MOHANDAS COLLEGE OF ENGINEERING AND TECHNOLOGY

ANAD, NEDUMANGAD THIRUVANANTHAPURAM-695544

.....

LAB RECORD

Name: .....

Reg No: .....

Class: .....Branch: .....

From: .....To: .....

MOHANDAS COLLEGE
OF ENGINEERING AND TECHNOLOGY

ANAD, NEDUMANGAD THIRUVANANTHAPURAM-695544

……

LAB RECORD

Name: ……..

Reg No: ……..

Class: ……Branch: ……

From: ……To: ……

Certified Bonafide Record of Work Done By

…… Place:

……

Date: …… **(Staff in Charge)**

Examiners

External **Internal**

INDEX

SL.NO	NAME OF THE EXPERIMENT	DATE	PAGE NO	SIGNATURE

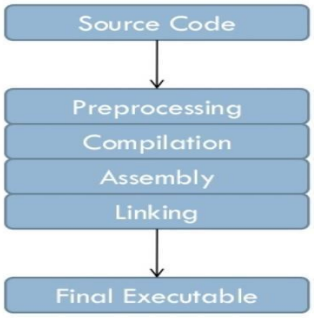
Experiment No :1

Date: 19-09-2023

Advanced GCC Commands In LINUX

GCC stands for GNU Compiler Collections. GCC Compiler is a very powerful and popular C compiler for various Linux distributions.

Compiling C files with gcc



GCC Compiler Options

Specify the Output Executable Name

Source File Name is main.c

In its most basic form, gcc compiler can be used as : gcc main.c

The above command executes the complete compilation process and outputs an executable with name a.out.

Writes the build output to an output file using –o option.

Use option -o, as shown below, to specify the output file name for the executable.

Syntax:

```
gcc main.c -o main
```

The command above would produce an output file with name ‘main’.

Produce only the compiled code using the -c option

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

Syntax:

```
gcc -c main.c
```

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

Use compile time macros using -D option

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

Syntax:

```
$ gcc -Dname [options] [source files] [-o output file]
$ gcc -Dname=definition [options] [source files] [-o output file]
```

Example

```
// main.c
#include <stdio.h> void main()
{
    #ifdef DEBUG printf("Debug
    run\n");
    #else
    printf("Release run\n"); #endif
}
```

Build *main.c* and run it with DEBUG defined:

```
$ gcc -D DEBUG main.c -o main
$ ./main
```

Or build *main.c* and run it without DEBUG defined:

```
$ gcc main.c -o main
```

```
$ ./main Release run
$
```

Link with shared libraries using -l option

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

Syntax:

```
$ gcc [options] [source files] [object files] [-Ldir] -llibname [-o outfile]
```

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’.

Include directory of header files using –I option

The option –I can be used to include directory of header files

Syntax:

```
$ gcc -Idir [options] [source files] [object files] [-o output file]
```

Example:

proj/src/myheader.h:

```
// myheader.h #define
NUM1 5
//main.c

#include <stdio.h> #include
"myheader.h" void main()
{
        int num = NUM1; printf("num=%d\n",
        num);
}
```

Debug information to be used by -g option

gcc -g generates debug information to be used by GDB debugger with 4 options –g0,-g1,-g,-g3

Syntax:

$ gcc -glevel [options] [source files] [object files] [-o output file]

$ gcc -g main.c -o main

$ gdb main

Setting the compiler optimization level using-O option

Set the compiler's optimization level several options such as -O0, -O1 ,-O2 ,-O3, - Os, -Ofast

Syntax:

$ gcc -Olevel [options] [source files] [object files] [-o output file]

$ gcc -O main.c -o main

Produce all the intermediate files using -save-temps function

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

Syntax:

$ gcc -save-temps source file

Example:

$ gcc -save-temps main.c

$ ls

a.out main.c main.i main.o main.s

main.i(preprocessed output) main.o(complied code) main.s(Assembly output)

GDB Commands in LINUX

The GNU Debugger, commonly abbreviated as GDB, is a command line tool that can be used to debug programs written in various programming languages such as C,C++,Ada, Fortran etc. The console can be using gdb command on terminal. gdb help us to find out watch or modify the variables in runtime, current state of the program, change the execute flow dynamically etc.

