GCC command in Linux with examples

GCC stands for GNU Compiler Collections which is used to compile mainly C and C++ language. It can also be used to compile Objective C and Objective C++. The most important option required while compiling a source code file is the name of the source program, rest every argument is optional like a warning, debugging, linking libraries, object file etc. The different options of gcc command allow the user to stop the compilation process at different stages.

Syntax:

gcc [-c|-S|-E] [-std=standard]

Example: This will compile the source.c file and give the output file as a.out file which is default name of output file given by gcc compiler, which can be executed using ./a.out

gcc source.c

Most Usefull Options with Examples: Here source.c is the C program code file.

-o opt: This will compile the source.c file but instead of giving default name hence executed using ./opt, it will give output file as opt. -o is for output file option.

gcc source.c -o opt

-Werror: This will compile the source and show the warning if any error is there in the program, -W is for giving warnings.

gcc source.c -Werror -o opt

-Wall: This will check not only for errors but also for all kinds warning like unused variables errors, it is good practice to use this flag while compiling the code.

gcc source.c -Wall -o opt

-ggdb3: This command give us permissions to debug the program using gdb which will be described later, -g option is for debugging.

gcc -ggdb3 source.c -Wall -o opt-lm : This command link math.h library to our source file, -l option is used for linking

particular library, for math.h we use -lm.

gcc -Wall source.c -o opt -lm

-std=c11 :This command will use the c11 version of standards for compiling the source.c program, which allows to define variable under loop initializations also using newer standards version is preferred.

gcc -Wall -std=c11 source.c -o opt

-c : This command compile the program and give the object file as output, which is used to make libraries.

-v : This option is used for the verbose purpose.

GDB command in Linux with examples

gdb is the acronym for GNU Debugger. This tool helps to debug the programs written in C, C++, Ada, Fortran, etc. The console can be opened using the gdb command on terminal.

Syntax:

gdb [-help] [-nx] [-q] [-batch] [-cd=dir] [-f] [-b bps] [-tty=dev] [-s symfile] [-e prog] [-se prog] [-c core] [-x cmds] [-d dir] [prog[core|procID]]

Example:

The program to be debugged should be compiled with -g option. The below given C++ file that is saved as gfg.cpp. We are going to use this file in this article.

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
using namespace std;
int findSquare(int a)
{
 return a * a; }
int main(int n, char** args)
{
 for (int i = 1; i < n; i++)
 {
 int a = atoi(args[i]);
 cout << findSquare(a) << endl;
```

```
 }
 return 0;
}
```

Compile the above C++ program using the command:

g++ -g -o gfg gfg.cpp

To start the debugger of the above gfg executable file, enter the command gdb gfg. It opens the gdb console of the current program, after printing the version information.

run [args] : This command runs the current executable file. In the below image, the program was executed twice, one with the command line argument 10 and another with the command line argument 1, and their corresponding outputs were printed.

quit or q : To quit the gdb console, either quit or q can be used.

help : It launches the manual of gdb along with all list of classes of individual commands.

break : The command break [function name] helps to pause the program during execution when it starts to execute the function. It helps to debug the program at that point. Multiple breakpoints can be inserted by executing the command wherever necessary. b findSquare command makes the gfg executable pause when the debugger starts to execute the findSquare function.

bbreak [function name]

break [file name]:[line number]

break [line number]

break *[address]

break **any of the above arguments** if [condition]

b **any of the above arguments**

In the above example, the program that was being executed(run 10 100), paused when it encountered findSquare function call. The program pauses whenever the function is called. Once the command is successful, it prints the breakpoint number, information of the program counter, file name, and the line number. As it encounters any breakpoint during execution, it prints the breakpoint number,

function name with the values of the arguments, file name, and line number. The breakpoint can be set either with the address of the instruction(in hexadecimal form preceded with *0x) or the line number and it can be combined with if condition(if the condition fails, the breakpoint will not be set) For example, break findSquare if a == 10.

continue : This command helps to resume the current executable after it is paused by the breakpoint. It executes the program until it encounters any breakpoint or runs time error or the end of the program. If there is an integer in the argument(repeat count), it will consider it as the continue repeat count and will execute continue command “repeat count” number of times.

continue [repeat count]

c [repeat count]

next or n : This command helps to execute the next instruction after it encounters the breakpoint.

Whenever it encounters the above command, it executes the next instruction of the executable by printing the line in execution.

delete : This command helps to deletes the breakpoints and checkpoints. If the delete command is executed without any arguments, it deletes all the breakpoints without modifying any of the checkpoints. Similarly, if the checkpoint of the parent process is deleted, all the child checkpoints are automatically deleted.

d

delete

delete [breakpoint number 1] [breakpoint number 2] ...

delete checkpoint [checkpoint number 1] [checkpoint number 2] ...

In the above example, two breakpoints were defined, one at the main and the other at the findSquare. Using the above command findSquare breakpoint was deleted. If there is no argument after the command, the command deletes all the breakpoints.

clear : This command deletes the breakpoint which is at a particular function with the name FUNCTION_NAME. If the argument is a number, then it deletes the breakpoint that lies in that particular line.

clear [line number]

clear [FUNCTION_NAME]

In the above example, once the clear command is executed, the breakpoint is deleted after printing the breakpoint number.

disable [breakpoint number 1] [breakpoint number 2] …. : Instead of deleting or clearing the breakpoints, they can be disabled and can be enabled whenever they are necessary.

enable [breakpoint number 1] [breakpoint number 2] …. : To enable the disabled breakpoints, this command is used.

info : When the info breakpoints in invoked, the breakpoint number, type, display, status, address, the location will be displayed. If the breakpoint number is specified, only the information about that particular breakpoint will be displayed. Similarly, when the info checkpoints are invoked, the checkpoint number, the process id, program counter, file name, and line number are displayed.

info breakpoints [breakpoint number 1] [breakpoint number 2] ...

info checkpoints [checkpoint number 1] [checkpoint number 2] ...checkpoint command and restart command : These command creates a new process and keep that process in the suspended mode and prints the created process's process id.

For example, in the above execution, the breakpoint is kept at function findSquare and the program was executed with the arguments "1 10 100". When the function is called initially with a = 1, the breakpoint happens. Now we create a checkpoint and hence gdb returns a process id(4272), keeps it in the suspended mode and resumes the original thread once the continue command is invoked. Now the breakpoint happens with a = 10 and another checkpoint(pid = 4278) is created. From the info checkpoint information, the asterisk mentions the process that will run if the gdb encounters a continue. To resume a specific process, restart command is used with the argument that specifies the serial number of the process. If all the process are finished executing, the info checkpoint command returns nothing.

set args [arg1] [arg2] … : This command creates the argument list and it passes the

specified arguments as the command line arguments whenever the run command without any argument is invoked. If the run command is executed with arguments after set args, the arguments are updated. Whenever the run command is ran without the arguments, the arguments are set by default.

show args : The show args prints the default arguments that will passed if the run command is executed. If either set args or run command is executed with the arguments, the default arguments will get updated, and can be viewed using the above show args command.

display [/format specifier] [expression] and undisplay [display id1] [display id2] … : These command enables automatic displaying of expressions each time whenever the execution encounters a breakpoint or the n command. The undisplay command is used to remove display expressions. Valid format specifiers are as follows:

o - octal

x - hexadecimal

d - decimal

u - unsigned decimalt - binary

f - floating point

a - address

c - char

s - string

i - instruction

In the above example, the breakpoint is set at line 12 and ran with the arguments 1 10 100. Once the breakpoint is encountered, display command is executed to print the value of i in hexadecimal form and value of args[i] in the string form. After then, whenever the command n or a breakpoint is encountered, the values are displayed again until they are disabled using undisplay command.

print : This command prints the value of a given expression. The display command prints all the previously displayed values whenever it encounters a breakpoint or the next command, whereas the print command saves all the previously displayed values and prints whenever it is called.

print [Expression]

print $[Previous value number]

print {[Type]}[Address]

print [First element]@[Element count]

print /[Format] [Expression]

file : gdb console can be opened using the command gdb command. To debug the executables from the console, file [executable filename] command is used.

GPROF Tutorial – How to use Linux GNU GCC Profiling Tool

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.

In this article, we will explore the GNU profiling tool 'gprof'.

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).

The last step above produces an analysis file which is in human readable form. This file contains a couple of tables (flat profile and call graph) in addition to some other information. While flat profile gives an overview of the timing information of the functions like time consumption for the execution of a particular function, how many times it was called etc. On the other hand, call graph focuses on each function like the functions through which a particular function was called, what all functions were called from within this particular function etc So this way one can get idea of the execution time spent in the sub-routines too.

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.

From the man page of gcc :

-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.

So, 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.

Note that while execution if the program changes the current working directory (using chdir) then gmon.out will be produced in the new current working directory. Also, your program needs to have sufficient permissions for gmon.out to be created in 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.txtNote 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.

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

4. Print information related to specific function in flat profileThis 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.tx

WRITE A C PROGRAM TO MERGE TWO SORTED ARRAY.

AIM: Two merge two sorted array and store it in the third array

ALGORITHM:

STEP1: START
STEP2: Declare two arrays a and b and read the elements
STEP3: Traverse both the arrays
STEP4: Find the minimum element among the current two elements of both
the arrays and update the merge array with the least value and increment its index to next position.
STEP5: Repeat procedure until both arrays are exhausted and return merge
array.
STEP6: STOP

SOURCECODE:

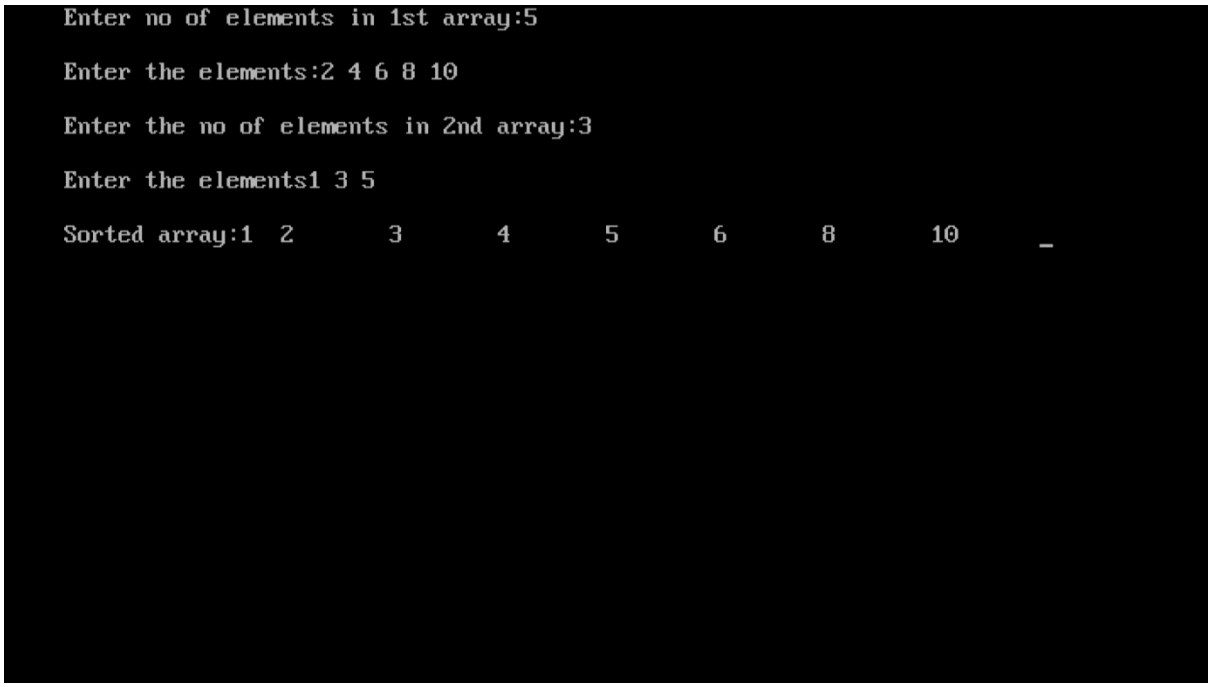
```
#include<stdio.h>
#include<conio.h>
void merge (int [], int, int [], int, int []);
void main ()
{
 int a [100], b [100], m, n, c, sort [200];
 clrscr();
 printf("Enter no of elements in 1st array:");
 scanf("%d", &m);
 printf("\nEnter the elements:", m); for (c=0; c<m;c++)
 {
 scanf("%d", &a[c]);
 }
 printf("\nEnter the no of elements in 2nd array:");
```

