i == 0$:

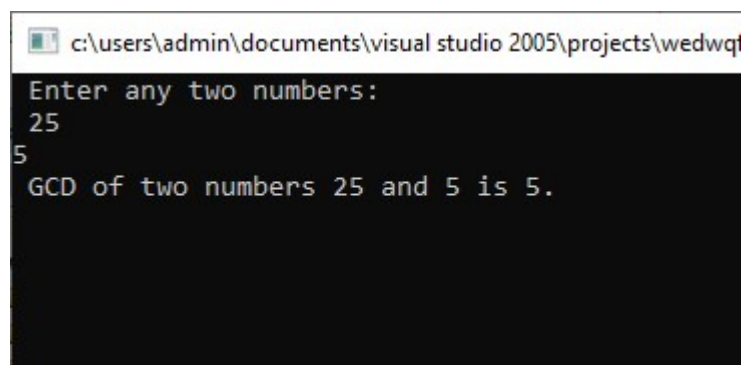
Step 4.2: gcd = i

Step 5: Print gcd

Step 6: Stop

Program:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    // declare the variables
    int n1, n2, i, GCD_Num;
    printf ( " Enter any two numbers: \n ");
    scanf ( "%d %d", &n1, &n2);
    // use for loop
    for( i = 1; i <= n1 && i <= n2; ++i)
    {
        if (n1 % i == 0 && n2 % i == 0)
            GCD_Num = i;
        /* if n1 and n2 is completely divisible by i, the divisible
number will be the GCD_Num */
    }
    // print the GCD of two numbers
    printf (" GCD of two numbers %d and %d is %d.", n1, n2,
GCD_Num);
    getch();
    return 0;
}
```

OUTPUT:-

The screenshot shows a Windows command prompt window with the title bar "c:\users\admin\documents\visual studio 2005\projects\wedwq". The prompt displays the output of the C program: "Enter any two numbers:", followed by the user input "25" and "5" on separate lines. The final output line is "GCD of two numbers 25 and 5 is 5."

C) To find X^N where X is real number and N is an integer number.

Algorithm:

Step 1: Start

Step 2: Declare variable base, exp, result;

Step 3: Input base and exp

Step 4: use pow(base, exp) function

Step 5: Print result

Step 6: Stop

Program:

```
#include <math.h>
```

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

```
int main()
```

```
{
```

```
    double base, exp, result;
```

```
    printf("Enter a base number: ");
```

```
    scanf("%lf", &base);
```

```
    printf("Enter an exponent: ");
```

```
    scanf("%lf", &exp);
```

```
    // calculates the power
```

```
    result = pow(base, exp);
```

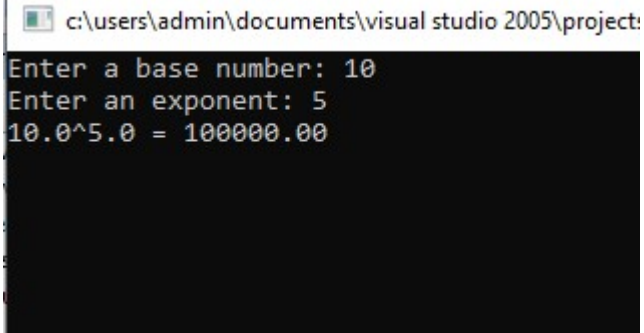
```
    printf("%.1lf^%.1lf = %.2lf", base, exp, result);
```

```
    getch();
```

```
    return 0;
```

```
}
```

OUTPUT:-



```
c:\users\admin\documents\visual studio 2005\project:
Enter a base number: 10
Enter an exponent: 5
10.0^5.0 = 100000.00
```

d)To Solve Towers of Hanoi problem.

Algorithm:

Step 1: Start.

Step 2: Read N number of discs.

Step 3: Move all the discs from source to destination by using temp rod.

Step 4: Stop.

Program:

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

```
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
```

```
{
```

```
    if (n == 1)
```

```
    {
```

```
        printf("\n Move disk 1 from rod %c to rod %c", from_rod, to_rod);
```

```
        return;
```

```
    }
```

```
        towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
```

```
        printf("\n Move disk %d from rod %c to rod %c", n, from_rod, to_rod);
```


```
        towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
```

```
    }
```

```
int main()
```

```
{  
    int n = 4; // Number of disks  
    towerOfHanoi(n,'A', 'C', 'B'); // A, B and C are names of rods  
    return 0;  
}
```

OUTPUT:-

 c:\users\admin\documents\visual studio 2005\projects\w