Compiling Programs with Debugging Information

To compile a C program with debugging information that can be read by the **GNU Debugger**, make sure the gcc compiler is run with the -g option.

Syntax:

$ gcc -g -o *output_file input_file*

Example:

$gcc -g -o main main.c

Common GDB Commands

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 without saying the program name. i.e., it runs the loaded executable program with program arguments arg1 ... argn.

Syntax:

[r]un (arg1 arg2 ... argn):

Example:

(gdb) run 2048 24 4

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. It set a

breakpoint on either a function, a line given by a line number, or the instruction located at a particular address.

Syntax:

[b]reak <function name or file name: line# or *memory address>

Example:

(gdb) break main

Breakpoint 1 at 0x80488f6: file main.c, line 67.

next

"next" will continue until the next source line in the current function

Syntax:

[n]ext

Example:

(gdb) next

print

"print" *expression* will print out the value of the expression, which could be just a variable name.

Syntax:

[p]rint <expression>

Example:

(gdb) print list[0]@25

It prints out the first 25 values in an array called list

Display

Set an Automatic Display. It displays an expression every time the program stops.

Syntax:

display expression

Example:

(gdb) display/I $pc

It would display the next instruction after each step.

6. help

Gdb provides online documentation. Just typing help will give you a list of topics. Then you can type help *topic* /command to get information about that topic/command.

Syntax:

[h]elp [topic/command]

Example:

(gdb) help

gprof: LINUX GNU GCC Profiling Tool

Profiling allows you to learn where your program spent its time and which functions called which other functions while it was executing. 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.

GNU profiling tool 'gprof' produces an execution profile of C, Pascal, or Fortran77 programs. `Gprof` calculates the amount of time spent in each routine. Next, these times are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle.

gprof is executing through the following steps:

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).

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. (-pg : Generate extra code to write profile information suitable for the analysis program gprof.)

```
$ gcc -pg testgprof.c new_testgprof.c -o testgprof
```

The option '-pg' can be used with the gcc command that compiles (-c option), gcc command that links(-o option on object files).

Step-2 : Execute the code

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

```
$ ./testgprof
```

If the profiling is enabled then on executing the program, file gmon.out (in built profile file for dynamic call graph and profile) will be generated.

```
$ ls
gmon.out testgprof testgprof.c
```

