**20MCA135 DATA STRUCTURES LAB**

*Lab Report Submitted B y*

**VINEETH P**

**Reg. No.: AJC21MCA2110**

*In Partial fulfillment for the Award of the Degree Of*

**MASTER OF COMPUTER APPLICATIONS (2 Year)**

**(MCA)**

**APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY**



**AMAL JYOTHI COLLEGE OF ENGINEERING**

**KANJIRAPPALLY**

[Affiliated to APJ Abdul Kalam Technological University, Kerala. Approved by AICTE, Accredited by NAAC with ‘A’ grade. Koovapally, Kanjirappally, Kottayam, Kerala – 686518]

**2021-2022**

**DEPARTMENT OF COMPUTER APPLICATIONS**

**AMAL JYOTHI COLLEGE OF ENGINEERING**

**KANJIRAPPALLY**



**CERTIFICATE**

This is to certify that the lab report, “**20MCA135 DATA STRUCTURES LAB”** is the bonafide work of **VINEETH P (Reg. No:AJC21MCA-2110)** in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2021-22.

Mrs. Sruthymol Kurian

Staff In-Charge

### 

**CONTENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No** | **Content** | **Date** | **Page No:** |
| 1 | Familiarization with gdb  Advanced use of gcc: Important Options -o,-c,-D,-1,-I,-g,,-O,-save,-temps,-pg  Important Commands-  Break,run,next,print,display,help Using gproof: Compile, Execute and Profile | **10-11-2021** | **4** |
| 2 | Merge two sorted arrays and store it in a third array | **15-11-2021** | **19** |
| 3 | Singly linked stack | **15-11-2021** | **22** |
| 4 | Circular Queue | **24-11-2021** | **26** |
| 5 | Doubly Linked List | **01-12-2021** | **30** |
| 6 | Binary Search Tree | **08-12-2021** | **39** |
| 7 | Set Data Structure and Set operations using Bit Strings | **15-12-2021** | **45** |
| 8 | Graph Traversal Techniques(DFS and BFS) and Topological Sorting | **06-01-2022** | **47** |
| 9 | Prim’s algorithm for finding the minimum cost spanning tree | **12-01-2022** | **52** |
| 10 | Kruskal’s Algorithm | **20-01-2022** | **54** |

**Program No: 1**

**Advanced use of GCC**

The GNU Compiler Collection (GCC) is a collection of compilers and libraries for C, C++, Objective-C, Fortran, Ada, [Go](https://linuxize.com/post/how-to-install-go-on-ubuntu-18-04/) , and D programming languages. Many open-source projects, including the GNU tools and the Linux kernel, are compiled with GCC.

## Installing GCC on Ubuntu

The default Ubuntu repositories contain a meta-package named build-essential that contains the GCC compiler and a lot of libraries and other utilities required for compiling software.

Perform the steps below to install the GCC Compiler Ubuntu 18.04:

1. Start by updating the packages list:

*sudo apt-get update*

1. Install the build-essential package by typing:

*sudo apt-get install build-essential*

The command installs a bunch of new packages including gcc, g++ and make.

You may also want to install the manual pages about using GNU/Linux for development:

*sudo apt-get install manpages-dev*

1. To validate that the GCC compiler is successfully installed, use the gcc -version command which prints the GCC version:

*gcc -version Or gcc -v*

The default version of GCC available in the Ubuntu 18.04 repositories is 7.4.0:

**Compile and run a c++ program**

Now go to that folder where you will create C/C++ programs. I am creating my programs in Desktop directory. Type these commands:

$ cd Desktop

$ sudo mkdir tst

$ cd tst

Open a file using any editor . Add this code in the file:

#include <iostream>

using namespace std;

int main() {

cout << "Hello World!";

return 0;

}

Save the file and exit.

Compile the program using any of the following command:

1. *$ sudo g++ p1.cpp (p1 is the filename)*

*(or)*

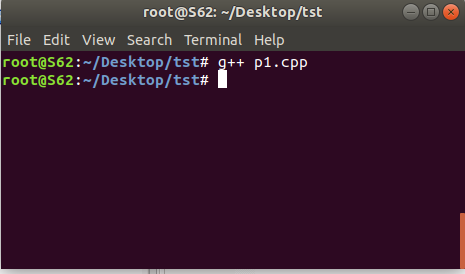
1. *$ sudo g++ -o p1 p1.cpp*

*1. $ sudo g++ p1.cpp*

To compile your c++ code, use:

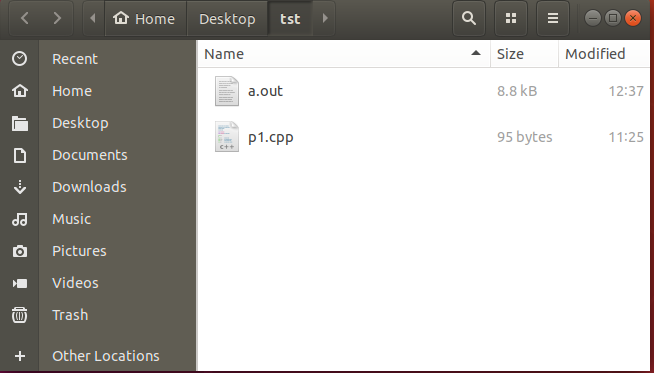
*g++ p1.cpp*

p1.cpp in the example is the name of the program to be compiled.



This will produce an executable in the same directory called a.out which you can run by typing this in your terminal:

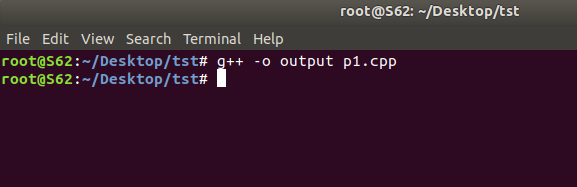
*./a.out*



*$ sudo g++ -o output p1.cpp*

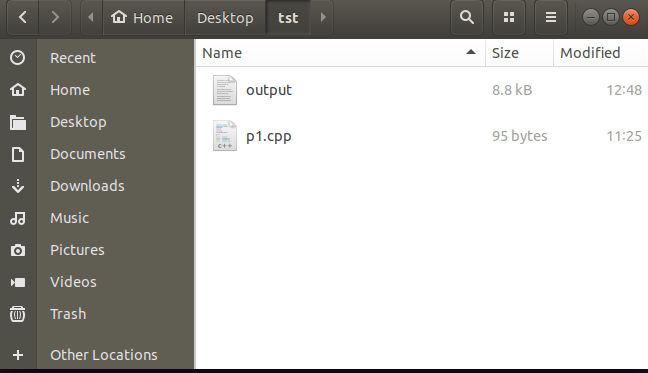
To specify the name of the compiled output file, so that it is not named **a.out**, use **-o** with your g++ command.

g++ -o output p1.cpp



This will compile p1.cpp to the binary file named output, and you can type ./output to run the compiled code.

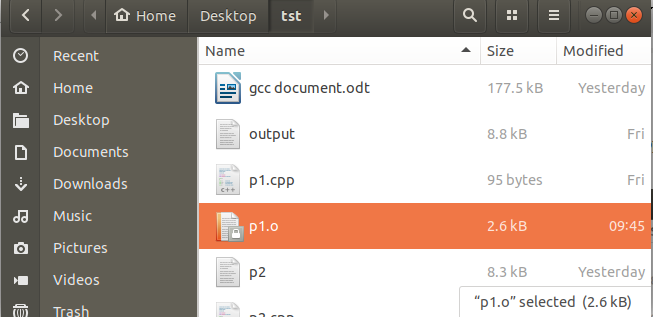
*./output.out*

**

**GCC: Important Options**

* *-c*

To produce only the compiled code (without any linking), use the -C option.

gcc -C p2.cpp

The command above would produce a file main.o that would contain machine level code or the compiled code.

* *-D*

The compiler option D can be used to define compile time macros in code.

Here is an example :

#include<stdio.h>

int main(void)

{

#ifdef MY\_MACRO

printf("\n Macro defined \n");

#endif

char c = -10;

// Print the string

printf("\n The Geek Stuff [%d]\n", c);

return 0;

}

The compiler option -D can be used to define the macro MY\_MACRO from command line.

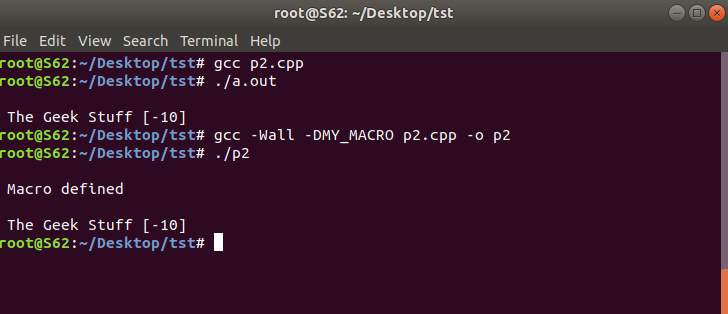
$ gcc -Wall -DMY\_MACRO main.c -o main

$ ./main

Macro defined

The Geek Stuff [-10]

The print related to macro in the output confirms that the macro was defined.



* *-l*

The option -l can be used to link with shared libraries. For example:

gcc -Wall main.c -o main -lCPPfile

The gcc command mentioned above links the code main.c with the shared library libCPPfile.so to produce the final executable ‘main’.

* *-g*

A program which goes into an infinite loop or "hangs" can be difficult to debug. On most systems a foreground process can be stopped by hitting Control-C, which sends it an interrupt signal (SIGINT). However, this does not help in debugging the problem--the SIGINT signal terminates the process without producing a core dump. A more sophisticated approach is to *attach* to the running process with a debugger and inspect it interactively.

For example, here is a simple program with an infinite loop:

int

main (void)

{

usigned int i = 0;

while (1) { i++; };

return 0;

}

In order to attach to the program and debug it, the code should be compiled with the debugging option -g:

$ gcc -Wall -g loop.c

$ ./a.out

(program hangs)

Once the executable is running we need to find its process id (PID). This can be done from another session with the command ps x:

$ ps x

PID TTY STAT TIME COMMAND

... ..... . ....

891 pts/1 R 0:11 ./a.out

* *-save-temps*

Through this option, output at all the stages of compilation is stored in the current directory. Please note that this option produces the executable also.

For example :

$ gcc -save-temps p2.cpp

$ ls

a.out p2.c p2.i p2.o p2.s

So we see that all the intermediate files as well as the final executable was produced in the output.