```
 scanf("%d", &n);
 printf("\nEnter the elements", n);
 for (c = 0; c < n; c++)
 {
 scanf("%d", &b[c]);
 }
 merge (a, m, b, n, sort);
printf("\nSorted array:");
 for (c = 0; c < m + n; c++) {
 printf("%d\t", sort[c]);
 }
 getch();
}
void merge (int a [], int m, int b [], int n, int sorted []) {
int i, j, k;
 j = k = 0;
 for (i = 0; i < m + n;) {
 if (j < m && k < n) {
 if (a[j] < b[k]) {sorted[i] = a[j];
j++;
 }
 else {
sorted[i] = b[k];
 k++;
 }
 i++; }
 else if (j == m) {
 for (; i < m + n;) {
 sorted[i] = b[k];
```

```
 k++;
 i++; }
 }
 else {
 for (; i < m + n;) {
 sorted[i] = a[j];
 j++;
 i++;
 }
 }
 }
}
```

RESULT: THE REQUIRED OUTPUT IS OBTAINED.

OUTPUT


Enter no of elements in 1st array:5
Enter the elements:2 4 6 8 10
Enter the no of elements in 2nd array:3
Enter the elements1 3 5
Sorted array:1 2 3 4 5 6 8 10

2 WRITE A C PROGRAM TO IMPLEMENT STACK AS ARRAY.

AIM:Program to implement stack as array.

ALGORITHM:

Step 1: Include all the header files which are used in the program.

Step 2: Declare all the functions used in stack implementation.

Step 3: Create a one-dimensional array with fixed size (int stack [SIZE])

Step 4: Define an integer variable 'top' and initialize with '-1'. (int top = -1)

Step 5: In main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

PUSH

Step 1: Check whether the stack is FULL. (top == SIZE-1)

Step 2: If it is FULL, then display "Stack Overflow" and terminate the function.

Step 3: If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value top] = value).

POP

Step 1: Check whether stack is EMPTY. (top == -1)

Step 2: If it is EMPTY, then display "Stack is Underflow" and terminate the function.

Step 3: If it is NOT EMPTY, then delete stack[top] and decrement top value by one (top--).

DISPLAY

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--).

Step 4: Repeat above step until i value becomes '0'.

DISPLAY TOP ELEMENT

Step 1: Declare val to store top element.

Step 2: If it is EMPTY, then display “Stack is Empty” and terminate the function.

Step 3: If it is not EMPTY, then val=stack[top] then display val.

SOURCECODE:

```
 #include<stdio.h>
int a[50],top=-1,size,choice=0,i;
void push();
void pop();
void display();
void sttop();
void main()
{
clrscr();
printf("enter the number of elements in the stack\n");
scanf("%d",&size);
while(choice!=5)
{
printf("\nselect an operation \n");
printf("\n1.push \n2.pop \n3.display \n4.Stack top \n5.exit");
printf("\nenter your choice");
scanf("%d",&choice);
switch(choice)
{
case 1:
{
push();
break;
}
case 2:
{
pop();
```

```
break;
}
case 3:
{
display();
break;
}
case 4:
{
sttop();
break;
}
case 5:
{ printf("Exiting...");

  break;
 }
default:
printf("invalid");
}
}
}

void push()
{
int item;
if(top>size-1)
printf("the stack is full");
else
{
printf("enter the element to be inserted");
scanf("%d",&item);
top=top+1;
a[top]=item;
}
}
void pop()
{
if(top==-1)
printf("the stack is emepty");
else
{
top=top-1;
}
}
void display()
{
```

```
for(i=top;i>=0;i--)
{
printf("%d\t",a[i]);
}
if(top==-1)
{
printf("stack is empty");
}}
void sttop()
{ int item;
 if(top==-1)
 { printf("\n Stack is empty...");
   }
   else

  printf("\n%d",a[top]);
  }
```

RESULT: THE REQUIRED OUTPUT IS OBTAINED.

OUTPUT

```
2.pop
3.display
4.Stack top
5.exit
enter your choice1
enter the element to be inserted3

select an operation

1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice1
enter the element to be inserted4

select an operation

1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice1_
```

```
1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice1
enter the element to be inserted5

select an operation

1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice3
5       4       3
select an operation

1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice
```

```
1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice2

select an operation

1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice3
4       3
select an operation

1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice
```

```
1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice1
enter the element to be inserted78

select an operation

1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice3
78      7       4       3
select an operation

1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice_
```

```
1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice3
78      7       4       3
select an operation

1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice4

78
select an operation

1.push
2.pop
3.display
4.Stack top
5.exit
enter your choice
```

3. WRITE A C PROGRAM TO IMPLEMENT QUEUE USING ARRAY.

AIM: program to implement queue using array

ALGORITHM:

Step 1: start

Step 2: Include all the header files which are used in the program.

Step 3: Declare all the functions used in stack implementation.

Step 4: Create a one-dimensional array with fixed size (int stack [SIZE])

Step 5: Define an integer variable 'top' and initialize with '-1'. (int top = -1)

Step 6: In main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

Step 7 : stop

INSERT

Step 1: start

Step 2: define a variable item.

Step 3 : check the condition of queue overflow.[if(rear==maxsize-1)], if the condition is true display “queue overflow” ,and terminate the function.

Step 4: if it is NOT FULL ,then increment the value of rear by one and insert the item to the queue[rear].

Step 5: stop.

DELETE

Step 1: start.

Step 2: define a variable item.

Step3 : check the condition of queue underflow,[if(front==-1)],if the condition is true display “queue underflow”,and terminate the function.

Step 4: if it is NOT EMPTY,then delete q[front] and increment front by one.
Step 5 : stop.

DISPLAY

Step 1: start

Step 2: declare a variable i.

Step 3: check whether the queue is empty[rear==-1],if it is empty ,then display ”The queue is empty” and terminate the function.

Step 4 : if it is not empty initialize the variable i with front. display q[i] value and increment i value by one(i++)

Step 5: repeat the above step until I value become 0.

Step 6 : stop

DISPLAY THE FRONT ELEMENT

Step 1: declare a variable to store the front element, item

Step 2. If the queue is empty display “The queue is empty” and terminate the function.

Step 3: if it is not empty ,then item=q[front],then display the front element.

DISPLAY THE REAR ELEMENT

Step 1: declare a variable to store the front element, item

Step 3. If the queue is empty display “The queue is empty” and terminate the function.

Step 4: if it is not empty ,then item=q[rear],then display the front element.

SOURCECODE:

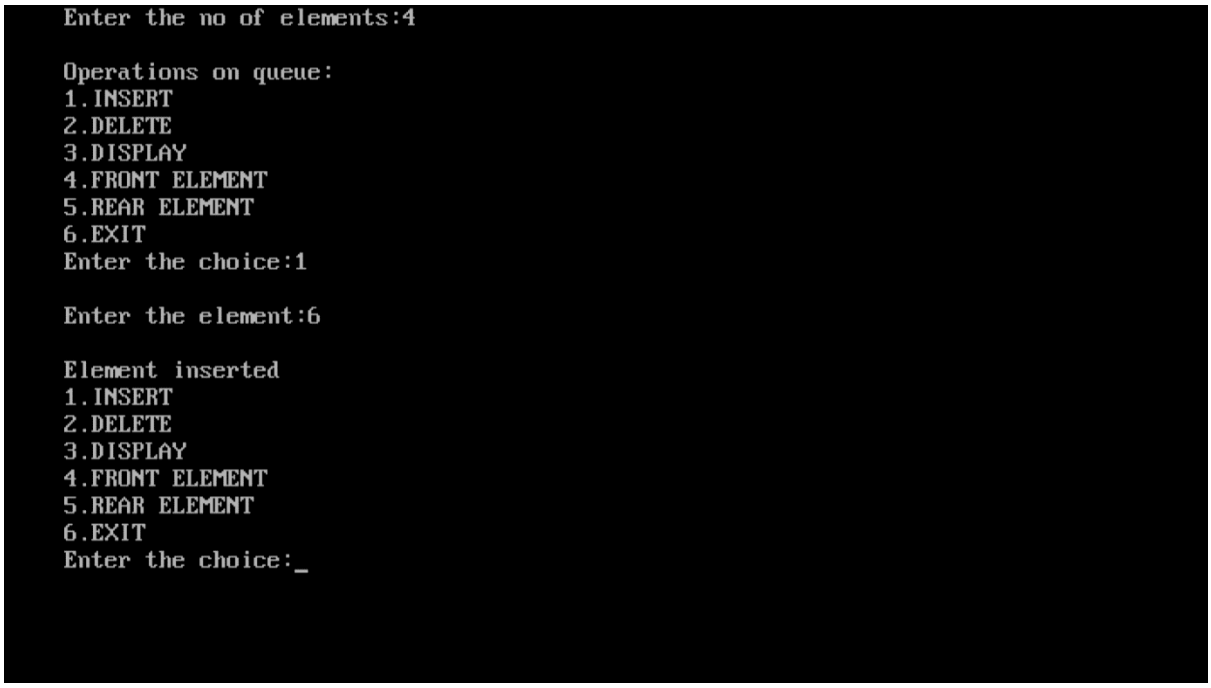
```
#include<stdio.h>
#include<conio.h>
#define maxsize 25
void insert();
void delete();
void display();
void fele();
void rele();
int front=-1,rear=-1;
int q[maxsize];
void main()
{
  int ch,n;
  clrscr();
  printf("Enter the no of elements:");
  scanf("%d",&n);
  printf("\nOperations on queue:");
  while(ch!=6)
  {
   printf("\n1.INSERT\n2.DELETE\n3.DISPLAY\n4.FRONT ELEMENT\n5.REAR
ELEMENT\n6.EXIT");
   printf("\nEnter the choice:");
   scanf("%d",&ch);
   switch(ch)
    {
    case 1:{
              insert();
              break;
              }
    case 2:{
              delete();
              break;
              }
    case 3:{
              display();
              break;
              }
    case 4:{
              fele();
              break;
```

```
            }
case 5:{
            rele();
            break;
            }
case 6:{
           printf("\nExiting");
           break;
           }
default:printf("\nInvalid choice");
}
};
getch();
}
void insert()
{
 int item;
 printf("\nEnter the element:");
 scanf("%d",&item);
 if(rear==maxsize-1)
  {
  printf("\nOverflow");
  return;
  }
 if(front==-1 && rear==-1)
 {
 front=0;
 rear=0;
 }
 else
   rear++;
 q[rear]=item;
 printf("\nElement inserted");
 }
 void delete()
 {
  int item;
  if(front==-1 || front>rear)
  {
  printf("\nUnderflow");
  return;
  }
  else
  {
  item=q[front];
  if(front==rear)
  {
```

```
front=-1;
rear=-1;
}
else
 front++;
 }
printf("Element deleted");
}
void display()
{
 int i;
 if(rear==-1)
      printf("\nEmpty queue");
 else
      {
       printf("\nThe elements are:");
       for(i=front;i<=rear;i++)
       {
        printf("%d\t",q[i]);
        }
       }
       }
      void fele()
      {
      int item;
      if(front==-1)
       printf("\nEmpty queue");
      else
      {
      item=q[front];
      printf("\nThe front element is %d",item);
      }
      }
      void rele()
      {
      int item;
      if(rear==-1)
      printf("\nEmpty queue");
      else
      {
      item=q[rear];
      printf("The rear element is %d",item);
      }}
```

RESULT: THE REQUIRED OUTPUT IS OBTAINED.

OUTPUT


Enter the no of elements:4
Operations on queue:
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:1
Enter the element:6
Element inserted
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:_