```
Move disk 1 from rod A to rod B  
Move disk 2 from rod A to rod C  
Move disk 1 from rod B to rod C  
Move disk 3 from rod A to rod B  
Move disk 1 from rod C to rod A  
Move disk 2 from rod C to rod B  
Move disk 1 from rod A to rod B  
Move disk 4 from rod A to rod C  
Move disk 1 from rod B to rod C  
Move disk 2 from rod B to rod A  
Move disk 1 from rod C to rod A  
Move disk 3 from rod B to rod C  
Move disk 1 from rod A to rod B  
Move disk 2 from rod A to rod C  
Move disk 1 from rod B to rod C
```

EXERCISE – 06

Design, Develop and Implement a menu driven Program for the following operations on Linear QUEUE of Characters. (Array Implementation of Queue with maximum size MAX).

- a. Insert an Element on to Linear QUEUE.**
- b. Delete an Element from Linear QUEUE.**
- c. Demonstrate Overflow and Underflow situations on Linear QUEUE.**
- d. Display the status of Linear QUEUE.**

ALGORITHM:

Step 1: Start.

Step 2: Initialize queue size to MAX.

Step 3: Insert the elements into queue. If queue is full give a message as ‘queue is overflow’

Step 4: Delete an element from the queue. If queue is empty give a message as ‘queue is Underflow’.

Step 5: Display the contents of the queue.

Step 6: Stop.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
    int choice;
    while (1)
```

```

{
    printf("1.Insert element to queue \n");
    printf("2.Delete element from queue \n");
    printf("3.Display all elements of queue \n");
    printf("4.Quit \n");
    printf("Enter your choice : ");
    scanf("%d", &choice);
    switch (choice)
    {
        case 1:insert();
            break;
        case 2: delete();
            break;
        case 3: display();
            break;
        case 4: exit(1);
            default:
                printf("Wrong choice \n");
    } /* End of switch */
} /* End of while */
} /* End of main() */

```

```

void insert()
{
    int add_item;
    if (rear == MAX-1)
        printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
            /*If queue is initially empty */
            front = 0;

```

```

        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
} /* End of insert() */


void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
} /* End of delete() */

void display()
{
    int i;
    if (front == - 1 || front > rear)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
}

```

```
}  
} /* End of display() */
```

OUTPUT:-

 c:\users\admin\documents\visual studio 2005\projects\wedwqf\debu

```
1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit  
Enter your choice : 1  
Inset the element in queue : 10  
1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit  
Enter your choice : 1  
Inset the element in queue : 20  
1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit  
Enter your choice : 1  
Inset the element in queue : 30  
1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit  
Enter your choice : 1  
Queue Overflow  
1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit  
Enter your choice : 3  
Queue is :  
10 20 30
```



```
Queue is :  
10 20 30  
1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit  
Enter your choice : 2  
Element deleted from queue is : 10  
1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit  
Enter your choice : 2  
Element deleted from queue is : 20  
1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit  
Enter your choice : 2  
Element deleted from queue is : 30  
1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit  
Enter your choice : 2  
Queue Underflow  
1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit  
Enter your choice : 3  
Queue is empty  
1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit  
Enter your choice :
```

EXERCISE – 07

Design, Develop and Implement a menu driven Program for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX).

- a) Insert an Element on to Circular QUEUE.**
- b) Delete an Element from Circular QUEUE.**
 - d) Demonstrate Overflow and Underflow situations on Circular QUEUE.**
 - e) Display the status of Circular QUEUE.**

Circular queue is a linear data structure. It follows FIFO principle. In circular queue the last node is connected back to the first node to make a circle. Circular linked list follow the First In First Out principle. Elements are added at the rear end and the elements are deleted at front end of the queue. The queue is considered as a circular queue when the positions 0 and MAX-1 are adjacent. Any position before front is also after rear.

ALGORITHM:

Step 1: Start.

Step 2: Initialize queue size to MAX.

Step 3: Insert the elements into circular queue. If queue is full give a message as „queue is overflow” Step 4: Delete an element from the circular queue. If queue is empty give a message as „queue is underflow”.

Step 5: Display the contents of the queue. Step 6: Stop.