* *-pg*

*Generate extra code to write profile information suitable for the analysis program gprof. You must use this option when compiling the source files you want data about, and you must also use it when linking.*

# *GDB Tutorial*

Gdb is a debugger for C (and C++). It allows you to do things like run the program up to a certain point then stop and print out the values of certain variables at that point, or step through the program one line at a time and print out the values of each variable after executing each line. It uses a command line interface.

This is a brief description of some of the most commonly used features of gdb.

## Compiling

To prepare your program for debugging with gdb, you must compile it with the -g flag. So, if your program is in a source file called memsim.c and you want to put the executable in the file memsim, then you would compile with the following command:

gcc -g -o memsim memsim.c

## Invoking and Quitting GDB

To start gdb, just type gdb at the unix prompt. Gdb will give you a prompt that looks like this: (gdb). From that prompt you can run your program, look at variables, etc., using the commands listed below (and others not listed). Or, you can start gdb and give it the name of the program executable you want to debug by saying

gdb *executable*

To exit the program just type quit at the (gdb) prompt (actually just typing q is good enough).

## Commands

#### help

Gdb provides online documentation. Just typing help will give you a list of topics. Then you can type help *topic* to get information about that topic (or it will give you more specific terms that you can ask for help about). Or you can just type help *command* and get information about any other command.

#### file

file *executable* specifies which program you want to debug.

#### run

run will start the program running under gdb. (The program that starts will be the one that you have previously selected with the file command, or on the unix command line when you started gdb. You can give command line arguments to your program on the gdb command line the same way you would on the unix command line, except that you are saying run instead of the program name:

run 2048 24 4

You can even do input/output redirection: run > outfile.txt.

#### break

A ``breakpoint'' is a spot in your program where you would like to temporarily stop execution in order to check the values of variables, or to try to find out where the program is crashing, etc. To set a breakpoint you use the break command.

break *function* sets the breakpoint at the beginning of *function*. If your code is in multiple files, you might need to specify *filename:function*.

break *linenumber* or break *filename:linenumber* sets the breakpoint to the given line number in the source file. Execution will stop before that line has been executed.

#### delete

delete will delete all breakpoints that you have set.

delete *number* will delete breakpoint numbered *number*. You can find out what number each breakpoint is by doing info breakpoints. (The command info can also be used to find out a lot of other stuff. Do help info for more information.)

#### clear

clear *function* will delete the breakpoint set at that function. Similarly for *linenumber*, *filename:function*, and *filename:linenumber*.

#### continue

continue will set the program running again, after you have stopped it at a breakpoint.

#### step

step will go ahead and execute the current source line, and then stop execution again before the next source line.

#### next

next will continue until the next source line in the current function (actually, the current innermost stack frame, to be precise). This is similar to step, except that if the line about to be executed is a function call, then that function call will be completely executed before execution stops again, whereas with step execution will stop at the first line of the function that is called.

#### until

until is like next, except that if you are at the end of a loop, until will continue execution until the loop is exited, whereas next will just take you back up to the beginning of the loop. This is convenient if you want to see what happens after the loop, but don't want to step through every iteration.

#### list

list *linenumber* will print out some lines from the source code around *linenumber*. If you give it the argument *function* it will print out lines from the beginning of that function. Just list without any arguments will print out the lines just after the lines that you printed out with the previous list command.

#### print

print *expression* will print out the value of the expression, which could be just a variable name. To print out the first 25 (for example) values in an array called list, do

print list[0]@25

**Gprof**

Profiling is an important aspect of software programming. Through profiling one can determine the parts in program code that are time consuming and need to be re-written. This helps make your program execution faster which is always desired.  
  
In very large projects, profiling can save your day by not only determining the parts in your program which are slower in execution than expected but also can help you find many other statistics through which many potential bugs can be spotted and sorted out.

### *How to use gprof*

Using the gprof tool is not at all complex. You just need to do the following on a high-level:

* Have profiling enabled while compiling the code
* Execute the program code to produce the profiling data
* Run the gprof tool on the profiling data file (generated in the step above).

*Lets try and understand the three steps listed above through a practical example. Following test code will be used throughout the article :*

*//test\_gprof.c*

#include<stdio.h>

void new\_func1(void);

void func1(void)

{

printf("\n Inside func1 \n");

int i = 0;

for(;i<0xffffffff;i++);

new\_func1();

return;

}

static void func2(void)

{

printf("\n Inside func2 \n");

int i = 0;

for(;i<0xffffffaa;i++);

return;

}

int main(void)

{

printf("\n Inside main()\n");

int i = 0;

for(;i<0xffffff;i++);

func1();

func2();

return 0;

}

*//test\_gprof\_new.c*

#include<stdio.h>

void new\_func1(void)

{

printf("\n Inside new\_func1()\n");

int i = 0;

for(;i<0xffffffee;i++);

return;

}

### Step-1 : Profiling enabled while compilation

In this first step, we need to make sure that the profiling is enabled when the compilation of the code is done. This is made possible by adding the ‘-pg’ option in the compilation step.

lets compile our code with ‘-pg’ option :

$ gcc -Wall -pg test\_gprof.c test\_gprof\_new.c -o test\_gprof

Please note : The option ‘-pg’ can be used with the gcc command that compiles (-c option), gcc command that links(-o option on object files) and with gcc command that does the both(as in example above).

### Step-2 : Execute the code

In the second step, the binary file produced as a result of step-1 (above) is executed so that profiling information can be generated.

$ ls

test\_gprof test\_gprof.c test\_gprof\_new.c

$ ./test\_gprof

Inside main()

Inside func1

Inside new\_func1()

Inside func2

$ ls

gmon.out test\_gprof test\_gprof.c test\_gprof\_new.c

So we see that when the binary was executed, a new file ‘gmon.out’ is generated in the current working directory.

### *Step-3 : Run the gprof tool*

In this step, the gprof tool is run with the executable name and the above generated ‘gmon.out’ as argument. This produces an analysis file which contains all the desired profiling information.

$ gprof test\_gprof gmon.out > analysis.txt

Note that one can explicitly specify the output file (like in example above) or the information is produced on stdout.

$ ls

analysis.txt gmon.out test\_gprof test\_gprof.c test\_gprof\_new.c

So we see that a file named ‘analysis.txt’ was generated. As produced above, all the profiling information is now present in ‘analysis.txt’. Lets have a look at this text file :

Flat profile:

Each sample counts as 0.01 seconds.

% cumulative self self total

time seconds seconds calls s/call s/call name

33.86 15.52 15.52 1 15.52 15.52 func2

33.82 31.02 15.50 1 15.50 15.50 new\_func1

33.29 46.27 15.26 1 15.26 30.75 func1

0.07 46.30 0.03 main

% the percentage of the total running time of the

time program used by this function.

cumulative a running sum of the number of seconds accounted

seconds for by this function and those listed above it.

self the number of seconds accounted for by this

seconds function alone. This is the major sort for this

listing.

calls the number of times this function was invoked, if

this function is profiled, else blank.

self the average number of milliseconds spent in this

ms/call function per call, if this function is profiled,

else blank.

total the average number of milliseconds spent in this

ms/call function and its descendents per call, if this

function is profiled, else blank.

name the name of the function. This is the minor sort

for this listing. The index shows the location of

the function in the gprof listing. If the index is

in parenthesis it shows where it would appear in

the gprof listing if it were to be printed.

Call graph (explanation follows)

granularity: each sample hit covers 2 byte(s) for 0.02% of 46.30 seconds

index % time self children called name

[1] 100.0 0.03 46.27 main [1]

15.26 15.50 1/1 func1 [2]

15.52 0.00 1/1 func2 [3]

-----------------------------------------------

15.26 15.50 1/1 main [1]

[2] 66.4 15.26 15.50 1 func1 [2]

15.50 0.00 1/1 new\_func1 [4]

-----------------------------------------------

15.52 0.00 1/1 main [1]

[3] 33.5 15.52 0.00 1 func2 [3]

-----------------------------------------------

15.50 0.00 1/1 func1 [2]

[4] 33.5 15.50 0.00 1 new\_func1 [4]

-----------------------------------------------

This table describes the call tree of the program, and was sorted by

the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the

index number at the left hand margin lists the current function.

The lines above it list the functions that called this function,

and the lines below it list the functions this one called.

This line lists:

index A unique number given to each element of the table.

Index numbers are sorted numerically.

The index number is printed next to every function name so

it is easier to look up where the function in the table.

% time This is the percentage of the `total' time that was spent

in this function and its children. Note that due to

different viewpoints, functions excluded by options, etc,

these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this

function by its children.

called This is the number of times the function was called.

If the function called itself recursively, the number

only includes non-recursive calls, and is followed by

a `+' and the number of recursive calls.

name The name of the current function. The index number is

printed after it. If the function is a member of a

cycle, the cycle number is printed between the

function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly

from the function into this parent.

children This is the amount of time that was propagated from

the function's children into this parent.

called This is the number of times this parent called the

function `/' the total number of times the function

was called. Recursive calls to the function are not

included in the number after the `/'.

name This is the name of the parent. The parent's index

number is printed after it. If the parent is a

member of a cycle, the cycle number is printed between

the name and the index number.

If the parents of the function cannot be determined, the word

`' is printed in the `name' field, and all the other

fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly

from the child into the function.

children This is the amount of time that was propagated from the

child's children to the function.

called This is the number of times the function called

this child `/' the total number of times the child

was called. Recursive calls by the child are not

listed in the number after the `/'.

name This is the name of the child. The child's index

number is printed after it. If the child is a

member of a cycle, the cycle number is printed

between the name and the index number.

If there are any cycles (circles) in the call graph, there is an

entry for the cycle-as-a-whole. This entry shows who called the

cycle (as parents) and the members of the cycle (as children.)

The `+' recursive calls entry shows the number of function calls that

were internal to the cycle, and the calls entry for each member shows,

for that member, how many times it was called from other members of

the cycle.

Index by function name

[2] func1 [1] main

[3] func2 [4] new\_func1

So (as already discussed) we see that this file is broadly divided into two parts :

1. Flat profile  
2. Call graph

The individual columns for the (flat profile as well as call graph) are very well explained in the output itself.

### Customize gprof output using flags

There are various flags available to customize the output of the gprof tool. Some of them are discussed below:

### 1. Suppress the printing of statically(private) declared functions using -a

If there are some static functions whose profiling information you do not require then this can be achieved using -a option :

$ gprof -a test\_gprof gmon.out > analysis.txt

### 2. Suppress verbose blurbs using -b

As you would have already seen that gprof produces output with lot of verbose information so in case this information is not required then this can be achieved using the -b flag.

$ gprof -b test\_gprof gmon.out > analysis.txt

### 3. Print only flat profile using -p

In case only flat profile is required then :

$ gprof -p -b test\_gprof gmon.out > analysis.txt

Note that I have used(and will be using) -b option so as to avoid extra information in analysis output.