```
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:3

The elements are:6      11      34      67      7       4       9
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:4

The front element is 6
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:
```

```
The elements are:6      11      34      67      7       4       9
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:4

The front element is 6
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:5
The rear element is 9
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:_
```

```
The front element is 6
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:5
The rear element is 9
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:2
Element deleted
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:_
```

```
The elements are:11     34      67      7       4       9
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:2
Element deleted
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:3

The elements are:34     67      7       4       9
1.INSERT
2.DELETE
3.DISPLAY
4.FRONT ELEMENT
5.REAR ELEMENT
6.EXIT
Enter the choice:
```

6. WRITE A C PROGRAM TO IMPLEMENT CIRCULAR QUEUE AND ITS OPERATIONS

AIM: Program to implement circular queue.

ALGORITHM:

Step 1 : Include all the header files which are used in the program and define a constant 'MAX' with specific value.

Step 2 : Create a one dimensional array with above defined MAX (queue[MAX]) Define two integer variables 'front' and 'rear' and initialize both with '-1'.

Step 3: Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on circular queue.

Enqueue()

Step 1: If queue is FULL, then display "Queue Overflow".

Step 2 : If it is NOT FULL, then check rear == MAX - 1 && front != 0 if it is TRUE, then set rear = -1.

Step 3 : Increment rear value by one (rear++), set queue[rear] = value and check 'front ==-1' if it is TRUE, then set front = 0.

dequeue()

Step 1 : If queue is EMPTY, then display "Queue is EMPTY!!!

Step 2 : If it is NOT EMPTY, then display queue[front] as deleted element and increment the front value by one . Then check whether front ==MAX, if it is TRUE, then set front = 0. Then check whether both front –1 and rear are equal if it TRUE, then set both front and rear to '-1' (front = rear = -1).

search()
Step 1 : If queue is EMPTY, then display "Queue is EMPTY!!! . Step 2 : While queue[front]>queue[rear] then check if queue[front]==n then element found.

Step 3: Else element not found.

Display()
Step 1 : If queue is EMPTY, then display "Queue is EMPTY!!! .

Step 2: Check (front_pos>rear_pos) if it is true iterate the while loop when (front_pos>rear_pos) and print queue[front_pos]

Step 3:Check (front_pos<=rear_pos) if it is true then define a variable 'i' and initialize with front_pos. Display queue[i] value and increment i value by one (i++)

Step 4: Repeat above step until i<=rear_pos.

SOURCE CODE:

```
#include<stdio.h>
#define MAX 10
void enqueue();
void dequeue();
void search();
void display();
int c,queue[MAX],item,front=-1,rear=-1,i,n,flag=0;
void main()
{
do
{
printf("\n1)Insertion\n2)Deletion\n3)Search\n4)Display\n5)Exit\n");
scanf("%d",&c);
switch(c)
{
case 1:enqueue();
 break;
case 2:dequeue();
 break;
case 3:search();
 break; case 4:display();
 break;
case 5:printf("\n***************Exit point**************\n");
 break;
default:printf("\n INVALID CHOICE");
}
}while(c!=5);
}
void enqueue()
{
if((rear+1)%MAX==front){
printf("\n***************** QUEUE OVERFLOW*****************");
}
```

```
else
{
if(rear==-1)
front=0;
printf("\nEnter the element to insert : ");
scanf("%d",&item);
rear=(rear+1)%MAX;
queue[rear]=item;
}
}
void dequeue()
{
if(front==-1){
printf("\n**********************QUEUE
UNDERFLOW***************");
}
else if(front==rear){
printf("\nThe deleted element is:%d\n",queue[front]);
front=rear=-1;
}
else{
printf("\nThe deleted element is:%d\n",queue[front]);
front=(front+1)%MAX;}
}
void search()
{
printf("\nEnter the value to search\n");
scanf("%d",&n);
if(front==-1){
printf("\n**********************QUEUE
UNDERFLOW***************\n");
}
else
{
int front_pos,rear_pos;
front_pos=front,rear_pos=rear;
if(front_pos>rear_pos){
while(front_pos>rear_pos)
{
if(queue[front_pos]==n){
flag=1;
break;
}
front_pos=((front_pos+1))%MAX;
}
}
if(front_pos<=rear_pos){
```

```
for(i=front_pos;i<=rear_pos;i++){
if(queue[i]==n){
flag=1;
break;
}
}
}
}
if(flag==1)
printf("\nElement is found ");
else
printf("\nElement is not found\n");}
void display()
{
int front_pos=front,rear_pos=rear;
if(front==-1)
{
printf("\n**********************QUEUE
UNDERFLOW***************\n");
}
else{
printf("\n.............................\n");
if(front_pos>rear_pos){
while(front_pos>rear_pos)
{
printf(" %d ",queue[front_pos]);
front_pos=((front_pos+1))%MAX;
}
}
if(front_pos<=rear_pos){
for(i=front_pos;i<=rear_pos;i++)
printf(" %d ",queue[i]);
}
printf("\n.............................\n");
}
}
```

RESULT:THE REQUIRED OUTPUT IS OBTAINED

OUTPUT

```
1)Insertion
2)Deletion
3)Search
4)Display
5)Exit
1

Enter the element to insert : 10

1)Insertion
2)Deletion
3)Search
4)Display
5)Exit
1

Enter the element to insert : 11

1)Insertion
2)Deletion
3)Search
4)Display
5)Exit
_
```

```
1)Insertion
2)Deletion
3)Search
4)Display
5)Exit
4

.............................
 10  11  12  13  14
.............................

1)Insertion
2)Deletion
3)Search
4)Display
5)Exit
```

```
1)Insertion
2)Deletion
3)Search
4)Display
5)Exit
2

The deleted element is:10

1)Insertion
2)Deletion
3)Search
4)Display
5)Exit
3
```

```
1)Insertion
2)Deletion
3)Search
4)Display
5)Exit
3

Enter the value to search
12

Element is found
1)Insertion
2)Deletion
3)Search
4)Display
5)Exit
5_
```

5. WRITE A C PROGRAM TO IMPLEMENT SINGLY LINKED LIST AS STACK TO PERFORM OPERATIONS SUCH AS PUSH POP AND LINEAR SEARCH

AIM: Program to implement singly linked list operations.

ALGORITHM:

Step 1: Include all the header files which are used in the program.

Step 2: Declare all the functions used in stack implementation.

Step 3: Create a structure with two member variables for storing one data part and one address part(data and *link)

Step 4: Define an pointer variable '*top'.

Step 5: In main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

PUSH()

Step 1: Read an element to be pushed on to stack item

Step 2: check overflow condition of stack before inserting element into stack top=max-1

Step 3: update the top pointer and insert an element into stack Top=top+1 & Stack[top]=item

POP()

Step1: Check underflow condition of stack before deleting element from stack top=-1

Step2: Display deleted element pointed by top (Stack[top])

Step3: Decrement top pointer by 1 (top=top-1)

SEARCH()

Step 1: Read the element to be searched

Step 2: Iterate the loop until top!=NULL

Step 3: If top->data == item then display element found message otherwise not found.

Step 4: Else top=top->link

DISPLAY()

Step 1:Initialise a pointer variable '*ptr' and check the list empty condition then print "Stack is empty" and exit.

Step 2:If it is not empty then assign ptr=top and iterate while loop until (ptr!=NULL)
Step 3:Print ptr->data and assign ptr=ptr->lin

SOURCECODE:

```
#include<stdio.h>
#include<stdlib.h>
void push();
void pop();
void display();
void search();
struct node
{
int data;
struct node *link;
};
struct node *top;
void main ()
{
int ch,n;
printf("\n\t****Stack using linked list*****\n");
while(ch != 5)
{
printf("\n1.Push\n2.Pop\n3.Display\n4.Search\n5.Exit");
printf("\n Enter your choice : ");
scanf("%d",&ch);
switch(ch){
case 1: push(); break;
case 2: pop();break;
case 3: display(); break;
case 4: search(); break;
case 5:exit(0);
default: printf("Invalid choice ");
}
}
}
void push ()
{
int ele;
```

```
struct node *newNode;
newNode= (struct node*)malloc(sizeof(struct node));
if(newNode== NULL)
printf("");
else
{
printf("Enter the element : ");
scanf("%d",&ele);
newNode ->data = ele;
if(top==NULL)
{
newNode -> link = NULL;
}
else
{
newNode ->link = top;
}
top=newNode;
}
}
void pop()
{
int item;
struct node *temp;if (top == NULL)
{
printf("Underflow");
}
else
{
item = top->data;
temp = top;
top = top ->link;
free(temp);
printf("%d is deleted",item);
}
}
void display()
{
int i;
struct node *ptr;
if(top == NULL)
{
printf("Stack is empty \n");
}
else
{
ptr=top;
```

```
printf("Stack elements are :");
while(ptr!=NULL)
{
printf(" \t%d-->",ptr ->data);
ptr = ptr ->link;
}
}
}
void search()
{
int item,i,flag;
if(top==NULL)
printf("\nList is empty");else
{
printf("\nEnter the element to search:");
scanf("%d",&item);
while(top!=NULL)
{
if(top->data==item)
{
flag=1;
break;
}
i++;
}
}
if(flag==1)
printf("\nElement found");
else
{
printf("\nElement not found");
top=top->link;
}
}
```

RESULT:THE REQUIRED OUTPUT IS OBTAINED.

```
        ****Stack using linked list*****

1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice : 1
Enter the element : 78

1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice : 1
Enter the element : 45

1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice :
```

```
 Enter your choice : 1
Enter the element : 45

1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice : 1
Enter the element : 12

1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice : 1
Enter the element : 2

1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice :
```

```
5.Exit
 Enter your choice : 1
Enter the element : 12

1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice : 1
Enter the element : 2

1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice : 3
Stack elements are :    2-->    12-->   45-->   78-->
1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice :
```

```
4.Search
5.Exit
 Enter your choice : 1
Enter the element : 2

1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice : 3
Stack elements are :    2-->    12-->   45-->   78-->
1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice : 2
2 is deleted
1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice : _
```

```
 Enter your choice : 3
Stack elements are :    2-->    12-->   45-->   78-->
1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice : 2
2 is deleted
1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice : 4

Enter the element to search:12

Element found
1.Push
2.Pop
3.Display
4.Search
5.Exit
 Enter your choice : _
```

6. WRITE A C PROGRAM TO IMPLEMENT DOUBLY LINKED LISTOPERATIONS

AIM :Program to implement doubly linked list and its operations

ALGORITHM:

Step 1: Include all the header files which are used in the program.

Step 2: Declare all the functions used in doubly linked list implementation.

Step 3: Create a structure with three member variables for storing one data part and two address parts(data,*prev,*next).

Step 4: In main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

Insert_first(),insert_last() and insert_any() will add node to the list:

Step1: It first checks whether the head is null, then it will insert the node as the head.

Step2: Both head and tail will point to a newly added node.

Step3: Head's previous pointer will point to null and tail's next pointer will point to null.

Step4: If the head is not null, the new node will be inserted at the end of the list such that new node's previous pointer will point to tail.

Step5: The new node will become the new tail. Tail's next pointer will point to null.

delete_first(),delete _last() and _ delete any() will delete the node from list display() will show all the nodes present in the list.

Step1: Define a new node 'current' that will point to the head.

Step:2 Print current.data till current points to null.

Step3: Current will point to the next node in the list in each iteration

SOURCE CODE:

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