Program:

```
#include <stdio.h>
#include <conio.h>
#define SIZE 5
int CQ[SIZE];
int front=-1;
int rear=-1, ch;
int IsCQ_Full();
int IsCQ_Empty();
void CQ_Insert(int );
void CQ_Delet();
void CQ_Display();
```

```

void main()
{
    printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
    while(1)
    {
        int ele;
        printf("Enter your choice\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: if(IsCQ_Full())
                printf("Circular Queu Overflow\n");
            else
            {
                printf("Enter the element to be inserted\n");
                scanf("%d",&ele);
                CQ_Insert(ele);
            }
            break;
            case 2: if(IsCQ_Empty())
                printf("Circular Queue Underflow\n");
            else
                CQ_Delet();
            break;
            case 3: if(IsCQ_Empty())
                printf("Circular Queue Underflow\n");
            else
                CQ_Display();
            break;
            case 4: exit(0);
        }
    }
}

void CQ_Insert(int item)
{
    if(front==-1)front++;
    rear = (rear+1)%SIZE;
    CQ[rear] =item;
}

void CQ_Delet()
{
    int item; item=CQ[front];
    printf("Deleted element is: %d",item);
    front = (front+1)%SIZE;
}

void CQ_Display()
{

```

```

    int i;
    if(front==-1)
        printf("Circular Queue is Empty\n");
    else
    {
        printf("Elements of the circular queue are..\n");
        for(i=front;i!=rear;
            i=(i+1)%SIZE);
        {
            printf("%d\t",CQ[i]);
        }
        printf("%d\n",CQ[i]);
    }
}
int IsCQ_Full()
{
    if(front ==(rear+1)%SIZE)return 1;
    return 0;
}
int IsCQ_Empty()
{
    if(front == -1)
        return 1;
    else if(front == rear)
    {
        printf("Deleted element is: %d",CQ[front]);front=-1;
        return 1;
    }
    return 0;
}
}

```

OUTPUT:-

```
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice
1
Enter the element to be inserted
10
Enter your choice
1
Enter the element to be inserted
20
Enter your choice
1
Enter the element to be inserted
30
Enter your choice
3
Elements of the circular queue are..
30    30
Enter your choice
2
Deleted element is: 10Enter your choice
10
Enter your choice
3
Elements of the circular queue are..
30    30
Enter your choice
2
Deleted element is: 20Enter your choice
20
Enter your choice
3
Deleted element is: 30Circular Queue Underflow
Enter your choice
_
```

EXERCISE – 08

Design, Develop and Implement a menu driven Program for the following operations on Singly Linked List (SLL) for integer data. (Students will be asked to demonstrate various combinations of insertions and deletions).

- a. Creation of N nodes.
- b. Insertion [beginning, end, position].
- c. Deletion [beginning, end, position].
- d. Search for key.
- e. Display [traverse].

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
struct dlinklist
{
    struct dlinklist *left;
    int data;
    struct dlinklist *right;
};
typedef struct dlinklist node;
node *start = NULL;
node* getnode()
{
    node * newnode;
    newnode = (node *) malloc(sizeof(node));
    printf("\n Enter data: ");
    scanf("%d", &newnode -> data);
    newnode -> left = NULL;
    newnode -> right = NULL;
    return newnode;
}
int countnode(node *start)
{
    if(start == NULL)
        return 0;
    else
        return 1 + countnode(start -> right);
}
int menu()
{
    int ch;
```

```

        clrscr();
        printf("\n 1.Create");
        printf("\n-----");
        printf("\n 2. Insert a node at beginning ");
        printf("\n 3. Insert a node at end");
        printf("\n 4. Insert a node at middle");
        printf("\n-----");
        printf("\n 5. Delete a node from beginning");
        printf("\n 6. Delete a node from Last");
        printf("\n 7. Delete a node from Middle");
        printf("\n-----");
        printf("\n 8. Traverse the list from Left to Right");
        printf("\n 9. Traverse the list from Right toLeft ");
        printf("\n-----");
        printf("\n 10.Count the Number of nodes in the list");
        printf("\n 11.Exit ");
        printf("\n\n Enter your choice: ");
        scanf("%d", &ch);
        return ch;
    }
void createlist(int n)
{
    int i;
    node *newnode;
    node *temp;
    for(i = 0; i < n; i++)
    {
        newnode = getnode();
        if(start == NULL)
            start = newnode;
        else
        {
            temp = start;
            while(temp -> right)
                temp = temp -> right;
            temp -> right = newnode;
            newnode -> left = temp;
        }
    }
}

void traverse_left_to_right()
{
    node *temp;
    temp = start;
    printf("\n The contents of List: ");

```