### 4. Print information related to specific function in flat profile

This can be achieved by providing the function name along with the -p option:

$ gprof -pfunc1 -b test\_gprof gmon.out > analysis.txt

### 5. Suppress flat profile in output using -P

If flat profile is not required then it can be suppressed using the -P option :

$ gprof -P -b test\_gprof gmon.out > analysis.txt

### 6. Print only call graph information using -q

gprof -q -b test\_gprof gmon.out > analysis.txt

### 7. Print only specific function information in call graph.

This is possible by passing the function name along with the -q option.

$ gprof -qfunc1 -b test\_gprof gmon.out > analysis.txt

### 8. Suppress call graph using -Q

If the call graph information is not required in the analysis output then -Q option can be used.

$ gprof -Q -b test\_gprof gmon.out > analysis.txt

**Program no: 2**

**Aim: To merge two arrays into a third array , and sort third array using any of the sorting technique**

**Algorithm:**

1. START
2. Read two arrays of size M and N respectively
3. Print the values of each of the arrays
4. Merge array 1 and array 2 into a third array array3
5. Using bubble sort or selection sort or exchange sort , sort the array3

5.1. For i=0 to i=n-1

5.1.1. For j=0 to j<n-i

5.1.2..if (array3[j]>array3[j+1])

5.1.2.1. TEMP = array3[i+1];

5.1.2.2. array3[j+1]=array3[j];

5.1.2.3. array3[j]= TEMP;

1. Display the final sorted merged array
2. STOP

**Source Code:**

#include<stdio.h>

#include<stdlib.h>

void main()

{

int a[20],b[20],c[40];

int n,i,j,temp,m;

printf("\n Enter the size of array 1: ");

scanf("%d",&n);

printf("\n Enter %d elements into the array 1:\n",n);

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

scanf("%d",&a[i]);

printf("\n Enter the size of array 2:\n");

scanf("%d",&m);

printf("\n Enter %d elements into the array 2: \n",m);

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

scanf("%d",&b[i]);

printf("\n The array 1 contains :\n");

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

printf("%d\t",a[i]);

printf("\n The array 2 contains:\n ");

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

printf("%d\t",b[i]);

printf("\n");

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

c[i]=a[i];

printf("%d\t",i);

for(i=n;i<(m+n);i++)

c[i]=b[i-n];

printf("\n THE merged array in unsorted form\n");

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

printf("%d\t",c[i]);

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

{

for(j=0;j<(m+n)-i;j++)

{

if(c[j]>c[j+1])

{

temp=c[j];

c[j]=c[j+1];

c[j+1]=temp;

}

}

}

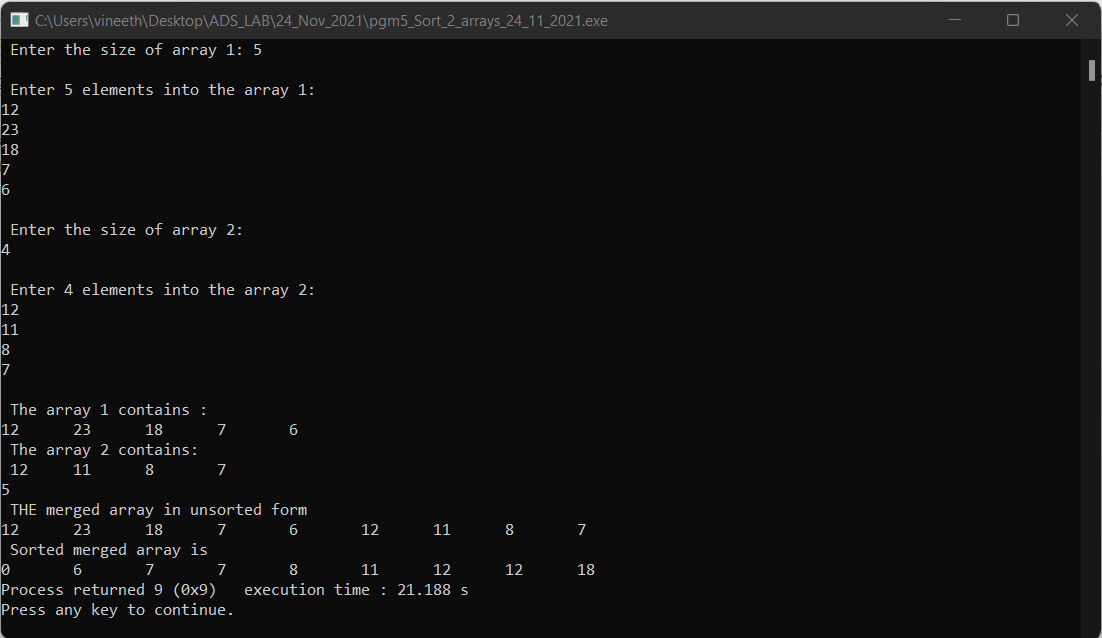
printf("\n Sorted merged array is\n");

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

printf("%d\t",c[i]);

}

**Output:**

****

**Program no:3**

**Aim: To implement a stack using linked list**

**Algorithm:**

1. START
2. Declare a structure containing a data part as well as an address part
3. Present a menu of operations push,pop,display to the users by switch-case
4. If the operation is push()

4.1. Check whether array is empty or not

4.1.1. If empty create a node in newnode and assign TOP=newnode

4.1.1. Store a value in newnode->data

4.1.2. Else, create a temporary node and assign struct node \*temp=TOP

4.1.2.1. While tem->next!=NULL, traverse the linked stack

4.1.2.1.1. Attach a newnode with a value at the end position

4.2. Go back to step 3

1. If operation is pop()

5.1. Check whether list is empty or top is NULL

5.1.1 If yes, print ‘ stack underflow’, else free the last element by iterating list.

5.2. Display the linked stack elements

5.3. Go back to step 3

1. If operation is display, traverse the list items one by one and print the values
2. If operation is for exit, quit the menu and return
3. STOP

**Source Code:**

#include<stdio.h>

#include<stdlib.h>

void push();

void pop();

void display();

struct stacknode

{

int data;

struct stacknode \*next;

}\*top=NULL;

void main()

{

int opt;

do

{

printf("\n SELECT AN OPTION FROM THE BELOW MENU\n");

printf("\n 1. PUISH OPERATION\n");

printf("\n 2. POP OPERATION\n");

printf("\n 3. VIEW LINKED STACK\n");

printf("\n 4. EXIT\n");

scanf("%d",&opt);

switch(opt)

{

case 1: push();

break;

case 2: printf("\n Poping Out the last item :\n");

pop();

break;

case 3: printf("\n The stack consists of items \n");

display();

break;

case 4: exit(0);

default: printf("\n Invalid Option\n");

}

}

while(opt!=4);

}

void push()

{

struct stacknode \*newnode;

newnode=(struct stacknode\*)malloc(sizeof(struct stacknode));

printf("\n enter a value :\n");

scanf("%d",&newnode->data);

if(top==NULL)

{

newnode->next=NULL;

top=newnode;

}

else

{

newnode->next=top;

top=newnode;

}

printf("\n Now, the items in the stack are\n");

display();

}

void pop()

{

if(top==NULL)

printf("\n Stack Underflow, Insert atleast one item\n");

else

{

struct stacknode \*temp=top;

top=temp->next;

printf("\n the item %d has been poped out...\n",temp->data);

free(temp);

printf("\n Now , the items in the stack are\n");

display();

}

}

void display()

{

struct stacknode \*temp=top;

if(top==NULL)

printf("\n STACK IS EMPTY!\n");

else

{

while(temp!=NULL)

{

printf("%d\t",temp->data);

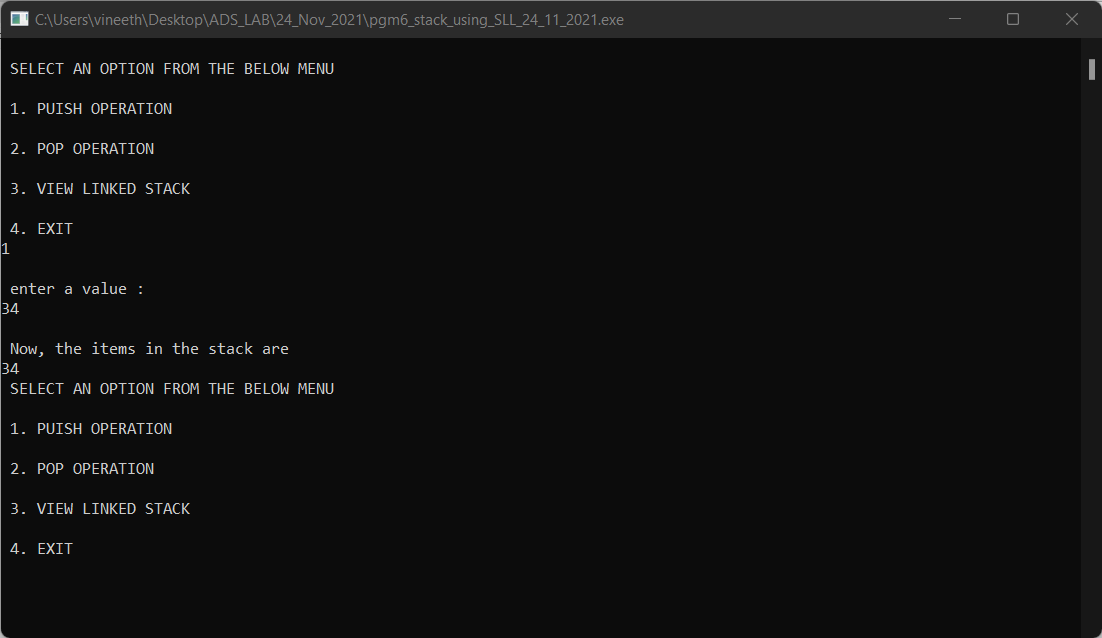
temp=temp->next;