```
{
 struct node *prev;
 struct node *next;
 int data;
};
struct node *head;
void insertion_first();
void insertion_last();
void insertion_any();
void deletion_first();
void deletion_last();
void deletion_any();
void search();
void display();
void main ()
{
int choice =0;
clrscr();
 while(choice != 9)
 {
 printf("\nDOUBLY LINKED LIST\n");
 printf("\n1.Insertion in begining \n2.Insertion at last \n3.Insert at anylocation \n4.Delete
from begining \n5.Delete from last \n6.Delete the node from any location \n7.Search
\n8.Display \n9.exit");


 printf("\nEnter your choice : \n");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1: insertion_first();
 break;
 case 2: insertion_last();
 break;
 case 3: insertion_any();  break;
 case 4: deletion_first();
 break;
 case 5: deletion_last();
 break;
 case 6: deletion_any();
 break;
 case 7: search();
 break;
 case 8: display();
 break;
 case 9: exit(0);
 break;
```

```
 default: printf(" Invalid choice!!!");
}
 }
}
void insertion_first()
{
 struct node *ptr;
 int item;
 ptr = (struct node *)malloc(sizeof(struct node));
 if(ptr == NULL)
 {
 printf("\nOVERFLOW");
 }
 else
 {
 printf("\nEnter Item value : \n");
 scanf("%d",&item);
 if(head==NULL)
 {
 ptr->next = NULL;
 ptr->prev=NULL;
 ptr->data=item;
 head=ptr;
 }  else
 {
 ptr->data=item;
 ptr->prev=NULL;
 ptr->next = head;
 head->prev=ptr;
 head=ptr;
 }
 printf("\nNode inserted\n");
}
}
void insertion_last()
{
struct node *ptr,*temp;
 int item;
 ptr = (struct node *) malloc(sizeof(struct node));
 if(ptr == NULL)
 {
 printf("\nOVERFLOW");
 }
 else
 {
 printf("\nEnter value : \n");
 scanf("%d",&item);
```

```
 ptr->data=item;
 if(head == NULL)
 {
 ptr->next = NULL;
 ptr->prev = NULL;
 head = ptr;
 }
 else
 {
 temp = head;
 while(temp->next!=NULL)
 {
 temp = temp->next;  }
 temp->next = ptr;
 ptr ->prev=temp;
 ptr->next = NULL;
 }
 }
 printf("\nnode inserted\n");
}
void insertion_any()
{
 struct node *ptr,*temp;
 int item,loc,i;
 ptr = (struct node *)malloc(sizeof(struct node));
 if(ptr == NULL)
 {
 printf("\n OVERFLOW");
 }
 else
 {
 temp=head;
 printf("Enter the location : \n");
 scanf("%d",&loc);
 for(i=0;i<loc;i++)
 {
 temp = temp->next;
 if(temp == NULL)
 {
 printf("\n There are less than %d elements", loc);
 return;
 }
 }
 printf("Enter value : ");
 scanf("%d",&item);
 ptr->data = item;
 ptr->next = temp->next;
```

```
 ptr -> prev = temp;
 temp->next = ptr;  temp->next->prev=ptr;
 printf("\nnode inserted\n");
 }
}
void deletion_first()
{
 struct node *ptr;
 if(head == NULL)
 {
 printf("\n UNDERFLOW");
 }
 else if(head->next == NULL)
 {
 head = NULL;
 free(head);
 printf("\nnode deleted\n");
 }
 else
 {
 ptr = head;
 head = head -> next;
 head -> prev = NULL;
 free(ptr);
 printf("\nnode deleted\n");
 }
}
void deletion_last()
{
 struct node *ptr;
 if(head == NULL)
 {
 printf("\n UNDERFLOW");
 }
 else if(head->next == NULL)
 {
 head = NULL;
 free(head);  printf("\nnode deleted\n");
 }
 else
 {
 ptr = head;
 if(ptr->next != NULL)
 {
 ptr = ptr -> next;
 }
 ptr -> prev -> next = NULL;
```

```
 free(ptr);
 printf("\nnode deleted\n");
 }
}
void deletion_any()
{
 struct node *ptr, *temp;
 int val;
 printf("\n Enter the data to be deleted : ");
 scanf("%d", &val);
 ptr = head;
 while(ptr -> data != val)
 ptr = ptr -> next;
 if(ptr -> next == NULL)
 {
 printf("\nCan't delete\n");
 }
 else if(ptr -> next -> next == NULL)
 {
 ptr ->next = NULL;
 }
 else
 {
 temp = ptr -> next;
 ptr -> next = temp -> next;
 temp -> next -> prev = ptr;
 free(temp);  printf("\nnode deleted\n");
 }
}
void search()
{
 struct node *ptr;
 int item,i=0,flag;
 ptr = head;
 if(ptr == NULL)
 {
 printf("\nEmpty List\n");
 }
 else
 {
 printf("\nEnter item which you want to search : \n");
 scanf("%d",&item);
 while (ptr!=NULL)
 {
 if(ptr->data == item)
 {
 printf("\nitem found at location %d ",i+1);
```

```
 flag=0;
 break;
 }
 else
 {
 flag=1;
 }
 i++;
 ptr = ptr -> next;
 }
 if(flag==1)
 {
 printf("\nItem not found\n");
 }
 }
} void display()
{
 struct node *ptr;
 printf("\n The values are \n");
 ptr = head;
 while(ptr != NULL)
 {
 printf("%d\n",ptr->data);
 ptr=ptr->next;
} }
```

RESULT: THE REQUIRED OUTPUT IS OBTAINED.

```
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
1

Enter Item value :
23

Node inserted

DOUBLY LINKED LIST

1.Insertion in begining
2.Insertion at last
3.Insert at anylocation
4.Delete from begining
5.Delete from last
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
```

```
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
1

Enter Item value :
12

Node inserted

DOUBLY LINKED LIST

1.Insertion in begining
2.Insertion at last
3.Insert at anylocation
4.Delete from begining
5.Delete from last
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
_
```

```
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
2

Enter value :
45

node inserted

DOUBLY LINKED LIST

1.Insertion in begining
2.Insertion at last
3.Insert at anylocation
4.Delete from begining
5.Delete from last
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
```

```
7.Search
8.Display
9.exit
Enter your choice :
8

 The values are
34
12
23
45

DOUBLY LINKED LIST

1.Insertion in begining
2.Insertion at last
3.Insert at anylocation
4.Delete from begining
5.Delete from last
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
_
```

```
5.Delete from last
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
7

Enter item which you want to search :
12

item found at location 2
DOUBLY LINKED LIST

1.Insertion in begining
2.Insertion at last
3.Insert at anylocation
4.Delete from begining
5.Delete from last
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
_
```

```
8.Display
9.exit
Enter your choice :
7

Enter item which you want to search :
12

item found at location 2
DOUBLY LINKED LIST

1.Insertion in begining
2.Insertion at last
3.Insert at anylocation
4.Delete from begining
5.Delete from last
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
3
Enter the location :
3
Enter value : 90
```

```
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
3
Enter the location :
3
Enter value : 90

node inserted

DOUBLY LINKED LIST

1.Insertion in begining
2.Insertion at last
3.Insert at anylocation
4.Delete from begining
5.Delete from last
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
```

```
3.Insert at anylocation
4.Delete from begining
5.Delete from last
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
4

node deleted

DOUBLY LINKED LIST

1.Insertion in begining
2.Insertion at last
3.Insert at anylocation
4.Delete from begining
5.Delete from last
6.Delete the node from any location
7.Search
8.Display
9.exit
Enter your choice :
_
```

7. WRITE A C PROGRAM TO CREATE A BINARY SEARCH TREE AND TRAVERSE IT.

AIM:Program to implement binary search tree

ALGORITHM:

Step 1: Include all the header files which are used in the program.

Step 2: Declare all the functions used in tree implementation.

Step 3: Create a structure tnode with three member variables (data,*lchild,*rchild)

Step 4: Inside the main function define the functions to implement the tree operations.

Step 5: Perform inorder,postorder,preorder traversals by calling their corresponding functions(Inorder(),Postorder(),Preorder()).

Tinsert() will insert the new value into a binary search tree:

Step1: It checks whether root is null, which means tree is empty. New node will become root node of tree.

Step2: If tree is not empty, it will compare value of new node with root node. If value of new node is greater than root, new node will be inserted to right subtree. Else, it will be inserted in left subtree.

Tdelete() will delete a particular node from the tree:

Step1: If value of node to be deleted is less than root node, search node in left subtree. Else, search in right subtree.

Step2: If node is found and it has no children, then set the node to null.

Step3: If node has one child then, child node will take position of node.

Step4: If node has two children then, find a minimum value node from its right subtree. This minimum value node will replace the current node.

Tsearch() will search a particular node from the tree.

SOURCECODE:

```
#include<stdio.h>
#include<conio.h>
struct tnode
{
struct tnode *lchild;
int data;
struct tnode *rchild;
};
typedef struct tnode tnode;
tnode *getnode();
main()
{
int a[100],i,n,item;
tnode *root;
clrscr();
root=NULL;
printf("Enter the no of elements:");
scanf("%d",&n);
printf("\nEnter the elements:");
for(i=0;i<n;i++)
 scanf("%d",&a[i]);
for(i=0;i<n;i++)
 Tinsert(&root,a[i]);
printf("\nBinary tree:\n");
Tdisplay(root,1);
printf("\nInorder Traversal\n");
Inorder(root);
printf("\nPreorder Traversal\n");
Preorder(root);
printf("\nPostorder Traversal\n");
Postorder(root);
printf("\nEnter the element to delete:");scanf("%d",&item);
Tdelete(&root,item);
printf("\nBinary tree after deletion\n");
Tdisplay(root,1);
printf("\nEnter the element to search:");
scanf("%d",&item);
Tsearch(item);
getch();
}
Tinsert(tnode **rt,int item)
{
 tnode *current=(*rt),*temp;
 if((*rt)==NULL)
```

```
{
(*rt)=getnode();
(*rt)->data=item;
(*rt)->lchild=NULL;
(*rt)->rchild=NULL;
return;
}
while(current!=NULL)
{
if(item<current->data)
{
if(current->lchild!=NULL)
current=current->lchild;
else
{
temp=getnode();
current->lchild=temp;
temp->data=item;
temp->rchild=NULL;
temp->lchild=NULL;
return;
}
}
else {
if(item>current->data)
{
if(current->rchild!=NULL)
current=current->rchild;
else
{
temp=getnode();
current->rchild=temp;
temp->data=item;
temp->rchild=NULL;
temp->lchild=NULL;
return;
}
}
else
{
printf("\nWrong data");
exit(0);
}
}
}
}
Inorder(tnode *rt)
```

```
{
if(rt!=NULL)
{
Inorder(rt->lchild);
printf("\t%d\t",rt->data);
Inorder(rt->rchild);
}
else
 return;
}
Preorder(tnode *rt)
{
if(rt!=NULL){
printf("\t%d\t",rt->data);
Preorder(rt->lchild);
Preorder(rt->rchild);
}
else
 return;
}
Postorder(tnode *rt)
{
if(rt!=NULL)
{
 Postorder(rt->lchild);
 Postorder(rt->rchild);
 printf("\t%d\t",rt->data);
}
else
 return;
}
Tdisplay(tnode *rt,int level)
{
int i;
if((rt)!=NULL)
{
 Tdisplay((rt)->rchild,level+1);
 printf("\n");
 for(i=0;i<level;i++)
 printf(" ");
 printf("%d",(rt)->data);
 Tdisplay((rt)->lchild,level+1);
 }
}
Tdelete(tnode **rt,int item)
{
tnode *current;
```