```

    if(start == NULL )
    printf("\n Empty List");
    else
    {
        while(temp != NULL)
        {
            printf("\t %d ", temp -> data);
            temp = temp -> right;
        }
    }
}

void traverse_right_to_left()
{
    node *temp;
    temp = start;
    printf("\n The contents of List: ");
    if(start == NULL)
    printf("\n Empty List");
    else
    {
        while(temp -> right != NULL)
            temp = temp -> right;
    }
    while(temp != NULL)
    {
        printf("\t %d", temp -> data);
        temp = temp -> left;
    }
}

void dll_insert_beg()
{
    node *newnode;
    newnode = getnode();
    if(start == NULL)
    start = newnode;
    else
    {
        newnode -> right = start;
        start -> left = newnode;
        start = newnode;
    }
}

void dll_insert_end()
{
    node *newnode, *temp;
    newnode = getnode();

```



```

    if(start == NULL)
    start = newnode;
    else
    {
        temp = start;
        while(temp -> right != NULL)
        temp = temp -> right;
        temp -> right = newnode;
        newnode -> left = temp;
    }
}

void dll_insert_mid()
{
    node *newnode,*temp;
    int pos, nodectr, ctr = 1;
    newnode = getnode();
    printf("\n Enter the position: ");
    scanf("%d", &pos);
    nodectr = countnode(start);
    if(pos - nodectr >= 2)
    {
        printf("\n Position is out of range..");
        return;
    }
    if(pos > 1 && pos < nodectr)
    {
        temp = start;
        while(ctr < pos - 1)
        {
            temp = temp -> right;
            ctr++;
        }
        newnode -> left = temp;
        newnode -> right = temp -> right;
        temp -> right -> left = newnode;
        temp -> right = newnode;
    }
    else
        printf("position %d of list is not a middle position ", pos);
}

void dll_delete_beg()
{
    node *temp;
    if(start == NULL)
    {
        printf("\n Empty list");
    }
}

```

```

        getch();
        return ;
    }
    else
    {
        temp = start;
        start = start -> right;
        start -> left = NULL;
        free(temp);
    }
}
void dll_delete_last()
{
    node *temp;
    if(start == NULL)
    {
        printf("\n Empty list");
        getch();
        return ;
    }
    else
    {
        temp = start;
        while(temp -> right != NULL)

            temp = temp -> right;
        temp -> left -> right = NULL;
        free(temp);
        temp = NULL;
    }
}
void dll_delete_mid()
{
    int i = 0, pos, nodectr;
    node *temp;
    if(start == NULL)
    {
        printf("\n Empty List");
        getch();
        return;
    }
    else
    {
        printf("\n Enter the position of the node to delete: ");
        scanf("%d", &pos);
        nodectr = countnode(start);
    }
}

```

```

        if(pos > nodectr)
        {
            printf("\nthis node does not exist");
            getch();
            return;
        }
        if(pos > 1 && pos < nodectr)
        {
            temp = start;
            i = 1;
            while(i < pos)
            {
                temp = temp -> right;
                i++;
            }
            temp -> right -> left = temp -> left;
            temp -> left -> right = temp -> right;
            free(temp);
            printf("\n node deleted..");
        }
        else
        {
            printf("\n It is not a middle position..");
            getch();
        }
    }
}

void main(void)
{
    int ch, n;
    clrscr();
    while(1)
    {
        ch = menu();
        switch( ch)
        {
            case 1 :
                printf("\n Enter Number of nodes to create: ");
                scanf("%d", &n);
                createlist(n);
                printf("\n List created..");
                break;
            case 2 :
                dll_insert_beg();
                break;
            case 3 :

```

```
        dll_insert_end();
        break;
        case 4 :
        dll_insert_mid();
        break;
        case 5 :
        dll_delete_beg();
        break;
        case 6 :
        dll_delete_last();
        break;
        case 7 :
        dll_delete_mid();
        break;
        case 8 :
        traverse_left_to_right();
        break;
        case 9 :
        traverse_right_to_left();
        break;
        case 10 :
        printf("\n Number of nodes: %d", countnode(start));
        break;
        case 11:
        exit(0);
    }
    getch();
}
```

OUTPUT: -

```
1.Create
-----
2. Insert a node at beginning
3. Insert a node at end
4. Insert a node at middle
-----
5. Delete a node from beginning
6. Delete a node from Last
7. Delete a node from Middle
-----
8. Traverse the list from Left to Right
9. Traverse the list from Right toLeft
-----
10.Count the Number of nodes in the list
11.Exit

Enter your choice: 1

Enter Number of nodes to create: 4

Enter data: 3

Enter data: 4_
```

```
4. Insert a node at middle
-----
5. Delete a node from beginning
6. Delete a node from Last
7. Delete a node from Middle
-----
8. Traverse the list from Left to Right
9. Traverse the list from Right toLeft
-----
10.Count the Number of nodes in the list
11.Exit