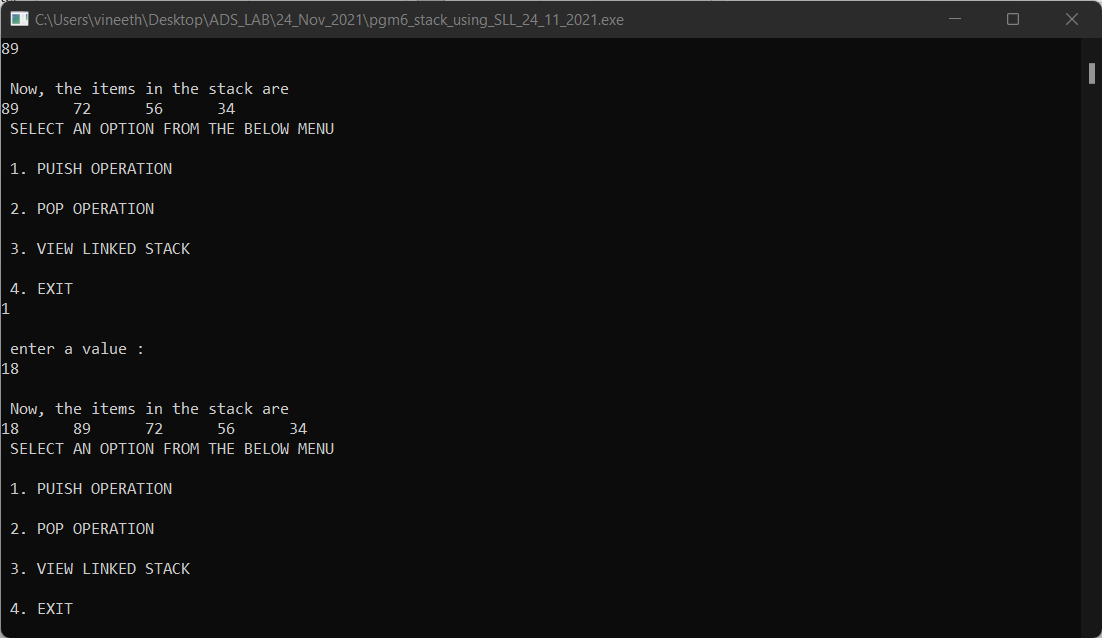
}

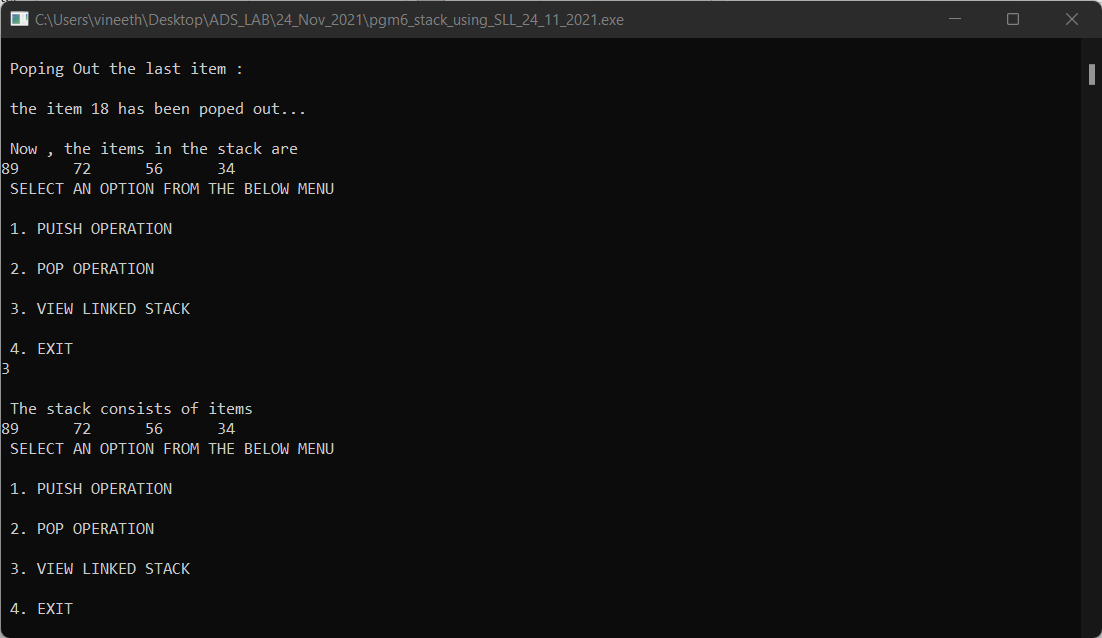
}

}

**Output:**

****





**Program no:4**

**Aim: To implement a circular queue using linked list**

**Algorithm:**

1. Define a structure to implement a node

2. Declare front and rear as NULL

3. If operation is enqueue(),

a. If rear==null, insert newnode to front , set front and rear as address of newnode

b. Else travel the list using temporary pointer and attach newnode at end

4. If operation is dequeue(),

a. If front==NULL, print ‘ list is empty’

b. Otherwise set front as address of second node, front = front->next

5. If display(),

a. While ( temp->rear!=NULL) , print(DATA(temp)

**Source Code:**

#include<stdio.h>

#include<stdlib.h>

void enqueue();

void dequeue();

void display();

struct node

{

int data;

struct node \*next;

}\*front=NULL;

struct node \*rear=NULL;

struct node \*newnode;

void create\_node()

{

newnode=(struct node\*)malloc(sizeof(struct node));

printf("\n enter a value for the node: ");

scanf("%d",&newnode->data);

}

void main()

{

int opt;

do

{

printf("\n SELECT A CHOICE\n");

printf("\n1. ENQUEUE\n");

printf("\n2. DEQUEUE\n");

printf("\n3. DISPLAY\n");

printf("\n4. EXIT\n");

scanf("%d",&opt);

switch(opt)

{

case 1: enqueue();

break;

case 2: dequeue();

break;

case 3: display();

break;

case 4: exit(0);

default: printf("\n Invalid Choice\n");

}

}

while(opt!=4);

}

void enqueue()

{

create\_node();

if(front==NULL && rear==NULL)

{

front=newnode;

rear=newnode;

newnode->next=front;

}

else

{

struct node \*temp=front;

while(temp->next!=front)

temp=temp->next;

newnode->next=temp->next;

temp->next=newnode;

rear=newnode;

}

display();

}

void dequeue()

{

if(front==NULL&&rear==NULL)

printf("\n QUEUE IS empty\n");

else

{

if(front->next==front)

{

printf("\n the node %d has been deleted",front->data);

front=NULL;

rear=NULL;

}

else

{

struct node \*temp=front;

struct node \*temp1=front;

while(temp->next!=front)

temp=temp->next;

temp->next=front->next;

front=front->next;

printf("\n the node %d has been deleted\n",temp1->data);

free(temp1);

}

display();

}

}

void display()

{

if(front==NULL)

printf("\n the CIRCULAR QUEUE IS empty...!!!\n");

else

{

struct node \*temp=front;

while(temp->next!=front)

{

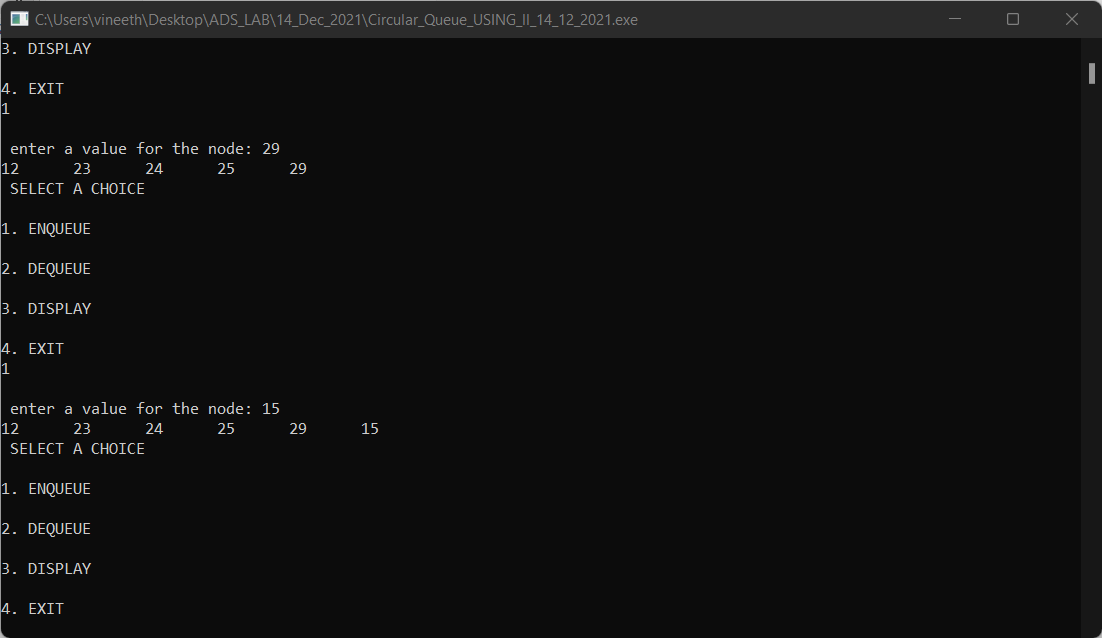
printf("%d\t",temp->data);

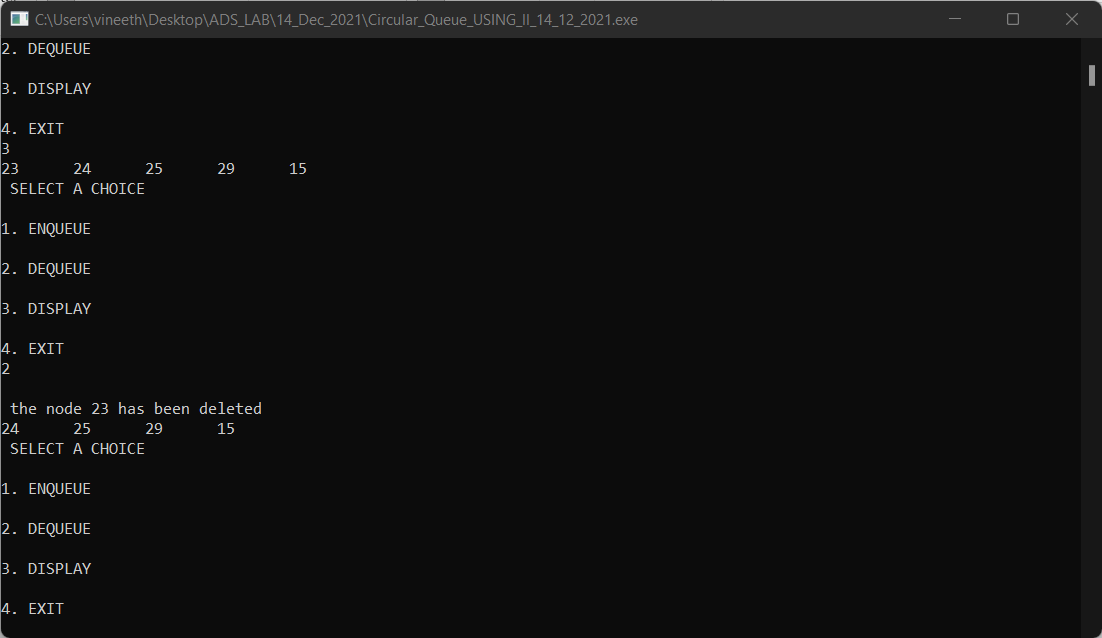
temp=temp->next;