```
if(*rt==NULL)
 { printf("\nError");
 getch();
 return;
 }
if(item<(*rt)->data)
 Tdelete(&((*rt)->lchild),item);
else
{
 if(item>(*rt)->data)
 Tdelete(&((*rt)->rchild),item);
 else
 {
 current=(*rt);
 if(current->rchild==NULL)
 {
 (*rt)=(*rt)->lchild;
 free(current);
 }
 else
 {
 if(current->lchild==NULL)
 {
 (*rt)=(*rt)->rchild;
 free(current);
 }
 else
 {
 current=(*rt)->rchild;
 while(current->lchild!=NULL)
 current=current->lchild;
 current->lchild=(*rt)->lchild;
 current=(*rt);
 (*rt)=(*rt)->rchild;
 free(current);
 }
 }
 } }
}
Tsearch(int item)
{
tnode *temp,*root;
if(root==NULL)
{
 printf("\nTree is empty");
 return;
}
```

```
if(item==root->data)
 {
 printf("\nElement found at root");
 return;
 }
else if(item<root->data)
{
 printf("\nElement found at left sub tree");
 return;
 }
else
{
 printf("\nElement found at right sub tree");
 return;
 }
}
tnode *getnode()
{
tnode *p;
p=(tnode *)malloc(sizeof(tnode));
return(p);
}
freenode(tnode *p)
{
free(p);
}
```

RESULT: THE REQUIRED OUTPUT IS OBTAINED

OUTPUT

```
Enter the no of elements:6

Enter the elements:4 1 12 34 78 16

Binary tree:

    78
   34
    16
  12
 4
  1
Inorder Traversal
        1               4               12              16              34
        78
Preorder Traversal
        4               1               12              34              16
        78
Postorder Traversal
        1               16              78              34              12
        4
Enter the element to delete:
```

```
    78
   34
    16
  12
 4
  1
Inorder Traversal
        1               4               12              16              34
        78
Preorder Traversal
        4               1               12              34              16
        78
Postorder Traversal
        1               16              78              34              12
        4
Enter the element to delete:34

Binary tree after deletion

   78
    16
  12
 4
  1
Enter the element to search:
```

```
    16
  12
 4
  1
Inorder Traversal
        1               4               12              16              34
        78
Preorder Traversal
        4               1               12              34              16
        78
Postorder Traversal
        1               16              78              34              12
        4
Enter the element to delete:34

Binary tree after deletion

   78
    16
  12
 4
  1
Enter the element to search:12

Element found at left sub tree_
```

8. WRITE A C PROGRAM TO IMPLEMENT SET DATASTRUCTURE AND SET OPERATIONS UNION,INTERSECTION,DIFFERENCE.

AIM: Program to implement set datastructure and is operations.

ALGORITHM:

Step 1: Include all the header files which are used in the program.

Step 2: Declare all the functions used in set implementation.

Step 3: Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on set data structure.

Step 4: Iterate do..while loop until when user enter ‘o’ means that it terminates the program.

Input

Step1: Declare two sets as arrays a[],b[].

Step2: Read the size and the elements in the two sets.

Step3: Print the two sets a and b.

Union

Step1: Initialize union U as empty.

Step2: Find smaller of m and n and sort the smaller array.

Step3: Copy the smaller array to U.

Step4: For every element x of larger array, do following

Step5: Binary Search x in smaller array. If x is not present, then copy it to U.

Step6: Return U.

Intersection

Step1: Initialize intersection I as empty.

Step2: Find smaller of m and n and sort the smaller array.

Step3: For every element x of larger array, do following

Step4: Binary Search x in smaller array. If x is present, then copy it to I.

Step5: Return I.

Difference

Step1: Declare c[10] to store the result.

Step2: Declare 'i' and initialise with 1 and incrementing by 1 until i<=9.

Step3: Check if it is the elements present in a also present in b.

Step4: If it is present copy it to c otherwise c=0. Step5: Return c.

SOURCECODE:

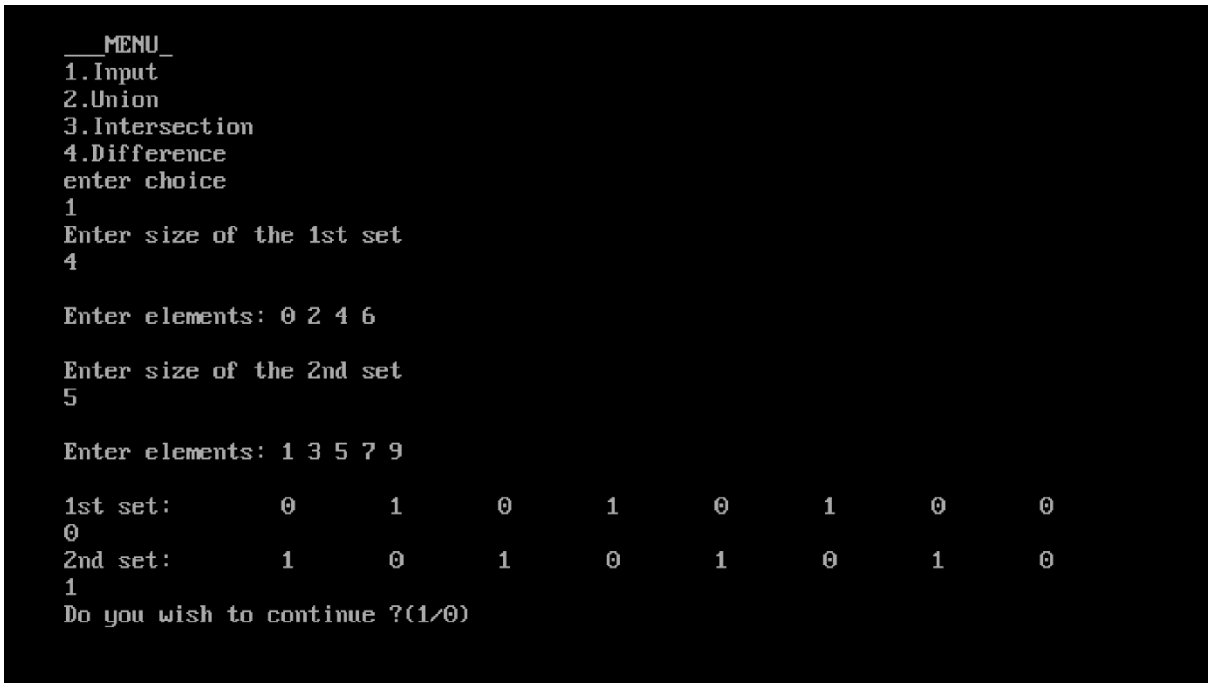
```
//Universal Set is {1,2,3,4,5,6,7,8,9}

#include <stdio.h>
void input();
void output();
void setunion();
void intersection();
void difference();
int a[]={0,0,0,0,0,0,0,0,0},b[]={0,0,0,0,0,0,0,0,0};
int c[]={0,0,0,0,0,0,0,0,0};
int main()
 {
  int ch,wish;
  clrscr();
  do
   {
  printf("\n___MENU_\n");
  printf("1.Input\n2.Union\n3.Intersection\n4.Difference\n");
  printf("enter choice\n");
  scanf("%d",&ch);
  switch(ch)
   {
     case 1:input();
              break;
```

```
    case 2:setunion();
           break;
    case 3:intersection();
           break;
    case 4:difference();
           break;
  }
  printf("\nDo you wish to continue ?(1/0)\n");
 scanf("%d",&wish);
  }while(wish==1);
 }
void input()
{
 int n,x,i;
 printf("Enter size of the 1st set\n");
 scanf("%d",&n);
 printf("\nEnter elements:\t");
 for(i=1;i<=n;i++)
 {
    scanf("%d",&x);
    a[x]=1;
 }
 printf("\nEnter size of the 2nd set\n");
 scanf("%d",&n);
 printf("\nEnter elements:\t");
 for(i=1;i<=n;i++)
 {
    scanf("%d",&x);
    b[x]=1;
 }
 printf("\n1st set:\t");
 for(i=1;i<=9;i++)
  {
   printf("%d\t",a[i]);
  }
  printf("\n2nd set:\t");
  for(i=1;i<=9;i++)
  {
  printf("%d\t",b[i]);
  }
}
void output(int c[])
{
   int i;
   printf("\n Set is:\t");
   for(i=1;i<=9;i++)
   {
```

```
      if (c[i]!=0)
      printf("%d\t",i);
    }

}
void setunion()
{
  int i,c[10];
  for(i=1;i<=9;i++)
    {
    if(a[i]!=b[i])
        c[i]=1;
    else
      c[i]=a[i];
    }
  for(i=1;i<=9;i++)
   {
   printf("%d",c[i]);
   }
   output(c);
}
void intersection()
{
  int i,c[10];
  for(i=1;i<=9;i++)
    {
    if (a[i]==b[i])
            c[i]=a[i];
    else
            c[i]=0;
    }
  for(i=1;i<=9;i++)
   {
   printf("%d",c[i]);
   }
   output(c);
}
void difference()
{
  int i,c[10];
  for(i=1;i<=9;i++)
    {
    if (a[i]==1 && b[i]==0)
            c[i]=1;
    else
      c[i]=0;
    }
```

```
  for(i=1;i<=9;i++)
   {
   printf("%d",c[i]);
   }


}
```
RESULT: THE REQUIRED OUTPUT IS OBTAINED.

OUTPUT


___MENU_
1.Input
2.Union
3.Intersection
4.Difference
enter choice
1
Enter size of the 1st set
4
Enter elements: 0 2 4 6
Enter size of the 2nd set
5
Enter elements: 1 3 5 7 9
1st set: 0 1 0 1 0 1 0 0
0
2nd set: 1 0 1 0 1 0 1 0
1
Do you wish to continue ?(1/0)

```
Enter size of the 2nd set
5

Enter elements: 1 3 5 7 9

1st set:        0       1       0       1       0       1       0       0
0
2nd set:        1       0       1       0       1       0       1       0
1
Do you wish to continue ?(1/0)
1

___MENU_
1.Input
2.Union
3.Intersection
4.Difference
enter choice
2
111111101
 Set is:        1       2       3       4       5       6       7       9

Do you wish to continue ?(1/0)
_
```

```
___MENU_
1.Input
2.Union
3.Intersection
4.Difference
enter choice
2
111111101
 Set is:        1       2       3       4       5       6       7       9

Do you wish to continue ?(1/0)
1

___MENU_
1.Input
2.Union
3.Intersection
4.Difference
enter choice
3
000000000
 Set is:
Do you wish to continue ?(1/0)
```