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 testgprof gmon.out > analysis.txt
```

Experiment No: 2

Date: 24-09-2024

MERGE TWO SORTED ARRAY AND STORE IN A THIRD ARRAY

AIM: To implement merge two sorted array.

ALGORITHM:

Step1: Start

Step2: Read the size of array 1

Step3: Enter the elements of array1

Step4: Read the size of second array

Step5: Enter the elements of array2

Step6: Initialize i=0,j=0

Step7: while(i<m&&j<n)

Step8: if(arr1[i]<arr[j][)then

Step9: array3[k]=array1[i]and increment i

Step10:else

Step11: array3[k]=array1[j]andincrementj

Step12: increment k

Step13: if(i>=m)then

Step14: while(j<n)

Step15: array3[k]=array1[j]andincrementj&k

Step16: if(j>=n)then

Step17: array3[k]=array1[i]andincrementi&k

Step18: print the array after merging

Step19:stop

SOURCE CODE:

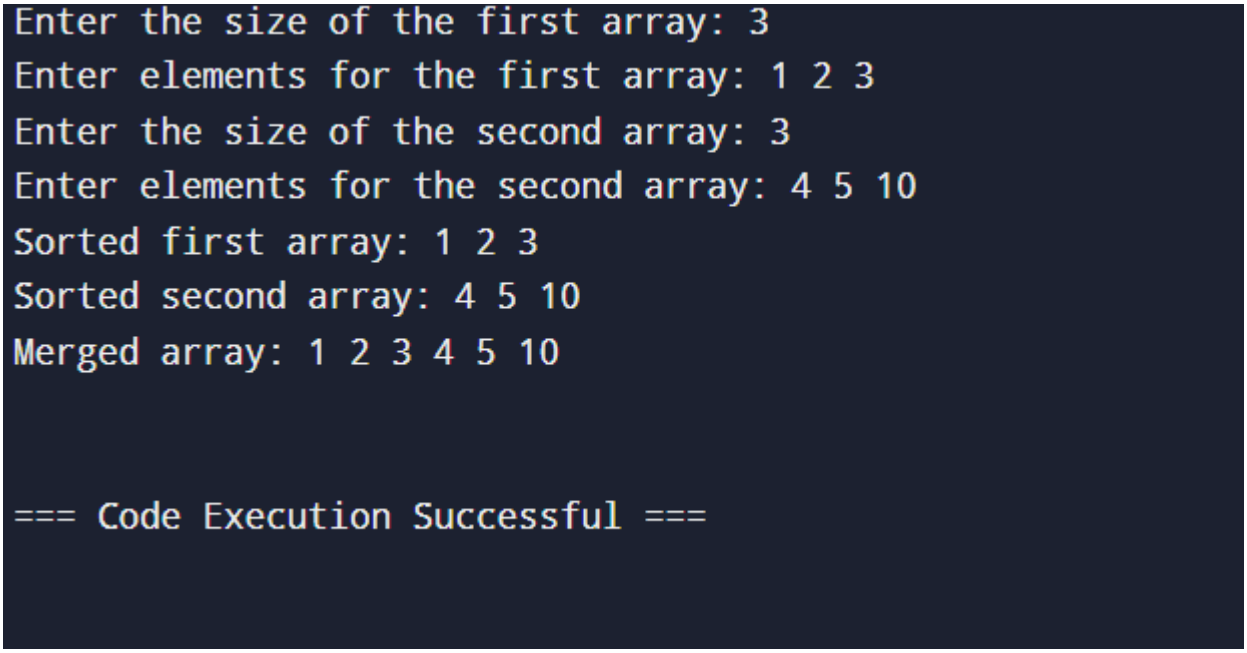
```
#include<stdio.h>
#include<stdlib.h>

void main()
{
 int i,j,k,m,n,a[10],b[10],c[20];
 clrscr();
 printf("\n enter size of first array:");
 scanf("%d",&m);
 printf("\n Enter sorted elements of first array:");
 for(i=0;i<m;i++)
 {
  scanf("%d",&a[i]);
 }
  printf("\n Enter size of second array:");
  scanf("%d",&n);
  printf("\n Enter sorted elements of second array:");
  for(j=0;j<n;j++)
  {
   scanf("%d",&b[j]);
  }
  i=j=k=0;
  for(k=0;k<m+n;k++)
  {
   if(i<m && j<n)
   {
    if(a[i]<b[j])
     {
      c[k]=a[i];
      i++;
```