}

printf("%d\t",temp->data);

}

**Output:**

****

**Program no: 5**

**Aim: To implement a doubly linked list and it’s operations**

**Algorithm:**

1.START

2.Display a menu of operations

3. If choice is for insertion

3.1. If beginning

3.1.1.set previous node of newnode to NULL

3.1.2.set next node of newnode to head

3.1.3.set head to point to newnode

3.2. If end

3.2.1.set temp to point to the head (temp=head)

3.2.2.Travel the doubly linked list till the next of temp is null

3.2.3.insert newnode into temp->next

3.2.4.set newnode->next as NULL

3.2.5.set newnode->previous=temp

3.3. If particular position

3.3.1.set temp to point to the first node (head)

3.3.2.travel till the desired position reach

3.3.3.insert the newnode at next to the temp

3.3.4.set previous pointer of newnode to temp

3.3.5.set next pointer of newnode to the current

next of temp

3.3.6.set next pointer of temp to newnode

4. If choice is for deletion

4.1. If beginning

4.1.1.make temp point to first node

4.1.2 .set previous link of next link of temp to be

NULL

4.1.3. Set head pointing to the node next to temp;

4.1.4..set NULL at next pointer of temp

4.2. If end

4.2.1.assign temp as newnode

4.2.2.traverse the list till next of temp equal to

4.2.3.set next pointer of previous link of temp to

NULL

4.2.4. set previous pointer of temp to NULL, and free temp

4.3. If specific position

4.3.1. Set temp pointing to the first node (head)

4.3.2. Read a logical position from which the node

is to be removed

4.3.3. Remove the desired node by traversing the

list

4.3.4. Set the next pointer of node residing before

temp as the node after temp

4.3.5. Set previous link of node after temp pointing to node before temp

5. If choice is for searching an item in the list, traverse the list

by checking that whether the item is matching with the data

part of current node

5.1. Assign temp = head

5.2. while(temp->data!=item),traverse by temp=temp->next

5.3. If data in temp is equal to item, print ‘ item found in list’

5.4. Otherwise print “ item does not exist”

6. If operation is for traversal or display

6.1. Set temp as head

6.2. Print the data contained in temp while temp

reaches last node

7. If user’s choice is none other than above , print ‘invalid

choice’

8. Continue steps 2 to 7 till the user input an option for exit

9. STOP

**Source Code:**

#include<stdio.h>

#include<stdlib.h>

int count=0;

void insert\_begin();

void insert\_end();

void insert\_pos();

void delete\_begin();

void delete\_end();

void delete\_pos();

void search\_key();

void traverse\_list();

struct node

{

int data;

struct node \*prev;

struct node \*next;

}\*head=NULL;

void main()

{

int opt,item;

do

{

printf("\n SELECT A VALID OPTION FROM THE MENU\n");

printf("\n1. INSERTION AT BEGINNING\n");

printf("\n2. INSERTION AT END\n");

printf("\n3. INSERTION AT A GIVEN POSITION\n");

printf("\n4. DELETION AT BEGINNING\n");

printf("\n5. DELETION AT END\n");

printf("\n6. DELETION AT A PARTICULAR POSITION\n");

printf("\n7. SEARCH FOR AN ITEM\n");

printf("\n8. DISPLAY LIST\n");

printf("\n9. EXIT\n");

scanf("%d",&opt);

switch(opt)

{

case 1: insert\_begin();

break;

case 2: insert\_end();

break;

case 3: insert\_pos();

break;

case 4: delete\_begin();

break;

case 5: delete\_end();

break;

case 6: delete\_pos();

break;

case 7: search\_key();

break;

case 8: traverse\_list();

break;

case 9: exit(0);

default: printf("\n Invalid Option\n");

}

}

while(opt!=9);

}

void insert\_begin()

{

int item;

printf("\n enter a value: ");

scanf("%d",&item);

struct node \*newnode;

newnode=(struct node\*)malloc(sizeof(struct node));

newnode->data=item;

if(head==NULL)

{

head=newnode;

newnode->prev=NULL;

newnode->next=NULL;

count++;

}

else

{

struct node \*temp=head;

temp->prev=newnode;

newnode->prev=NULL;

newnode->next=temp;

head=newnode;

count++;

}

printf("\n the items in the list are:\n");

traverse\_list();

}

void insert\_end()

{

int item;

printf("\n enter a value: ");

scanf("%d",&item);

struct node \*newnode;

newnode=(struct node\*)malloc(sizeof(struct node));

newnode->data=item;

if(head==NULL)

{

head=newnode;

newnode->prev=NULL;

newnode->next=NULL;

count++;

}

else

{

struct node \*temp=head;

while(temp->next!=NULL)

temp=temp->next;

temp->next=newnode;

newnode->prev=temp;

newnode->next=NULL;

count++;

}

printf("\n the items in the list are\n");

traverse\_list();

}

void insert\_pos()

{

int item,pos,i=1;

struct node \*temp=head;

printf("\n enter a value: ");

scanf("%d",&item);

struct node \*newnode;

newnode=(struct node\*)malloc(sizeof(struct node));

newnode->data=item;

printf("\n Enter the position to which the new node is to be inserted: ");

scanf("%d",&pos);

if(pos>count)

{

printf("\n invalid position\n");

}

while(temp->next!=NULL&&i!=pos-1)

{

temp=temp->next;

i++;

}

if(i==pos-1)

{

newnode->next=temp->next;

temp->next=newnode;

newnode->prev=temp;

count++;

}

else

{

if(pos==count)

{

while(temp->next!=NULL)

temp=temp->next;

temp->next=newnode;

newnode->next=NULL;

newnode->prev=temp;

count++;

}

else

printf("\n POSITION not found in list\n");

}

printf("\n the items in the list are\n");

traverse\_list();

}

void delete\_begin()

{

struct node \*temp=head;

if(head==NULL)

printf("\n doubly linked list is empty\n");

else

{

if(temp->next==NULL)

{

temp->prev=NULL;

head=NULL;

printf("\n the item %d has been deleted\n",temp->data);

free(temp);

count--;

traverse\_list();

}

else

{

head=temp->next;

temp->next->prev=NULL;

temp->prev=NULL;

temp->next=NULL;

printf("\n the item %d has been deleted from beginning\n",temp->data);

free(temp);

count--;

printf("\n the items in the list are\n");

traverse\_list();

}

}

}

void delete\_end()

{

struct node \*temp=head;

if(head==NULL)

printf("\n doubly linked list is empty\n");

else if(temp->next==NULL)

{

printf("\n the item %d has been deleted\n",temp->data);

temp->prev=NULL;

temp->next=NULL;

head=NULL;

free(temp);

count--;

printf("\n the items in the list are\n");

traverse\_list();

}

else

{

while(temp->next!=NULL)

temp=temp->next;

temp->prev->next=NULL;

temp->prev=NULL;

printf("\n the item %d has been deleted from end\n",temp->data);

free(temp);

count--;

printf("\n the items in the list are\n");

traverse\_list();

}

}

void delete\_pos()

{

int pos,i=1;

struct node \*temp=head;

if(head==NULL)

printf("\n the doubly linked list is empty\n");

else

{

printf("\n enter the position of node to be deleted: ");

scanf("%d",&pos);

if(pos>count)

printf("\n position is not within the list\n");

else

{

while(temp->next!=NULL&&pos!=i)

{

temp=temp->next;

i++;

}

temp->prev->next=temp->next;

temp->prev=NULL;

temp->next=NULL;

printf("\n the item %d has been deleted",temp->data);

free(temp);

count--;

printf("\n the items in the doubly linked list are\n");

traverse\_list();

}

}

}

void traverse\_list()

{

struct node \*temp=head;

if(head==NULL)

printf("\n list is empty\n");

else

{

while(temp!=NULL)

{

printf("%d\t",temp->data);

temp=temp->next;

}

}

}

void search\_key()

{

int item;

printf("\n enter an item to be searched: \n");

scanf("%d",&item);

struct node \*temp=head;

while(temp->data!=item&&temp->next!=NULL)

temp=temp->next;

if(temp->data==item)

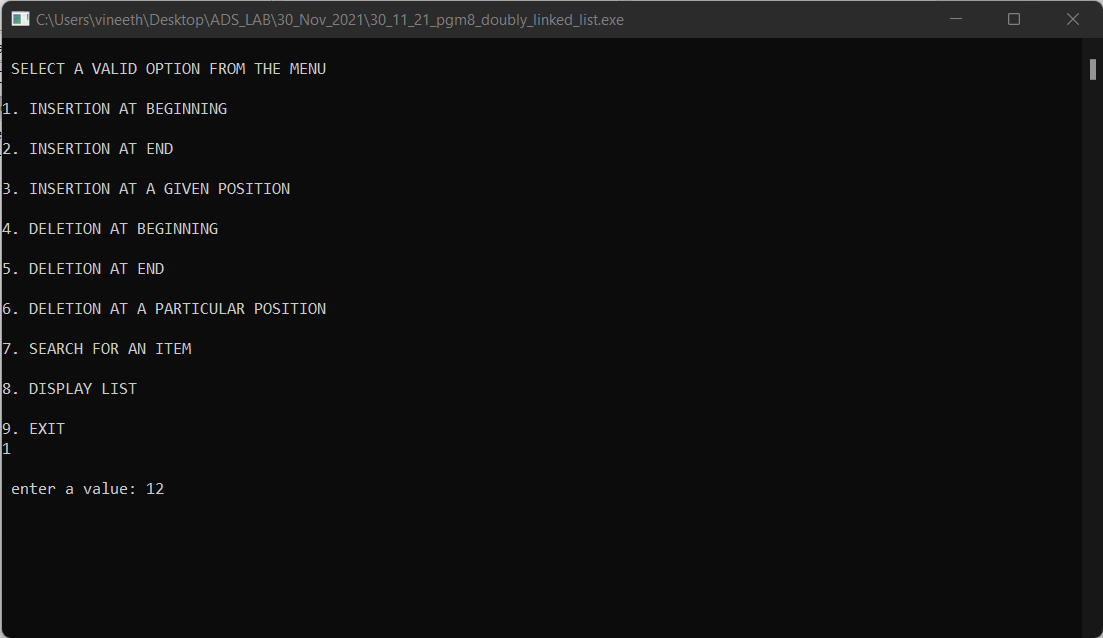
printf("\n the item %d found in the list",item);