Enter your choice: 1

Enter Number of nodes to create: 4

Enter data: 3

Enter data: 5

Enter data: 6

Enter data: 7

List created.._
```

1.Create

- 2. Insert a node at beginning
- 3. Insert a node at end
- 4. Insert a node at middle

- 5. Delete a node from beginning
- 6. Delete a node from Last
- 7. Delete a node from Middle

- 8. Traverse the list from Left to Right
- 9. Traverse the list from Right toLeft

- 10.Count the Number of nodes in the list
- 11.Exit

Enter your choice: 8

The contents of List: 3 5 6 7 _

1.Create

- 2. Insert a node at beginning
- 3. Insert a node at end
- 4. Insert a node at middle

- 5. Delete a node from beginning
- 6. Delete a node from Last
- 7. Delete a node from Middle

- 8. Traverse the list from Left to Right
- 9. Traverse the list from Right toLeft

- 10.Count the Number of nodes in the list
- 11.Exit

Enter your choice: 4

Enter data: 45

Enter the position: 3

1.Create

- 2. Insert a node at beginning
- 3. Insert a node at end
- 4. Insert a node at middle

- 5. Delete a node from beginning
- 6. Delete a node from Last
- 7. Delete a node from Middle

- 8. Traverse the list from Left to Right
- 9. Traverse the list from Right toLeft

- 10.Count the Number of nodes in the list
- 11.Exit

Enter your choice: 2

Enter data: 34

-

1.Create

- 2. Insert a node at beginning
- 3. Insert a node at end
- 4. Insert a node at middle

- 5. Delete a node from beginning
- 6. Delete a node from Last
- 7. Delete a node from Middle

- 8. Traverse the list from Left to Right
- 9. Traverse the list from Right toLeft

- 10.Count the Number of nodes in the list
- 11.Exit

Enter your choice: 8

The contents of List: 34 3 5 45 6 7 _

1.Create

- 2. Insert a node at beginning
 - 3. Insert a node at end
 - 4. Insert a node at middle
-

- 5. Delete a node from beginning
 - 6. Delete a node from Last
 - 7. Delete a node from Middle
-

- 8. Traverse the list from Left to Right
 - 9. Traverse the list from Right toLeft
-

- 10.Count the Number of nodes in the list
- 11.Exit

Enter your choice: 10

Number of nodes: 6_

1.Create

- 2. Insert a node at beginning
 - 3. Insert a node at end
 - 4. Insert a node at middle
-

- 5. Delete a node from beginning
 - 6. Delete a node from Last
 - 7. Delete a node from Middle
-

- 8. Traverse the list from Left to Right
 - 9. Traverse the list from Right toLeft
-

- 10.Count the Number of nodes in the list
- 11.Exit

Enter your choice:

7

Enter the position of the node to delete: 2

node deleted.._

1.Create

2. Insert a node at beginning

3. Insert a node at end

4. Insert a node at middle

5. Delete a node from beginning

6. Delete a node from Last

7. Delete a node from Middle

8. Traverse the list from Left to Right

9. Traverse the list from Right toLeft

10.Count the Number of nodes in the list

11.Exit

Enter your choice:

9

The contents of List: 7 6 45 5 34

EXERCISE – 09

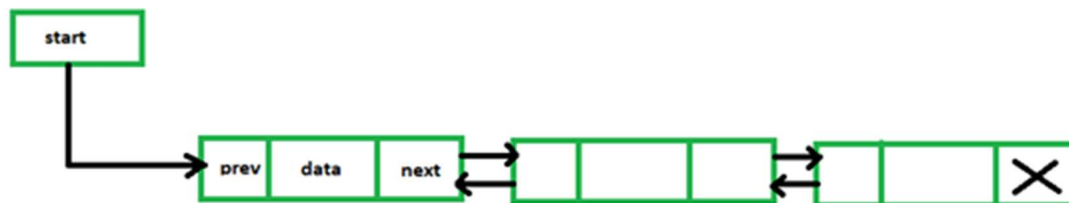
Design, Develop and Implement a menu driven Program for the following operations on Doubly Linked List (DLL) for integer data. (*Students will be asked to demonstrate various combinations of insertions and deletions*).

- a) Creation of N nodes.
- b) Insertion [beginning, end, position].
- c) Deletion [beginning, end, position].
- d) Search for key.
- e) Display [traverse].

Doubly Linked List:

A linked list (singly linked list) is a linear data structure that consists of two parts one is the data part and the other is the address part. A Doubly Linked List contains three parts: one is the data part and the other two are the address of the next and previous node in the list. All the common operations of a doubly linked list is given in the menu-driven program.

Diagrammatic representation:



Operations to be performed are:

- **traverse():** To see the contents of the linked list, it is necessary to traverse the given doubly linked list. The given traverse() function traverses and prints the content of the doubly linked list.
- **insertAtFront():** This function simply inserts an element at the front/beginning of the doubly linked list.
- **insertAtEnd():** This function inserts an element at the end of the doubly linked list.

- **insertAtPosition():** This function inserts an element at a specified position in the doubly linked list.
- **deleteFirst():** This function simply deletes an element from the front/beginning of the doubly linked list.
- **deleteEnd():** This function simply deletes an element from the end of the doubly linked list.
- **deletePosition():** This function deletes an element from a specified position in the doubly linked list.
- **SearchElement():** This function searches an element in the doubly linked list.
-

Program:

// C program for the all operations in the Doubly Linked List

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

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

// Linked List Node