```
      }
      else
      {
       c[k]=b[j];
       j++;
      }
     }
    else if(i<n)
     {
      c[k]=a[i];i++;
     }
    else
     {
      c[k]=b[j];j++;
     }
    }
   printf("\n The merged array is: ");
   for(k=0;k<m+n;k++)
   {
    printf("%d",c[k]);
  }
  getch();
}
```

RESULT: Output is obtained and the result is verified.

OUTPUT:


Enter the size of the first array: 3
Enter elements for the first array: 1 2 3
Enter the size of the second array: 3
Enter elements for the second array: 4 5 10
Sorted first array: 1 2 3
Sorted second array: 4 5 10
Merged array: 1 2 3 4 5 10
=== Code Execution Successful ===

Experiment No: 3

Date: 07-10-2024

CIRCULAR QUEUE USING ARRAY

AIM: To implement a circular queue using array.

ALGORITHM:

step1:start
Step2:enter the size of circular queue and assign to n
step3:enter the choice
step4:if choice=1 then do insertion
Step 5:if choice= 2 then do deletion
Step6:if choice=3 then display circular queue
Step7:if choice =4 then exit
step8:stop

Insert Operation
step1:start
step2:if((front==1&& rear==n)||(front==rear+1))the print queue is full
step3:elseif(front==NULL)then front=rear=1
step4:elseif(rear==n)thenrear=1
step5:elserear++
Step6:read the element to inserted and assign to y
step7:queue[rear]=y
step8:stop

Delete Operation
step1:start
step2:if(front==NULL)then print queue is empty
step3:else
Step4:delete queue[front]
step5:if(front==rear)then front=rear=NULL
step6:elseif(front==n)then front=1
Step7:else front++
step8:stop

Display Circular Queue
Step1:start
step2:if(front>rear)then
step3:for(i=front;i<=n;i++)
Step4:print queue[i]
step5:for(i=1;i<=rear;i++)
Step6:print queue[i]
step7:else
step8:for(i=front;i<=rear;i++)
step9:print queue[i]
step10:stop

SOURCE CODE:

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

#define MAX 5

int queue[MAX];
int front = -1;
int rear = -1;

int isFull() {
   return (front == (rear + 1) % MAX);
}

int isEmpty() {
   return front == -1;
}

void enqueue() {
int value;
   if (isFull()) {
      printf("Queue Overflow! Cannot enqueue more elements.\n");
      return;
   }

   printf("Enter value to enqueue: ");
```

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

    if (isEmpty()) {
        front = 0;
    }
    rear = (rear + 1) % MAX;
    queue[rear] = value;
    printf("Enqueued %d onto the queue.\n", value);
}

void dequeue() {
    if (isEmpty()) {
        printf("Queue Underflow! No elements to dequeue.\n");
        return;
    }
    printf("Dequeued %d from the queue.\n", queue[front]);

    if (front == rear) {
            front = rear = -1;
    } else {
        front = (front + 1) % MAX;
   }
}

void checkFirst() {
    if (isEmpty()) {
        printf("Queue Underflow! No elements in the queue.\n");
    } else {
        printf("First element: %d\n", queue[front]);
    }
}

void checkLast() {
    if (isEmpty()) {
        printf("Queue Underflow! No elements in the queue.\n");
    } else {
        printf("Last element: %d\n", queue[rear]);
    }
}

void display() {
```

```
 int i = front;
   if (isEmpty()) {
        printf("Queue Underflow! No elements in the queue to display.\n");
        return;
   }
   printf("Queue elements: ");
   while (i != rear) {
        printf("%d ", queue[i]);
        i = (i + 1) % MAX;
   }
   printf("%d\n", queue[rear]);
}

void search() {
int value,i = front,found = 0;
   if (isEmpty()) {
        printf("Queue Underflow! No elements in the queue to search.\n");
        return;
   }


   printf("Enter value to search: ");
   scanf("%d", &value);
   while (1) {
        if (queue[i] == value) {
           printf("Value %d found at position %d.\n", value, (i + 1) % MAX);
           found = 1;
           break;
       }
       if (i == rear) break;
       i = (i + 1) % MAX;
   }

   if (!found) {
      printf("Value %d not found in the queue.\n", value);
   }
}

int main() {
   int choice;
   while (1) {
```

```
        printf("\nMenu:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Check First Element\n");
        printf("4. Check Last Element\n");
        printf("5. Display Queue\n");
        printf("6. Search in Queue\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                checkFirst();
                break;
            case 4:
                checkLast();
                break;
            case 5:
                display();
                break;
            case 6:
                search();
                break;
            case 7:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

RESULT: Output is obtained and the result is verified.