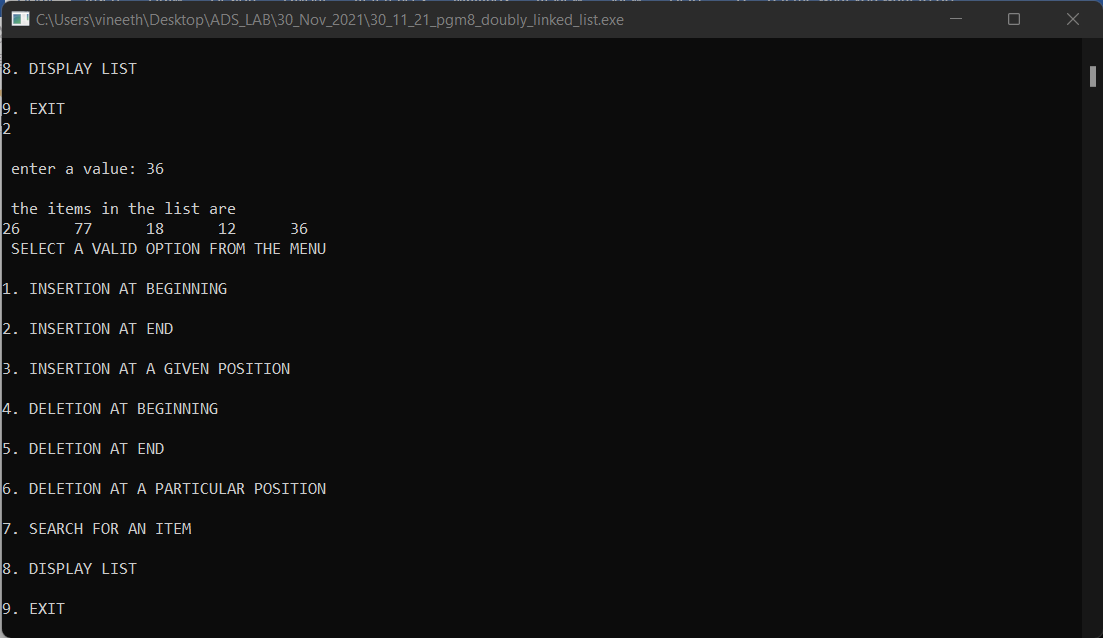
else

printf("\n the item %d not found in the list\n",item);

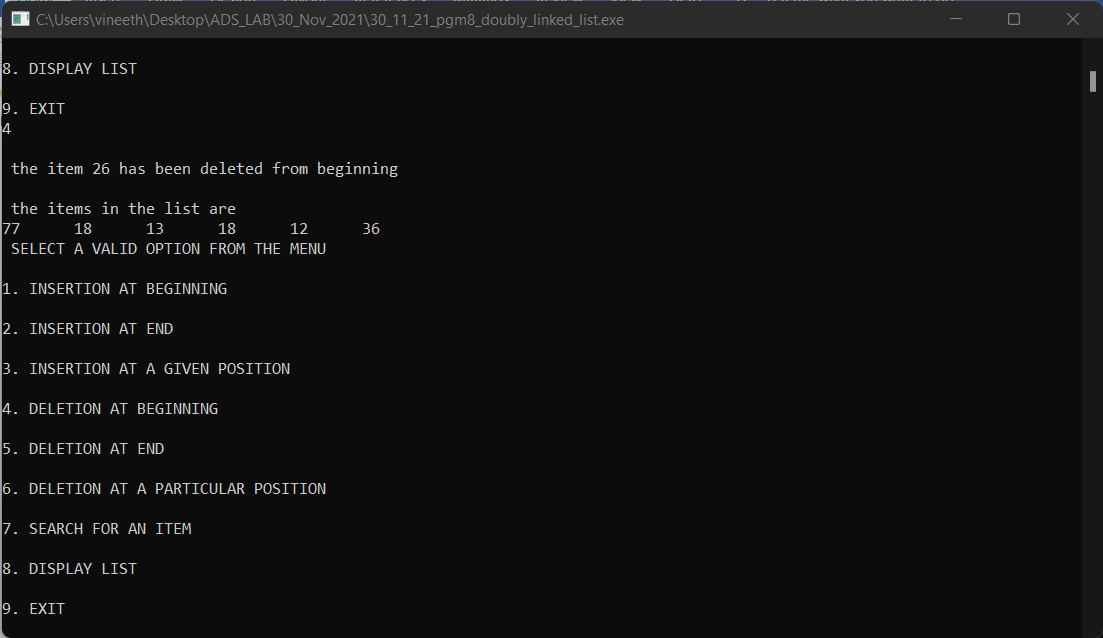
}

**Output:**

****







**Program no:6**

**Aim: To implement binary search tree**

**Algorithm:**

1. Start
2. Define a structure for BST
3. Display a menu of operations
4. If inorder traversal
   1. Visit the left child
   2. Process the node currently accessed
   3. Visit the right child
5. If preorder traversal
   1. Process the node currenly accessed
   2. Visit the left child
   3. Visit the right child
6. If postorder traversal
   1. Visit the left child
   2. Visit the right child
   3. Process the currently visited node
7. If insertion operaton
   1. Read a value in key
   2. If root is NULL, insert new node as root
   3. Else check if key less than root node
      1. Insert the newnode at left of root node
      2. Else insert newnode at the right of root node
8. If search operation
   1. Read an item to be searched in item
   2. Check if item lesser or greater than the root
   3. If item lesser than root node value
      1. Perform recursive search on the left subtree
      2. Else perform recursive search on the right subtree
9. If deletion operation
   1. Read a key to be deleted from the bst
   2. If key is lesser than the root node’s value
      1. If the element to be deleted is parent node
         1. Replace it with inorder successor
         2. Else replace it with inorder predecessor
      2. If element to be deleted is leaf node, simply delete key
10. Stop

**Source Code:**

#include <stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*l;

struct node \*r;

}\*root=NULL,\*temp=NULL,\*t1,\*t2;

void insert();

void create();

void search(struct node \*t);

void search1(struct node \*t,int data);

void inorder(struct node \*t);

void delete();

void delete1();

int smallest(struct node \*t);

int largest(struct node \*t);

int flag = 1;

int main()

{

int ch;

printf("\nOPERATIONS ---");

printf("\n1 - Insert an element into tree\n");

printf("2 - Inorder Traversal\n");

printf("3 - Delete a node \n");

printf("4 - Exit\n");

do

{

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert();

break;

case 2:inorder(root);

break;

case 3:delete();

break;

case 6:printf("\nInvalid option\n");

exit(0);

default :

printf("Wrong choice, Please enter correct choice ");

break;

}

}while(ch<4);

}

void insert()

{

create();

if (root == NULL)

root = temp;

else

search(root);

}

void create()

{

int data;

printf("Enter data of node to be inserted : ");

scanf("%d", &data);

temp = (struct node \*)malloc(1\*sizeof(struct node));

temp->data = data;

temp->l = temp->r = NULL;

}

void search(struct node \*t)

{

if ((temp->data > t->data) && (t->r != NULL))

search(t->r);

else if ((temp->data> t->data) && (t->r == NULL))

t->r = temp;

else if ((temp->data< t->data) && (t->l != NULL))

search(t->l);

else if ((temp->data < t->data) && (t->l == NULL))

t->l = temp;

}

void inorder(struct node \*t)

{

if (root == NULL)

{

printf("No elements in a tree to display");

return;

}

if (t->l != NULL)

inorder(t->l);

printf("%d ->", t->data);

if (t->r != NULL)

inorder(t->r);

}

void delete()

{

int data;

if (root == NULL)

{

printf("No elements in a tree to delete");

return;

}

printf("Enter the data to be deleted : ");

scanf("%d", &data);

t1 = root;

t2 = root;

search1(root, data);

}

void search1(struct node \*t, int data)

{

if ((data>t->data))

{

t1 = t;

search1(t->r, data);

}

else if ((data < t->data))

{

t1 = t;

search1(t->l, data);

}

else if ((data==t->data))

{

delete1(t);

}

}

void delete1(struct node \*t)

{

int k;

if ((t->l == NULL) && (t->r == NULL))

{

if (t1->l == t)

{

t1->l = NULL;

}

else

{

t1->r = NULL;

}

t = NULL;

free(t);

return;

}

else if ((t->r == NULL))

{

if (t1 == t)

{

root = t->l;

t1 = root;

}

else if (t1->l == t)

{

t1->l = t->l;

}

else

{

t1->r = t->l;

}

t = NULL;

free(t);

return;

}

else if (t->l == NULL)

{

if (t1 == t)

{

root = t->r;

t1 = root;

}

else if (t1->r == t)

t1->r = t->r;

else

t1->l = t->r;

t == NULL;

free(t);

return;

}

else if ((t->l != NULL) && (t->r != NULL))

{

t2 = root;

if (t->r != NULL)

{

k = smallest(t->r);

flag = 1;

}

else

{

k =largest(t->l);

flag = 2;

}

search1(root, k);

t->data = k;

}

}

int smallest(struct node \*t)

{

t2 = t;

if (t->l != NULL)

{

t2 = t;

return(smallest(t->l));

}

else

return (t->data);

}

int largest(struct node \*t)

{

if (t->r != NULL)

{

t2 = t;

return(largest(t->r));