```
1

___MENU_
1.Input
2.Union
3.Intersection
4.Difference
enter choice
3
000000000
 Set is:
Do you wish to continue ?(1/0)
1

___MENU_
1.Input
2.Union
3.Intersection
4.Difference
enter choice
4
010101000
 Set is:        2       4       6
Do you wish to continue ?(1/0)
_
```

9. WRITE A C PROGRAM TO IMPLEMNT DISJOINT SET AND ITS OPERATIONS.

AIM : Program to implement disjoint set.

ALGORITHM:

Step 1: Include all the header files which are used in the program.

Step 2: Create a structure DisjSet with three member functions (parent,rank,n) to store parent array,rank array,number of sets.Create a variable for the structure ‘dis’.

Step 3: Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on disjoint set.

makeSet()

Step 1: Declare ‘i’ and ‘x’.

Step 2: Initialise i with ‘0’ and incrementing by ‘1’(i++) until i i<dis.n and read x.

Step 3: Assign dis.parent[i]=i and dis.rank[i]=0.

displaySet()

Step 1: Declare ‘i’

Step 2: Iterate two loops for printing parent array and rank array initialise i with

‘0’ and incrementing by ‘1’(i++) until i<dis.n.

Step 3: Print two arrays(dis.parent[i] and dis.rank[i]).

find(intx)

Step 1: Check the ‘x’ is not the parent of itself it means that find the

representative(dis.parent[x]!=x) Step 2: If it is true then dis.parent[x]=find(dis.parent[x]), so we recursively call

find() on its parent and move I's node directly under the representative of

this set.

Step 3: Return dis.parent[x].

union(int x,int y)

Step 1: Find current sets of x and y(int xset=find(x) and int yset=find(y))

Step 2: Check if it is in the same set,if it is true then exit

Step 3: Put smaller ranked item under bigger ranked item if ranks are different and vice versa. Step 4: If ranks are same, then increment rank.

SOURCECODE

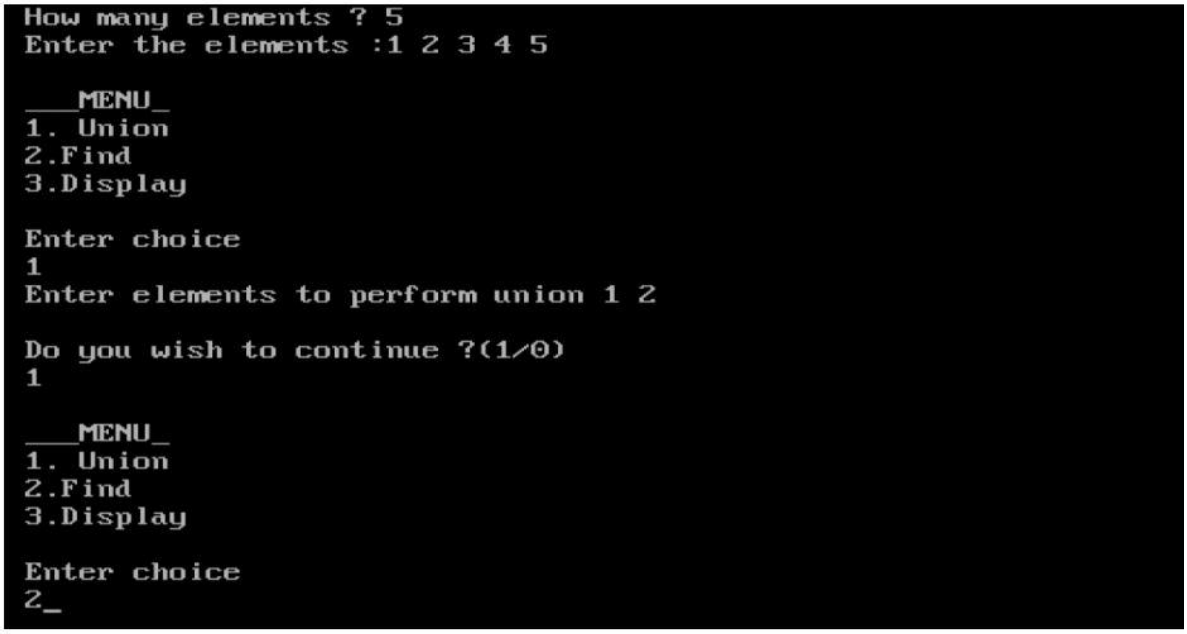
```
#include <stdio.h>
struct DisjSet
{
 int parent[10];
 int rank[10]; //rank[i] is the height of the tree representing the set
 int n;
}dis;
 // Creates n single item sets
 void makeSet()
 {
int i,x;
for ( i = 0; i < dis.n; i++) {
 scanf("%d",&x);
 dis.parent[i] = i;
 dis.rank[i]=0;
 }
 }
 //Displays Disjoint set
 void displaySet()
 {
 int i;
 printf("\nParent Array\n"); for ( i = 0; i < dis.n; i++) {
 printf("%d ",dis.parent[i]); }
 printf("\nRank Array\n");
 for (i = 0; i < dis.n; i++)
 {
 printf("%d ",dis.rank[i]);
 }
 printf("\n");
 }
 // Finds set of given item x
 int find(int x)
 {
 // Finds the representative of the set
 // that x is an element of
 if (dis.parent[x] != x) {

 // if x is not the parent of itself
 // Then x is not the representative of
 // his set,
 dis.parent[x] = find(dis.parent[x]);

 // so we recursively call Find on its parent
 // and move i's node directly under the
 // representative of this set
 }
 return dis.parent[x];
 }
```

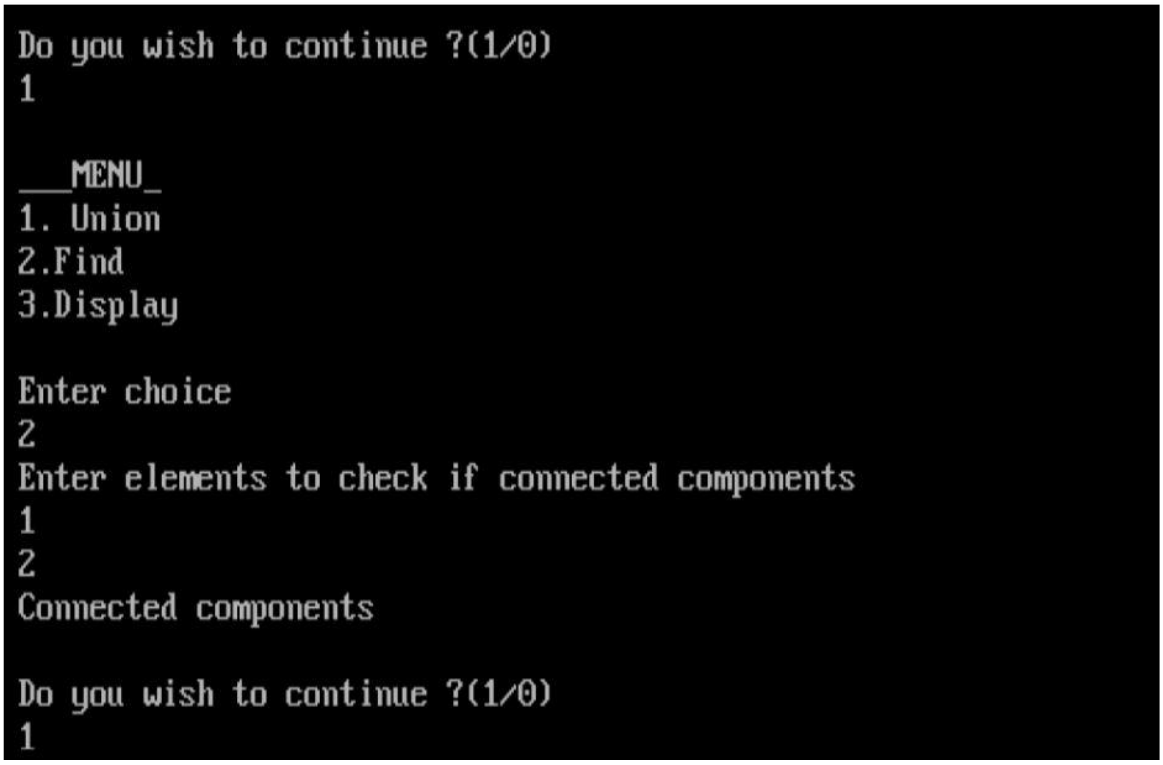
```
 // Do union of two sets represented
 // by x and y.
 void Union(int x, int y)
 {
 // Find current sets of x and y
 int xset = find(x);
 int yset = find(y);

 // If they are already in same set  if (xset == yset)
 return;
 // Put smaller ranked item under
 // bigger ranked item if ranks are
 // different
 if (dis.rank[xset] < dis.rank[yset]) {
 dis.parent[xset] = yset;
 dis.rank[xset]=-1;
 }
 else if (dis.rank[xset] > dis.rank[yset]) {
 dis.parent[yset] = xset;
 dis.rank[yset]=-1;
 }
 // If ranks are same, then increment
 // rank.
 else {
 dis.parent[yset] = xset;
 dis.rank[xset] = dis.rank[xset] + 1;
 dis.rank[yset]=-1;
 }
 }

int main()
{
 int n,x,y,ch,wish;
 clrscr();
 printf("How many elements ?");
 scanf("%d",&dis.n);
 printf("Enter the elements :");
 makeSet();
do
 {
 printf("\n___MENU_\n");
 printf("1. Union \n2.Find\n3.Display\n");
 printf("\nEnter choice\n");
 scanf("%d",&ch);
 switch(ch) {
 case 1: printf("Enter elements to perform union");
 scanf("%d %d",&x,&y);
 Union(x, y);
break;
 case 2: printf("Enter elements to check if connected components");
```

```
 scanf("%d %d",&x,&y);
 if (find(x) == find(y))
 printf("Connected components\n") ;
 else
printf("Not connected components \n") ;
 break;
 case 3: displaySet();
 break;
 }
 printf("\nDo you wish to contntinue ?(1/0)\n");
 scanf("%d",&wish);
 }while(wish==1);
return 0;
}
```

RESULT: THE REQUIRED OUTPUT IS OBTAINED.

OUTPUT


How many elements ? 5
Enter the elements :1 2 3 4 5
___MENU_
1. Union
2.Find
3.Display
Enter choice
1
Enter elements to perform union 1 2
Do you wish to continue ?(1/0)
1
___MENU_
1. Union
2.Find
3.Display
Enter choice
2_


Do you wish to continue ?(1/0)
1
___MENU_
1. Union
2.Find
3.Display
Enter choice
2
Enter elements to check if connected components
1
2
Connected components
Do you wish to continue ?(1/0)
1

```
___MENU_
1. Union
2.Find
3.Display

Enter choice
3

Parent Array
0 1 1 3 4
Rank Array
0 1 -1 0 0

Do you wish to continue ?(1/0)
0_
```

11. WRITE A C PROGRAM TO IMPLEMENT RED BLACK TREE.

AIM: Program to implement red black tree.

ALGORITHM:

Step 1: Include all the header files which are used in the program.

Step 2: Create a structure rbtNode with essential member

variables(data,color,*left,*right,*parent).Create a variable to access structure *root=NULL

Step 3: Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on red black tree implementation.

Insert()

Step 1: Check whether tree is Empty.

Step 2: If tree is Empty then insert the newNode as Root node with color Black and exit from the operation.

Step 3: If tree is not Empty then insert the newNode as a leaf node with Red color.

Step 4: If the parent of newNode is Black then exit from the operation.

Step 5: If the parent of newNode is Red then check the color of parent node's sibling of newNode.

Step 6: If it is Black or NULL node then make a suitable Rotation and Recolor it.

Step 7: If it is Red colored node then perform Recolor and Recheck it. Repeat the same until tree becomes Red Black Tree.

leftRotate()

Step 1: If the parent of newly inserting node is red node then perform this function by calling.

Step 1: If the parent of newly inserting node is black or null node then perform this function by calling.

inorder()

Step 1:To display the tree nodes as left,root,right

color()

Step 1:When adjacent red color appears at the time of inserting then recolor it using this function.

SOURCECODE:

```
#include<stdio.h>
#include<conio.h>
struct rbtNode

{
   int data;
   char color;
   struct rbtNode *left, *right, *parent;
};
struct rbtNode *root=NULL;
void leftRotate(struct rbtNode *x)
{
   struct rbtNode *y;
   y= x->right;
   x->right = y->left;


   if (y->left != NULL)
            y->left->parent = x;

   y->parent = x->parent;
   if (x->parent == NULL)// x is root
            root = y;

   else if((x->parent->left != NULL) && (x->data==x->parent->left->data))
            x->parent->left = y;

   else
            x->parent->right = y;

   y->left = x;
   x->parent = y;
```

```
   return;
}
void rightRotate(struct rbtNode *y)

{

   struct rbtNode *x   ;
   x= y->left;

   y->left = x->right;

   if (x->right != NULL)

         x->right->parent = y;

   x->parent =y->parent;

   if (y->parent == NULL)
         root= x;

   else if((y->parent->left!=NULL)&&(y->data==y->parent->left->data))

         y->parent->left = x;

   else
         y->parent->right = x;

   x->right = y;

   y->parent = x;
 return;
}
void color(struct rbtNode *z)
{
   struct rbtNode *y=NULL;
   while((z->parent!=NULL)&&(z->parent->color=='R'))
   {
         if((z->parent->parent->left!=NULL)&&(z->parent->data==z->parent->parent->left-
>data))
         {
            if(z->parent->parent->right!=NULL)
                  y=z->parent->parent->right;
            if((y!=NULL)&&(y->color=='R'))
            {
                  z->parent->color='B';
                  y->color='B';
                  z->parent->parent->color = 'R';
```