```
struct node
```

```
{
```

```
    int info;
```

```
    struct node *prev, *next;
```

```
};
```

```
struct node* start = NULL;
```

// Function to traverse the linked list

```
void traverse()
```

```
{
```

```
    // List is empty
```

```
    if (start == NULL)
```

```
    {
```

```
        printf("\nList is empty\n");
```

```
        return;
```

```
    }
```

```
    // Else print the Data
```

```
    struct node* temp;
```

```
    temp = start;
```

```
    while (temp != NULL)
```

```
    {
```

```
        printf("Data = %d\n", temp->info);
```

```
        temp = temp->next;
    }
}
```

// Function to insert at the front of the linked list

void insertAtFront()

```
{
    int data;
    struct node* temp;
    temp = (struct node*)malloc(sizeof(struct node));
    printf("\nEnter number to be inserted: ");
    scanf("%d", &data);
    temp->info = data;
    temp->prev = NULL;

    // Pointer of temp will be assigned to start
    temp->next = start;
    start = temp;
}
```

// Function to insert at the end of the linked list

void insertAtEnd()

```
{
    int data;
    struct node *temp, *trav;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    printf("\nEnter number to be inserted: ");
    scanf("%d", &data);
    temp->info = data;
    temp->next = NULL;
    trav = start;

    // If start is NULL
    if (start == NULL)
    {
        start = temp;
    }
}
```

```

// Changes Links
else
{
while (trav->next != NULL)
trav = trav->next;
temp->prev = trav;
trav->next = temp;
}
}

// Function to insert at any specified position in the linked list
void insertAtPosition()
{
int data, pos, i = 1;
struct node *temp, *newnode;
newnode = malloc(sizeof(struct node));
newnode->next = NULL;
newnode->prev = NULL;

// Enter the position and data
printf("\nEnter position : ");
scanf("%d", &pos);

// If start==NULL,
if (start == NULL)
{
start = newnode;
newnode->prev = NULL;
newnode->next = NULL;
}

// If position==1,
else if (pos == 1)
{
/* newnode->next = start;
newnode->next->prev = newnode;
newnode->prev = NULL;
start = newnode; */
// this is insertion of element at position =1

```

```

        insertAtFront();
    }

// Change links
else
{
    printf("\nEnter number to be inserted: ");
    scanf("%d", &data);
    newnode->info = data;
    temp = start;
    while (i < pos - 1)
    {
        temp = temp->next;
        i++;
    }
    newnode->next = temp->next;
    newnode->prev = temp;
    temp->next = newnode;
    temp->next->prev = newnode;
}
}

// Function to delete from the front of the linked list
void deleteFirst()
{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    else
    {
        temp = start;
        start = start->next;
        if (start != NULL)
            start->prev = NULL;
        free(temp);
    }
}

// Function to delete from the end of the linked list
void deleteEnd()

```

```

{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    temp = start;
    while (temp->next != NULL)
        temp = temp->next;
    if (start->next == NULL)
        start = NULL;
    else
    {
        temp->prev->next = NULL;
        free(temp);
    }
}

```

// Function to delete from any specified position from the linked list

void deletePosition()

```

{
    int pos, i = 1;
    struct node *temp, *position;
    temp = start;

    // If DLL is empty
    if (start == NULL)
        printf("\nList is empty\n");

    // Otherwise
    else
    {
        // Position to be deleted
        printf("\nEnter position : ");
        scanf("%d", &pos);

        // If the position is the first node
        if (pos == 1)
        {
            deleteFirst(); // im,proved by Jay Ghughriwala on GeeksforGeeks
            if (start != NULL)
            {

```

```

        start->prev = NULL;
    }
    free(position);
    return;
}

// Traverse till position
while (i < pos - 1)
{
    temp = temp->next;
    i++;
}
// Change Links
position = temp->next;
if (position->next != NULL)
    position->next->prev = temp;
temp->next = position->next;

// Free memory
free(position);
}
}

```

// Function to search element from the linked list

```

void searchElement()
{

    struct node* temp;
    temp = start;

    int pos = 0, info, element;
    printf("enter element to search:");
    scanf("%d",&element);
    while(temp != NULL)
    {
        if(temp -> info == element) // Element is found
        {
            printf("\n\nThe element %d is at index %d in the list", element,pos); //If found, print and
            exit

```