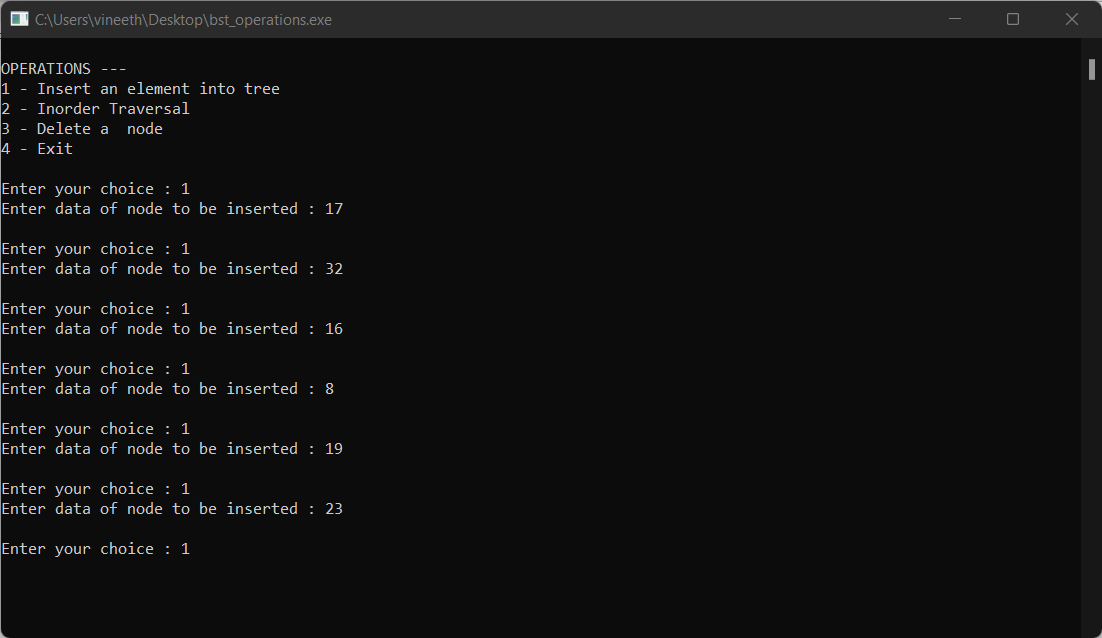
}

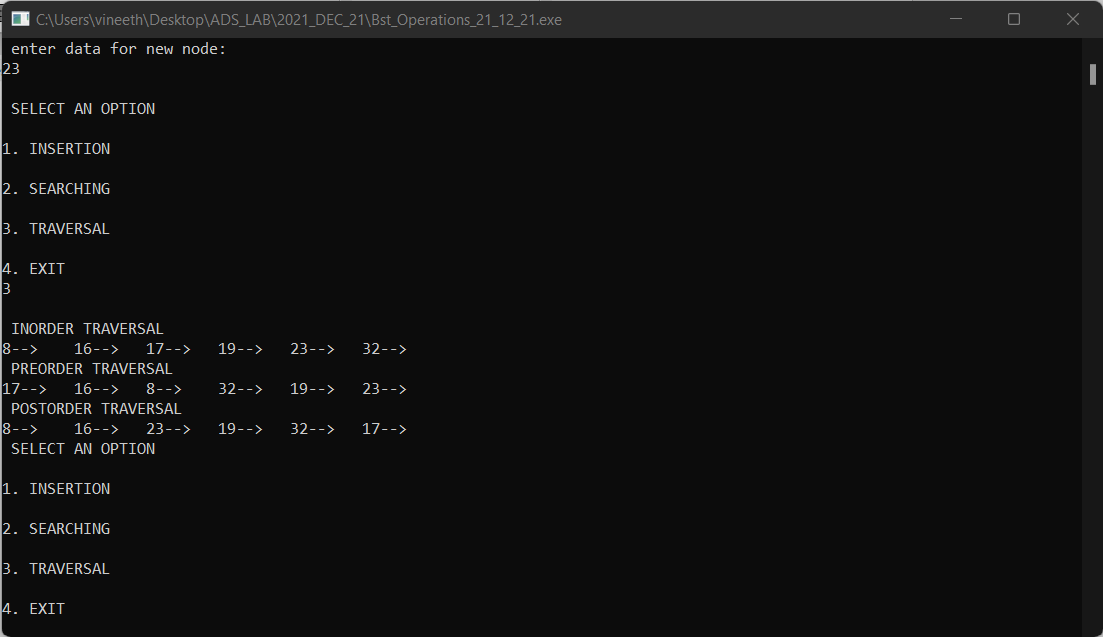
else

return(t->data);

}

**Output:**

****

****

****

**Program no:7**

**Aim: To implement set data structure and perform it’s operations**

**Algorithm:**

1. Start
2. Create two character array
3. Enter a bit string in array 1
4. Enter another bit string in array2
5. Display menu of operations
6. If union():
   1. For i =0 to strlen(array): print(aray1[i] or array2[i])
7. If intersection():
8. For i=0 to strlen(array): print(array1[i] and array2[i])

8. If set difference():

a. Declare an array3 for complementing array2

1. Store bitwise negation results on array2 in array3
2. For i=0 to strlen(array): print(array1[i] or array3[i])

9. Stop

**Source Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int main()

{

int len,i=0;

int str1[10],str2[10],str3[10];

int opt,c;

printf("\n Enter the length of the sets: ");

scanf("%d",&len);

printf("\n Enter the bits of first set:\n");

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

{

scanf("%d",&c);

if(c<0 || c>1)

printf("\n Input Error \n Please enter in binary format\n");

else

str1[i]=c;

}

printf("\n Enter the bits of second set:\n");

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

{

scanf("%d",&c);

if(c<0 || c>1)

printf("\n Input Error\n Please enter in binary format\n");

else

str2[i]=c;

}

printf("\n Union of the above set is\n");

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

{

if((str1[i]||str2[i])==1)

printf("1 ");

else

printf("0 ");

}

printf("\n Intersection of the above set is\n");

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

{

printf(str1[i]\*str2[i]);

}

printf("\n The set difference of the above 2 sets are:\n");

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

{

str3[i]=!(str2[i]);

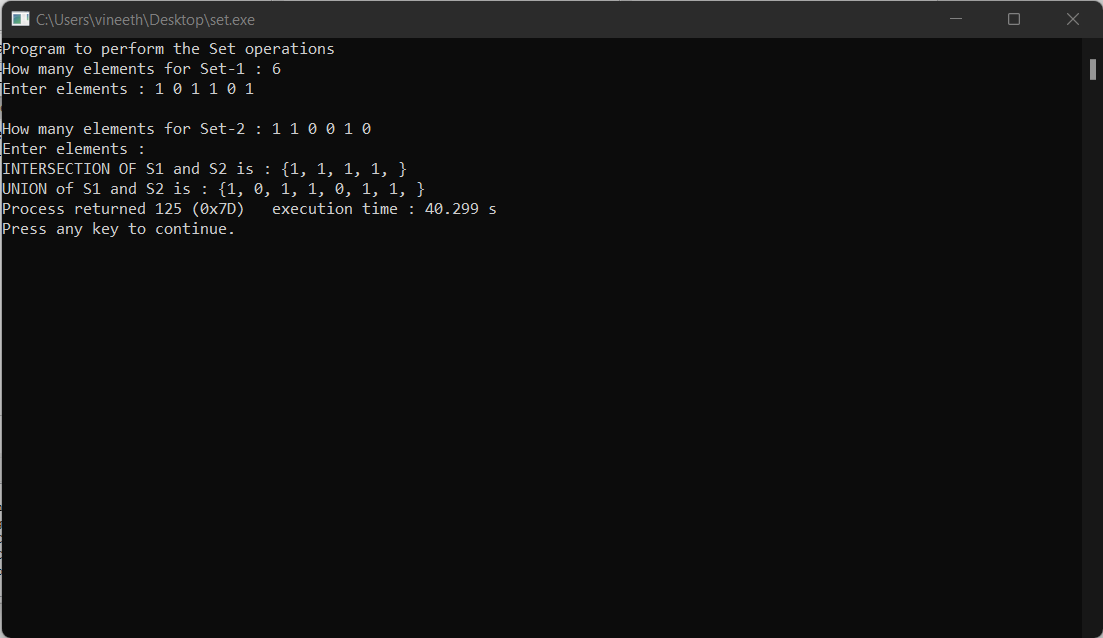
str3[i]=str1[i]||str3[i];

printf("%d ",str3[i]);

}

}

**Output:**

****

**Program no: 8**

**Aim: To perform graph traversal techniques(BFS & DFS) and topological sorting**

**Algorithm:**

**a)BFS:**

1. Create a queue Q

2. Mark v as visited and put v into Q

3. while Q is non-empty

4. remove the head u of Q

5. mark and enqueue all (unvisited) neighbours of u

**(b)DFS:**

DFS(G, u)

u.visited = true

for each v ∈ G.Adj[u]

if v.visited == false

DFS(G,v)

init() {

For each u ∈ G

u.visited = false

For each u ∈ G

DFS(G, u)

}

**(c)TOPOLOGICAL SORTING:**

1. For each vertex UϵV
2. do indegree [U]<-0
3. for each vertex UϵV
4. do for each VϵAdj[U]
5. do indegree[V]<-indegree[V]+1
6. Q<-Ø
7. For each vertex UϵV
8. Do if indegree [U]=0
9. Then EnQueue (Q,U)
10. While Q≠Ø
11. Do U<-DEQUEUE(Q)
12. Output U
13. For each VϵAdj [U]
14. Do indegree [V]<- indegree[V]-1
15. If indegree[V] = 0
16. Then EnQueue(Q,V)
17. Do if indegree[U]≠0
18. Repeat there is a cycle.

**Source Code:**

**(a)BFS:**

#include<stdio.h>

int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

void bfs(int v)

{

for(i = 1; i <= n; i++)

if(a[v][i] && !visited[i])

q[++r] = i;

if(f <= r) {

visited[q[f]] = 1;

bfs(q[f++]);

}

}

int main()

{

int v;

printf("\n Enter the number of vertices:");

scanf("%d", &n);

for(i=1; i <= n; i++)

{

q[i] = 0;

visited[i] = 0;

}

printf("\n Enter graph data in matrix form:\n");

for(i=1; i<=n; i++)

{

for(j=1;j<=n;j++)

{

scanf("%d", &a[i][j]);

}

}

printf("\n Enter the starting vertex:");

scanf("%d", &v);

bfs(v);

printf("\n The node which are reachable are:\n");

for(i=1; i <= n; i++)

{

if(visited[i])

printf("%d\t", i);

else

{

printf("\n Bfs is not possible. Not all nodes are reachable");

break;

}

}

}

**(b)DFS:**

#include<stdio.h>

int a[20][20],reach[20],n;

int dfs(int v)

{

int i;

reach[v]=1;

for (i=1;i<=n;i++)

if(a[v][i] && !reach[i])

{

printf("\n %d->%d",v,i);

dfs(i);

}

}

int main()

{

int i,j,count=0;

printf("\n Enter number of vertices:");

scanf("%d",&n);

for (i=1;i<=n;i++)

{

reach[i]=0;

for (j=1;j<=n;j++)

a[i][j]=0;

}

printf("\n Enter the adjacency matrix:\n");

for (i=1;i<=n;i++)

for (j=1;j<=n;j++)

scanf("%d",&a[i][j]);

dfs(1);

printf("\n");

for (i=1;i<=n;i++)

{

if(reach[i])

count++;

}

if(count==n)

printf("\n Graph is connected"); else

printf("\n Graph is not connected");

return 0;

}

**(c)TOPOLOGICAL SORTING:**

#include <stdio.h>

int main(){

int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;

printf("Enter the no of vertices:\n");

scanf("%d",&n);

printf("Enter the adjacency matrix:\n");

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

printf("Enter row %d\n",i+1);

for(j=0;j<n;j++)

scanf("%d",&a[i][j]);

}

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

indeg[i]=0;

flag[i]=0;