```
                if(z->parent->parent!=NULL)
                   z = z->parent->parent;

            }
            else
            {
                if((z->parent->right!=NULL)&&(z->data==z->parent->right->data))
                {
                   z=z->parent;
                   leftRotate(z);
                }
                z->parent->color='B';
                        z->parent->parent->color='R';
                rightRotate(z->parent->parent) ;
            }
        }
        else
        {
                if(z->parent->parent->left!=NULL)
                y=z->parent->parent->left;
            if((y!=NULL)&&(y->color=='R'))
            {
                z->parent->color='B';
                y->color='B';
                z->parent->parent->color='R';
                if(z->parent->parent!=NULL)
                   z=z->parent->parent;
            }
            else
            {
                        if((z->parent->left!=NULL)&&(z->data==z->parent->left->data))
                {
                   z=z->parent;
                   rightRotate(z);
                }
                z->parent->color='B';
                z->parent->parent->color='R';
                leftRotate(z->parent->parent);
            }

        }
    }
    root->color='B';
}
void insert(int val)
{
    struct rbtNode *x,*y,*z;
```

```
   z=(struct rbtNode *)malloc(sizeof(struct rbtNode));
   z->data=val;
   z->left=NULL;
   z->right=NULL;
   z->color='R';

   x=root;

   if(root==NULL)// root node or not
   {
         root=z;
         root->color='B';
         return;
   }

   while(x!=NULL)
   {
         y=x;
         if(z->data<x->data)
            x=x->left;
         else
            x=x->right;
   }
   z->parent=y;
   if(y==NULL)
            root=z;
   else if(z->data<y->data)
            y->left=z;
   else
            y->right=z;

   color(z);

}
void inorder(struct rbtNode *root)

{
   struct rbtNode *temp=root;
   if (temp != NULL)
   {
         inorder(temp->left);

            printf("%d-%c   ", temp->data,temp->color);

         inorder(temp->right);
   }
   return;
```

```
}
void displayTree(struct rbtNnode *root,int level)
{
  int i;
  struct rbtNode *temp=root;

  if(temp!=NULL)
  {
        displayTree(temp->right,level+1);
        printf("\n\n");
        for(i=0;i<level;i++)
              printf("    ");
        printf("%d%c",temp->data,temp->color);
        displayTree(temp->left,level+1);
  }
}
void main()
{
   int ch,val;
   clrscr();
   printf("*******RED BLACK TREE INSERTION PROGRAM********")  ;
   while(1)
   {
        printf("\n1.Insert\n2.Display\n3.Exit\nEnter choice   :   ");
        scanf("%d",&ch);
        switch(ch)
        {
           case 1 :
           printf("Enter element : ");
           scanf("%d",&val);
           insert(val);
           break;
           case 2:
           displayTree(root,1);
           printf("\n\nINORDER TRAVERSAL :     ");
           inorder(root);break;
           case 3:exit(0);
        }
   }

}
```

RESULT: THE REQUIRED OUTPUT IS OBTAINED.

OUTPUT

```
*******RED BLACK TREE INSERTION PROGRAM********
1.Insert
2.Display
3.Exit
Enter choice   :   1
Enter element : 90

1.Insert
2.Display
3.Exit
Enter choice   :   1
Enter element : 45

1.Insert
2.Display
3.Exit
Enter choice   :   1
Enter element : 67

1.Insert
2.Display
3.Exit
Enter choice   :   1
Enter element : 34
```

```
1.Insert
2.Display
3.Exit
Enter choice   :   1
Enter element : 34

1.Insert
2.Display
3.Exit
Enter choice   :   2


        90B

    67B

        45B

            34R

INORDER TRAVERSAL :     34-R   45-B   67-B   90-B
1.Insert
2.Display
3.Exit
Enter choice   :
```

12. WRITE A C PROGRAM TO IMPLEMENT BREADTH FIRST SEARCH.

AIM: PROGRAM TO IMPLEMENT BREADTH FIRST SEARCH.

ALGORITHM:

Step 1: Include all the header files which are used in the program.

Step 2: Declare a[20][20] for storing the vertices to perform traversal

Step 3: Declare q[20] for inserting elements in queue and declare visited[20] for printing the result.

Step 4: Start by putting any one of the graph's vertices at the back of a queue.

Step 5: Take the front item of the queue and add it to the visited list.

Step 6: Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.

Step 7: Keep repeating steps 5 and 6 until the queue is empty.

Step 8: Return visited[20] array

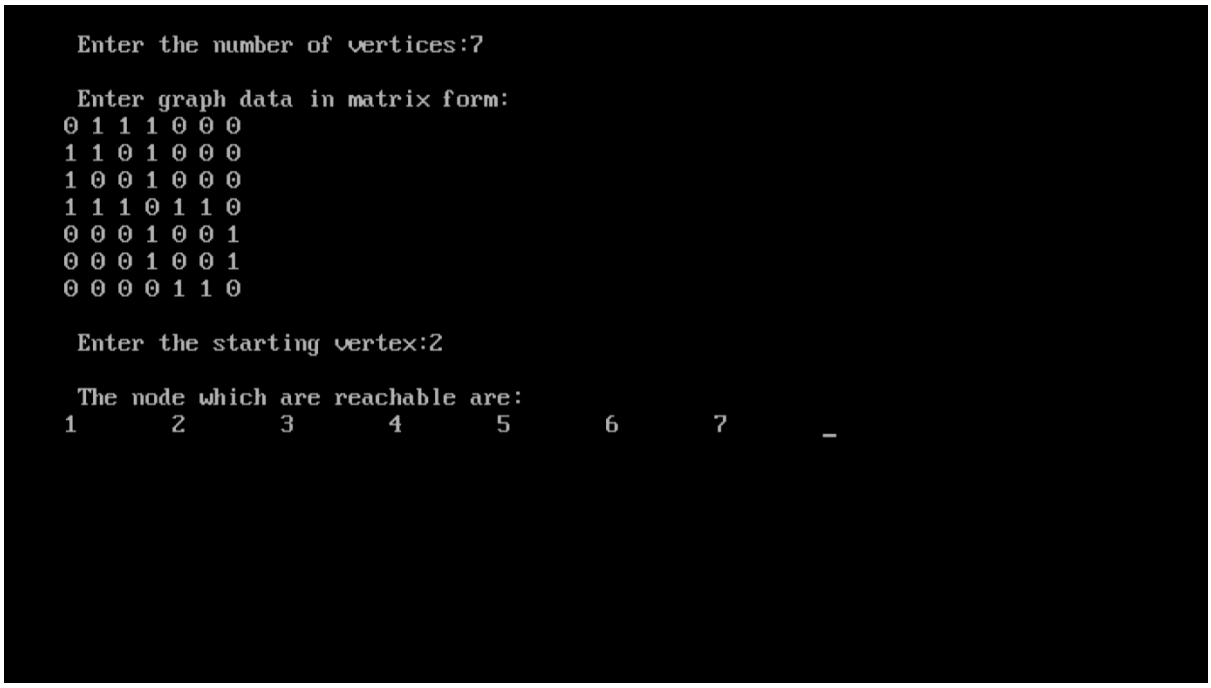
SOURCECODE:

```
#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++]);
```

```
        }
}
void 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");
        getch();
       }
```

RESULT: THE REQUIRED OUTPUT IS OBTAINED.

OUTPUT


Enter the number of vertices:7
Enter graph data in matrix form:
0 1 1 1 0 0 0
1 1 0 1 0 0 0
1 0 0 1 0 0 0
1 1 1 0 1 1 0
0 0 0 1 0 0 1
0 0 0 1 0 0 1
0 0 0 0 1 1 0
Enter the starting vertex:2
The node which are reachable are:
1 2 3 4 5 6 7

13. WRITE A C PROGRAM TO IMPLEMENT DEPTH FIRST SEARCH.

AIM:Program to implement depth first search

ALGORITHM:

Step 1: Include all the header files which are used in the program.

Step 2: Declare a[20][20] for storing the vertices to perform traversal

Step 3: Declare stk[20] for inserting elements in stack and declare reach[20] for printing the result.

Step 4: Start by putting any one of the graph's vertices on top of a stack.

Step 5: Take the top item of the stack and add it to the visited list.

Step 6: Create a list of that vertex's adjacent nodes. Add the ones which aren't in the reach list to the top of the stack.

Step 7: Keep repeating steps 5 and 6 until the stack is empty.

SOURCECODE:

```
#include<stdio.h>
int a[20][20],reach[20],n;
void 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);
     printf("\n%d",v);
  }
 }
void 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");
            getch();
}
```

RESULT: THE REQUIRED OUTPUT IS OBTAINED.

OUTPUT

```
 Enter number of vertices:6

 Enter the adjacency matrix:
0 1 0 1 0 1
1 0 1 0 1 0
0 1 0 1 1 0
1 0 1 0 0 1
0 1 1 0 0 0
1 0 0 1 0 0

 1->2
 2->3
 3->4
 4->6
4
3
 3->5
3
2
1

 Graph is connected_
```

14. WRITE A C PROGRAM TO IMPLEMENT PRIM'S ALGORITHM.

AIM: Program to implement prims's algorithm.

ALGORITHM:

Step 1: Include all the header files which are used in the program.

Step 2: Declare cost[10][10] for storing the weighted adjacency matrix and assign mincost=0.

Step 3: Declare visited[10] array for storing visited vertices and assign all vertices weight as infinity.

Step 4: Initialize the minimum spanning tree with a vertex chosen at random.

Step 5: Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree.

Step 6: Keep repeating step 5 until we get a minimum spanning tre

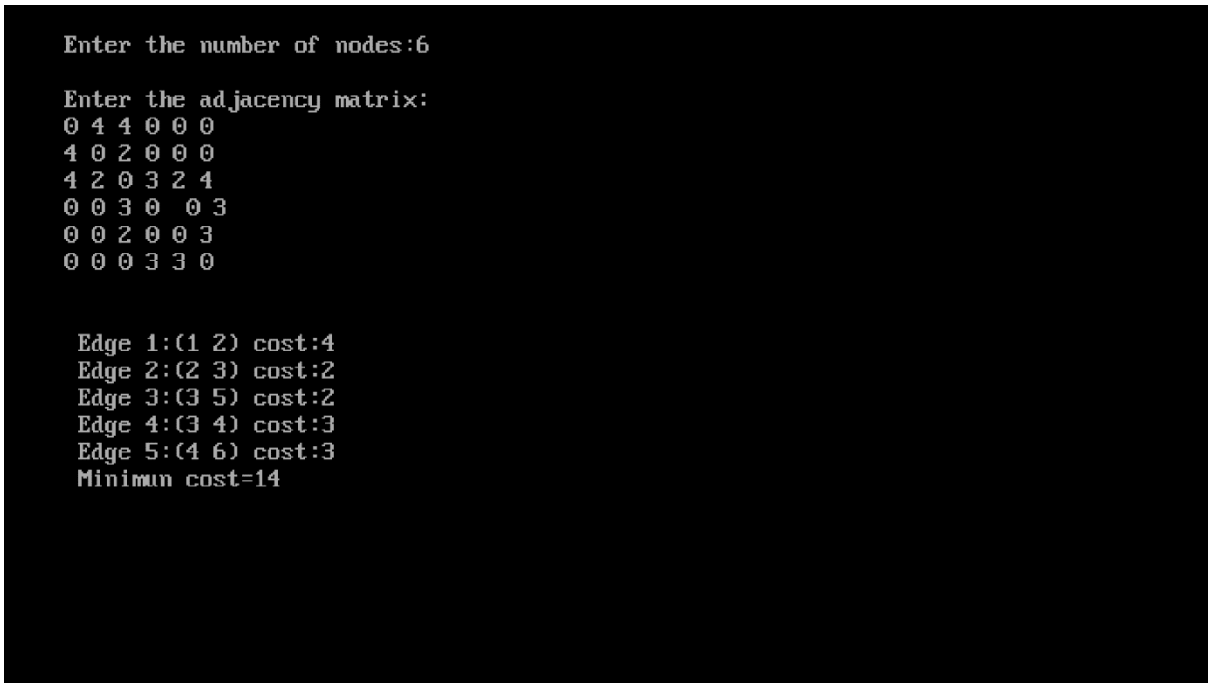
SOURCECODE:

```
#include<stdio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{
    printf("\nEnter the number of nodes:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
    {
        scanf("%d",&cost[i][j]);
```

```
            if(cost[i][j]==0)
                    cost[i][j]=999;
        }
        visited[1]=1;
        printf("\n");
        while(ne < n)
        {
            for(i=1,min=999;i<=n;i++)
            for(j=1;j<=n;j++)
            if(cost[i][j]< min)
            if(visited[i]!=0)
            {
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;
            }
            if(visited[u]==0 || visited[v]==0)
            {
                    printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                    mincost+=min;
                    visited[b]=1;
            }
            cost[a][b]=cost[b][a]=999;
        }
        printf("\n Minimun cost=%d",mincost);
        getch();
}
```

RESULT: THE REQUIRED OUTPUT IS OBTAINE

OUTPUT


Enter the number of nodes:6
Enter the adjacency matrix:
0 4 4 0 0 0
4 0 2 0 0 0
4 2 0 3 2 4
0 0 3 0 0 3
0 0 2 0 0 3
0 0 0 3 3 0
Edge 1:(1 2) cost:4
Edge 2:(2 3) cost:2
Edge 3:(3 5) cost:2
Edge 4:(3 4) cost:3
Edge 5:(4 6) cost:3
Minimun cost=14

15. WRITE A C PROGRAM TO IMPLEMENT KRUSKAL'S ALGORITHM.

AIM: Program to implement kruskal's aagorithm.

ALGORITHM:

Step 1: Include all the header files which are used in the program.

Step 2: Declare cost[9][9] for storing adjacency weighted matrix and assign mincost=0

Step 3: Sort all the edges from low weight to high.

Step 4: Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.

Step 5: Keep adding edges until we reach all vertices.

SOURCE CODE:

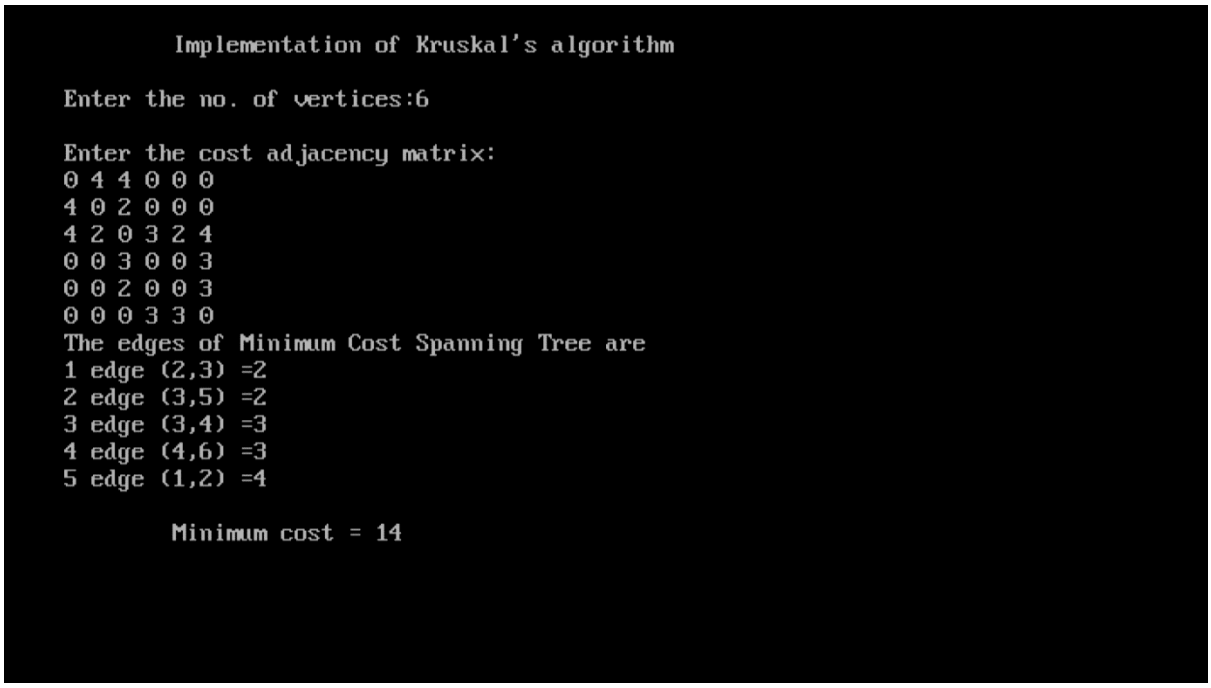
```
#include<stdio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
```

```
printf("\n\tImplementation of Kruskal's algorithm\n");
printf("\nEnter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter the cost adjacency matrix:\n");
for(i=1;i<=n;i++)
{
        for(j=1;j<=n;j++)
        {
                scanf("%d",&cost[i][j]);
                if(cost[i][j]==0)
                        cost[i][j]=999;
        }
}
printf("The edges of Minimum Cost Spanning Tree are\n");
while(ne < n)
{
  for(i=1,min=999;i<=n;i++)
        {
          for(j=1;j <= n;j++)
                {
                   if(cost[i][j] < min)
                        {
                          min=cost[i][j];
                          a=u=i;
                          b=v=j;
                        }
                }
        }
        u=find(u);
```

```
                v=find(v);
                if(uni(u,v))
                {
                  printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
                 mincost +=min;
                }
          cost[a][b]=cost[b][a]=999;
        }
   printf("\n\tMinimum cost = %d\n",mincost);
   getch();
}
int find(int i)
{
        while(parent[i])
        i=parent[i];
        return i;
}
int uni(int i,int j)
{
        if(i!=j)
        {
                parent[j]=i;
                return 1;
        }
        return 0;
}
```

RESULT: THE REQUIRED RESULT IS OBTAINED.

OUTPUT


Implementation of Kruskal's algorithm
Enter the no. of vertices:6
Enter the cost adjacency matrix:
0 4 4 0 0 0
4 0 2 0 0 0
4 2 0 3 2 4
0 0 3 0 0 3
0 0 2 0 0 3
0 0 0 3 3 0
The edges of Minimum Cost Spanning Tree are
1 edge (2,3) =2
2 edge (3,5) =2
3 edge (3,4) =3
4 edge (4,6) =3
5 edge (1,2) =4
Minimum cost = 14

16. WRITE A C PROGRAM TO IMPLEMENT DIJIKSTRA'S ALGORITHM.

AIM: To perform dijikstras's algorithm.

ALGORITHM:

Step 1: Include all the header files which are used in the program.

Step 2: Define INFINITY as 999 and MAX 10

Step 3: Read the number of vertices and adjacency weighted matrix.

Step 4: Read the starting node,u.

Step 5: Initialize visited array with false which shows that currently, the tree is empty.

Step 6: Initialize cost array with infinity which shows that it is impossible to reach any node from the start node via a valid path in the tree.

Step 7: The cost to reach the start node will always be zero, hence cost[start]=0.
Step 8: Now at every iteration we choose a node to add in the tree, hence we need n iterations to add n nodes in the tree:

8(a): Choose a node that has a minimum cost and is also currently nonvisited i.e., not present in the tree.

8(b): Update the cost of non-visited nodes which are adjacent to the newly added node with the minimum of the previous and new path.

8(c): Mark the node as visited.

SOURCECODE:

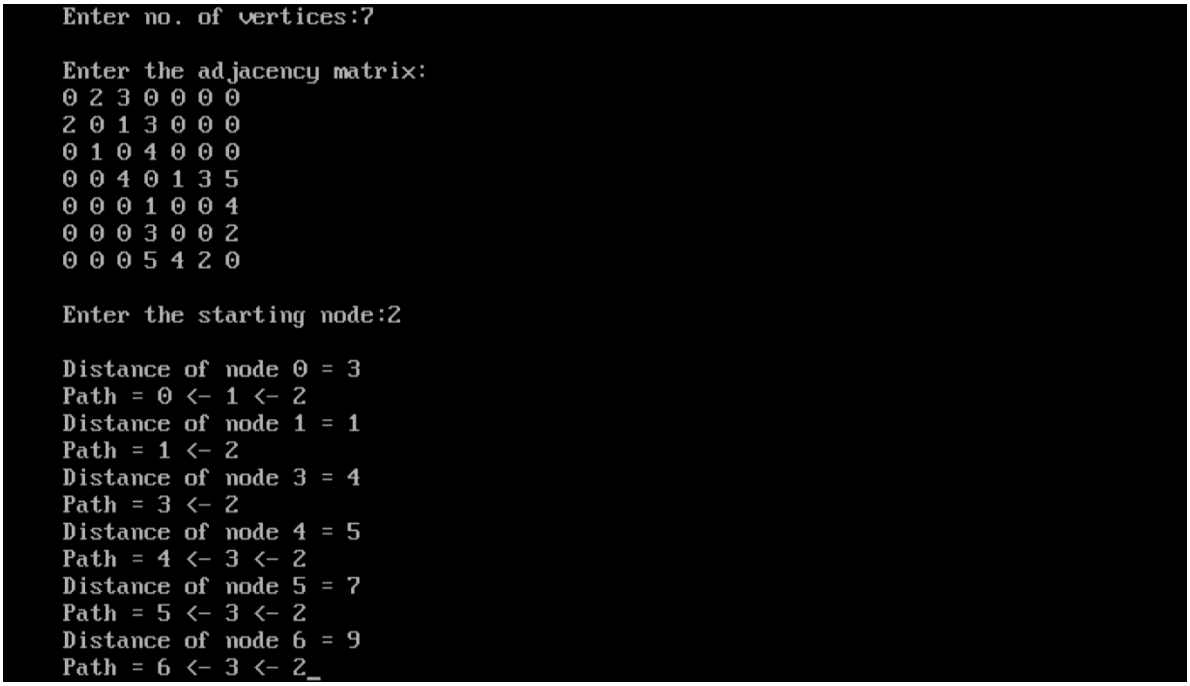
```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
```

```
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);
int main()
{
int G[MAX][MAX],i,j,n,u;
clrscr();
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);
getch();
}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{
int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];
for(i=0;i<n;i++)
{
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;count=1;
while(count<n-1)
{
mindistance=INFINITY;
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
```

```
if(mindistance+cost[nextnode][i]<distance[i])

distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node %d = %d",i,distance[i]);
printf("\nPath = %d",i);
j=i;
do
{
j=pred[j];
printf(" <- %d",j);
}while(j!=startnode);
}
}
```

RESULT : THE REQUIRED OUTPUT IS OBTAINED.

OUTPUT


Enter no. of vertices:7
Enter the adjacency matrix:
0 2 3 0 0 0 0
2 0 1 3 0 0 0
0 1 0 4 0 0 0
0 0 4 0 1 3 5
0 0 0 1 0 0 4
0 0 0 3 0 0 2
0 0 0 5 4 2 0
Enter the starting node:2
Distance of node 0 = 3
Path = 0 <- 1 <- 2
Distance of node 1 = 1
Path = 1 <- 2
Distance of node 3 = 4
Path = 3 <- 2
Distance of node 4 = 5
Path = 4 <- 3 <- 2
Distance of node 5 = 7
Path = 5 <- 3 <- 2
Distance of node 6 = 9
Path = 6 <- 3 <- 2_