```

        exit(0);
    }
    else
    {
        temp = temp ->next;    //If not found, traverse the list
        pos++;
    }
}
printf("\n\nThe element %d is not found in the list",element);
}
// Driver Code
int main()
{
    int choice;
    while (1)
    {

        printf("\n\t1 To see list i.e. display list elements\n");
        printf("\t2 For insertion at starting\n");
        printf("\t3 For insertion at end\n");
        printf("\t4 For insertion at any position\n");
        printf("\t5 For deletion of first element\n");
        printf("\t6 For deletion of last element\n");
        printf("\t7 For deletion of element at any position\n");
        printf("\t8 For searching of element\n");
        printf("\t9 To exit\n");
        printf("\nEnter Choice :\n");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                traverse();
                break;
            case 2:
                insertAtFront();
                break;
            case 3:
                insertAtEnd();

```

```
        break;
    case 4:
        insertAtPosition();
        break;
    case 5:
        deleteFirst();
        break;
    case 6:
        deleteEnd();
        break;
    case 7:
        deletePosition();
        break;

    case 8:
        searchElement();
        break;
    case 9:
        exit(1);
        break;
    default:
        printf("Incorrect Choice. Try Again \n");
        continue;
    }
}
return 0;
}
```

OUTPUT:

```
1  To see list i.e. display list elements
2  For insertion at starting
3  For insertion at end
4  For insertion at any position
5  For deletion of first element
6  For deletion of last element
7  For deletion of element at any position
8  For searching of element
9  To exit
```

Enter Choice :

1

List is empty

```
1  To see list i.e. display list elements
2  For insertion at starting
3  For insertion at end
4  For insertion at any position
5  For deletion of first element
6  For deletion of last element
7  For deletion of element at any position
8  For searching of element
9  To exit
```

Enter Choice :

2

Enter number to be inserted: 10

```
1  To see list i.e. display list elements
2  For insertion at starting
3  For insertion at end
4  For insertion at any position
5  For deletion of first element
6  For deletion of last element
7  For deletion of element at any position
8  For searching of element
9  To exit
```

- 1 To see list i.e. display list elements
- 2 For insertion at starting
- 3 For insertion at end
- 4 For insertion at any position
- 5 For deletion of first element
- 6 For deletion of last element
- 7 For deletion of element at any position
- 8 For searching of element
- 9 To exit

Enter Choice :

2

Enter number to be inserted: 10

- 1 To see list i.e. display list elements
- 2 For insertion at starting
- 3 For insertion at end
- 4 For insertion at any position
- 5 For deletion of first element
- 6 For deletion of last element
- 7 For deletion of element at any position
- 8 For searching of element
- 9 To exit

Enter Choice :

2

Enter number to be inserted: 20

- 1 To see list i.e. display list elements
- 2 For insertion at starting
- 3 For insertion at end
- 4 For insertion at any position
- 5 For deletion of first element
- 6 For deletion of last element
- 7 For deletion of element at any position
- 8 For searching of element
- 9 To exit

Enter Choice :

2

Enter number to be inserted: 30

- 1 To see list i.e. display list elements
- 2 For insertion at starting
- 3 For insertion at end
- 4 For insertion at any position
- 5 For deletion of first element
- 6 For deletion of last element
- 7 For deletion of element at any position
- 8 For searching of element
- 9 To exit

Enter Choice :

3

Enter number to be inserted: 50

- 1 To see list i.e. display list elements
- 2 For insertion at starting
- 3 For insertion at end
- 4 For insertion at any position
- 5 For deletion of first element
- 6 For deletion of last element
- 7 For deletion of element at any position
- 8 For searching of element
- 9 To exit

Enter Choice :

5

- 1 To see list i.e. display list elements
- 2 For insertion at starting
- 3 For insertion at end
- 4 For insertion at any position
- 5 For deletion of first element
- 6 For deletion of last element
- 7 For deletion of element at any position
- 8 For searching of element

Enter Choice :

5

- 1 To see list i.e. display list elements
- 2 For insertion at starting
- 3 For insertion at end
- 4 For insertion at any position
- 5 For deletion of first element
- 6 For deletion of last element
- 7 For deletion of element at any position
- 8 For searching of element
- 9 To exit

Enter Choice :

1

Data = 20

Data = 10

Data = 50

- 1 To see list i.e. display list elements
- 2 For insertion at starting
- 3 For insertion at end
- 4 For insertion at any position
- 5 For deletion of first element
- 6 For deletion of last element
- 7 For deletion of element at any position
- 8 For searching of element
- 9 To exit

Enter Choice :

6

- 1 To see list i.e. display list elements
- 2 For insertion at starting
- 3 For insertion at end
- 4 For insertion at any position
- 5 For deletion of first element
- 6 For deletion of last element
- 7 For deletion of element at any position
- 8 For searching of element
- 9 To exit

Enter Choice :

6

- 1 To see list i.e. display list elements
- 2 For insertion at starting
- 3 For insertion at end
- 4 For insertion at any position
- 5 For deletion of first element
- 6 For deletion of last element
- 7 For deletion of element at any position
- 8 For searching of element
- 9 To exit

Enter Choice :

1

Data = 20

Data = 10

- 1 To see list i.e. display list elements
- 2 For insertion at starting
- 3 For insertion at end
- 4 For insertion at any position
- 5 For deletion of first element
- 6 For deletion of last element
- 7 For deletion of element at any position
- 8 For searching of element
- 9 To exit

Enter Choice :

8

enter element to search:20

The element 20 is at index 0 in the list

...Program finished with exit code 0

Press ENTER to exit console.

EXERCISE – 10

Construct Binary Search Tree (BST) of N nodes and perform the following operations:

- a. **Insert new node into the BST.**
- b. **Traverse the BST in Inorder, Preorder and Post Order.**

Binary Search Tree (BST) is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.

Thus, a binary search tree divides all its sub-trees into two segments; left sub-tree and right sub-tree and can be defined as

$\text{left_subtree (keys)} \leq \text{node (key)} \leq \text{right_subtree (keys)}$

The operations of a binary search tree are:

- **Insert** – insert an element in a tree.
- **Preorder Traversal** – traverse a tree in a preorder manner.
- **Inorder Traversal** – traverse a tree in an inorder manner.
- **Postorder Traversal** – traverse a tree in a postorder manner

Algorithm:

Step 1: Start.

Step 2: Create a Binary Search Tree for N elements.

Step 3: Traverse the tree in inorder.

Step 4: Traverse the tree in preorder

Step 5: Traverse the tree in postorder.

Step 6: Stop

Program:

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