}

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

for(j=0;j<n;j++)

indeg[i]=indeg[i]+a[j][i];

printf("\nThe topological order is:");

while(count<n){

for(k=0;k<n;k++){

if((indeg[k]==0) && (flag[k]==0)){

printf("%d ",(k+1));

flag [k]=1;

}

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

if(a[i][k]==1)

indeg[k]--;

}

}

count++;

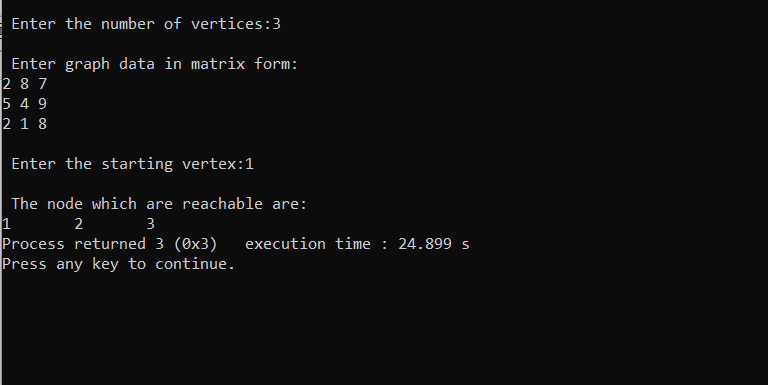
}

Return

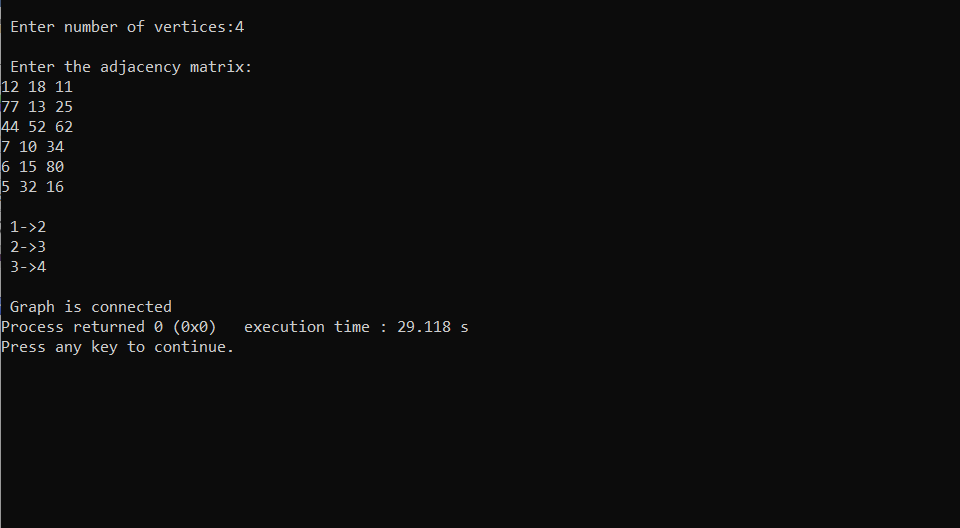
}

**OUTPUT**

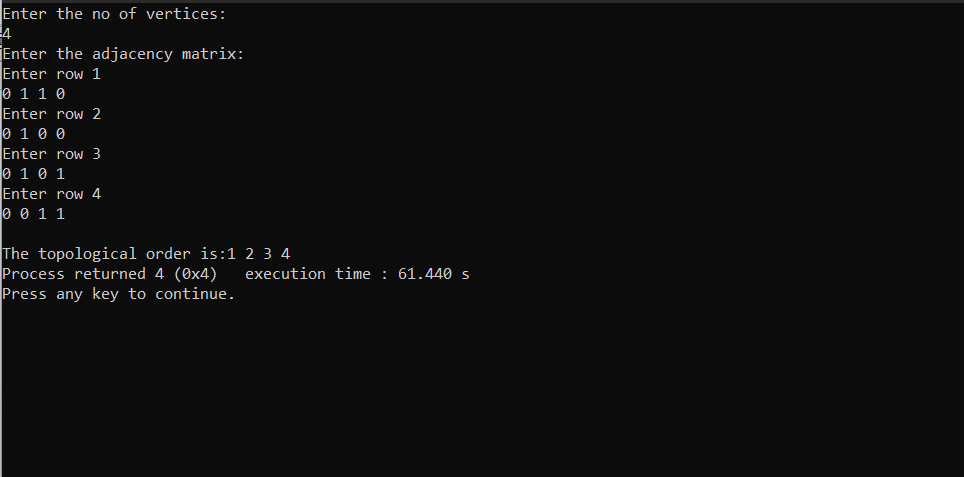
**BFS**

****

**DFS**

****

**TOPOLOGICAL SORTING**



**Program no:9**

**Aim: To implement prim’s algorithm for finding Minimum Cost Spanning Tree**

**Algorithm:**

MST PRIMS(G, w,t)

1. For each uϵV[G]
2. Do key[u]<-∞
3. П[u]<-NIL
4. Key[П]<- 0
5. Q<-V[G]
6. While Q≠Ø

Do u<-extract min(Q)

For each VϵAdj[u]

Do if VϵQ and w[u,v]<key[V]

Then П[V]<-u

Key[V]<-w[u,v]

**Source Code:**

#include<stdio.h>

#include<stdbool.h>

#define INF 9999999

#define V 5

int G[V][V] = {

{0, 9, 75, 0, 0},

{9, 0, 95, 19, 42},

{75, 95, 0, 51, 66},

{0, 19, 51, 0, 31},

{0, 42, 66, 31, 0}};

int main()

{

int no\_edge; // number of edge

int selected[V];

memset(selected, false, sizeof(selected));

no\_edge = 0;

selected[0] = true;

int x;

int y;

printf("Edge : Weight\n");

while (no\_edge < V - 1)

{

int min = INF;

x = 0;

y = 0;

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

{

if (selected[i])

{

for (int j = 0; j < V; j++)

{

if (!selected[j] && G[i][j])

{

if (min > G[i][j]) {

min = G[i][j];

x = i;

y = j;

}

}

}

}

}

printf("%d - %d : %d\n", x, y, G[x][y]);

selected[y] = true;

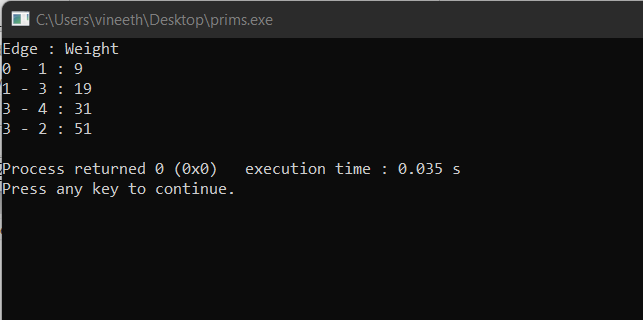
no\_edge++;

}

return 0;

}

**Output:**



**Program no:10**

**Aim: To implement Kruskal’s algorithm for finding Minimum Cost Spanning Tree**

**Algorithm:**

**KRUSKAL(G):**

A = ∅

For each vertex v ∈ G.V:

MAKE-SET(v)

For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):

if FIND-SET(u) ≠ FIND-SET(v):

A = A ∪ {(u, v)}

UNION(u, v)

return A

**Source Code:**

#include <stdio.h>

#define MAX 30

typedef struct edge

{

int u, v, w;

} edge;

typedef struct edge\_list

{

edge data[MAX];

int n;

}

edge\_list;

edge\_list elist;

int Graph[MAX][MAX], n;

edge\_list spanlist;

void kruskalAlgo();

int find(int belongs[], int vertexno);

void applyUnion(int belongs[], int c1, int c2);

void sort();

void print();

void kruskalAlgo()

{

int belongs[MAX], i, j, cno1, cno2;

elist.n = 0;

for (i = 1; i < n; i++)

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

{

if (Graph[i][j] != 0)

{

elist.data[elist.n].u = i;

elist.data[elist.n].v = j;

elist.data[elist.n].w = Graph[i][j];

elist.n++;

}

}

sort();

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

belongs[i] = i;

spanlist.n = 0;

for (i = 0; i < elist.n; i++)

{

cno1 = find(belongs, elist.data[i].u);

cno2 = find(belongs, elist.data[i].v);

if (cno1 != cno2)

{

spanlist.data[spanlist.n] = elist.data[i];

spanlist.n = spanlist.n + 1;

applyUnion(belongs, cno1, cno2);

}

}

}

int find(int belongs[], int vertexno)

{

return (belongs[vertexno]);

}

void applyUnion(int belongs[], int c1, int c2) {

int i;

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

if (belongs[i] == c2)

belongs[i] = c1;

}

void sort() {

int i, j;

edge temp;

for (i = 1; i < elist.n; i++)

for (j = 0; j < elist.n - 1; j++)

if (elist.data[j].w > elist.data[j + 1].w) {

temp = elist.data[j];

elist.data[j] = elist.data[j + 1];

elist.data[j + 1] = temp;

}

}

void print()

{

int i, cost = 0;

for (i = 0; i < spanlist.n; i++)

{

printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);

cost = cost + spanlist.data[i].w;

}

printf("\nSpanning tree cost: %d", cost);

}

int main()

{

int i, j, total\_cost;

n = 6;

Graph[0][0] = 0;

Graph[0][1] = 4;

Graph[0][2] = 4;

Graph[0][3] = 0;

Graph[0][4] = 0;

Graph[0][5] = 0;

Graph[0][6] = 0;

Graph[1][0] = 4;

Graph[1][1] = 0;

Graph[1][2] = 2;

Graph[1][3] = 0;

Graph[1][4] = 0;

Graph[1][5] = 0;

Graph[1][6] = 0;

Graph[2][0] = 4;

Graph[2][1] = 2;

Graph[2][2] = 0;

Graph[2][3] = 3;

Graph[2][4] = 4;

Graph[2][5] = 0;

Graph[2][6] = 0;

Graph[3][0] = 0;

Graph[3][1] = 0;

Graph[3][2] = 3;

Graph[3][3] = 0;

Graph[3][4] = 3;

Graph[3][5] = 0;

Graph[3][6] = 0;

Graph[4][0] = 0;

Graph[4][1] = 0;

Graph[4][2] = 4;

Graph[4][3] = 3;

Graph[4][4] = 0;

Graph[4][5] = 0;

Graph[4][6] = 0;

Graph[5][0] = 0;

Graph[5][1] = 0;

Graph[5][2] = 2;

Graph[5][3] = 0;

Graph[5][4] = 3;

Graph[5][5] = 0;

Graph[5][6] = 0;

kruskalAlgo();

print();

}

**Output:**