```

#include <stdlib.h>
struct BST
{
int data;
struct BST *left;
struct BST *right;
};
typedef struct BST NODE;
NODE *node;
NODE* createtree(NODE *node, int data)
{
    if (node == NULL)
    {
        NODE *temp;
        temp= (NODE*)malloc(sizeof(NODE));
        temp->data = data;
        temp->left = temp->right = NULL;
        return temp;
    }
    if (data < (node->data))
    {
        node->left = createtree(node->left, data);
    }
    else if (data > node->data)
    {
        node -> right = createtree(node->right, data);
    }
    return node;
}

void inorder(NODE *node)
{
    if(node != NULL)
    {
        inorder(node->left);
        printf("%d\t", node->data);
        inorder(node->right);
    }
}

```

```

void preorder(NODE *node)
{
    if(node != NULL)
    {
        printf("%d\t", node->data);
        preorder(node->left);
        preorder(node->right);
    }
}

void postorder(NODE *node)
{
    if(node != NULL)
    {
        postorder(node->left);
        postorder(node->right);
        printf("%d\t", node->data);
    }
}

void main()
{
    int data, ch, i, n;
    NODE *root=NULL;
    while (1)
    {
        printf("\n1.Insertion in Binary Search Tree\n");
        printf("2.Inorder\n");
        printf("3.Preorder\n");
        printf("4.Postorder\n");
        printf("5.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1: printf("\nEnter N value: ");
                    scanf("%d", &n);
                    printf("\nEnter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)\n");
                    for(i=0; i<n; i++)
                    {
                        scanf("%d", &data);
                        root=createtree(root, data);
                    }
                }
            }
    }

```

```

    }
    break;
    case 2: printf("\nInorder Traversal: \n");
    inorder(root);
    break;
    case 3: printf("\nPreorder Traversal: \n");
    preorder(root);
    break;
    case 4: printf("\nPostorder Traversal: \n");
    postorder(root);
    break;
    case 5: exit(0);
    default: printf("\nInvalid option");
    break;
}
}
}

```

Output:

```

1.Insertion in Binary Search Tree
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your choice: 1
Enter N value: 8
Enter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)
12
5
6
1
9
7
3
4
1.Insertion in Binary Search Tree
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your choice: 2
Inorder Traversal:
1      3      4      5      6      7      9      12
1.Insertion in Binary Search Tree
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your choice: 3
Preorder Traversal:
12      5      1      3      4      6      9      7
1.Insertion in Binary Search Tree
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your choice: 4
Postorder Traversal:
4      3      1      7      9      6      5      12
1.Insertion in Binary Search Tree
2.Inorder
3.Preorder
4.Postorder
5.Exit

```