OUTPUT:

```
Menu:
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
5. Display Queue
6. Search in Queue
7. Exit
Enter your choice: 1
Enter value to enqueue: 33
Enqueued 33 onto the queue.

Menu:
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
5. Display Queue
6. Search in Queue
7. Exit
Enter your choice: 1
Enter value to enqueue: 34
Enqueued 34 onto the queue.
```

```
Menu:
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
5. Display Queue
6. Search in Queue
7. Exit
Enter your choice: 1
Enter value to enqueue: 37
Enqueued 37 onto the queue.

Menu:
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
5. Display Queue
6. Search in Queue
7. Exit
Enter your choice: 3
First element: 33
```

```
Menu:
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
5. Display Queue
6. Search in Queue
7. Exit
Enter your choice: 3
First element: 33

Menu:
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
5. Display Queue
6. Search in Queue
7. Exit
Enter your choice: 4
Last element: 37
```

```
Enter your choice: 4
Last element: 37

Menu:
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
5. Display Queue
6. Search in Queue
7. Exit
Enter your choice: 5
Queue elements: 33 35 37

Menu:
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
5. Display Queue
6. Search in Queue
7. Exit
Enter your choice: 6
Enter value to search: 23
Value 23 not found in the queue.
```

```
Enter your choice: 6
Enter value to search: 23
Value 23 not found in the queue.

Menu:
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
5. Display Queue
6. Search in Queue
7. Exit
Enter your choice: 35
Invalid choice! Please try again.

Menu:
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
5. Display Queue
6. Search in Queue
7. Exit
Enter your choice: 6
Enter value to search: 35
Value 35 found at position 2.
```

```
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
5. Display Queue
6. Search in Queue
7. Exit
Enter your choice: 6
Enter value to search: 35
Value 35 found at position 2.

Menu:
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
5. Display Queue
6. Search in Queue
7. Exit
Enter your choice: 2
Dequeued 33 from the queue.

Menu:
1. Enqueue
2. Dequeue
3. Check First Element
4. Check Last Element
```

Experiment No: 4

Date:15-10-2024

a) SINGLY LINKED STACK

AIM: To implement singly linked stack

ALGORITHM:

Step1:start
Step2:Define a structure ‘node’ with two member val and next
Step3:Define a node pointer head
Step4:enter the choice
Step5:if choice==1t hen do push operation
Step6:of choice==2 then do pop operation
Step7:of choice==3 then display/traverse linked list
Step8:of choice==4t hen exit
Step9:stop

Push Operation:
Step1:start
Step2:createanewnode
Step3:if(head==NULL)then
Step4:ptr->val=val
Step5:ptr->next=NULL
Step6:head=ptr
Step7:else
Step8:ptr->val=val
Step9:ptr->next=head
Step10:head=ptr
Step11:printitempushed
Step12:stop

Pop Operation:
Step1:start
Step2:if(head==NULL)then

Step3:print underflow
Step4:else
Step5:item=head->val
Step6:ptr=head
Step7:head=head->next
Step8:free(ptr)
Step9:printItempopped
Step10:stop

Display/traverse Linked List:
Step1:start
Step2:if(ptr==NULL)then
Step3:print stack is empty
Step4:else
Step5:while(ptr!=NULL)
Step6:print the elements in the stack
Step7:stop

SOURCE CODE:

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

struct Node {
   int data;
   struct Node* next;
};

struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}
```

```
void push(struct Node** top, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *top;
   *top = newNode;
    printf("Inserted %d\n", data);
}

int pop(struct Node** top) {
    if (*top == NULL) {
        printf("Stack Underflow\n");
        return -1;
    }
    struct Node* temp = *top;
    int poppedData = temp->data;
    *top = (*top)->next;
   free(temp);
  printf("Popped %d\n", poppedData);
    return poppedData;
}

void traverse(struct Node* top) {
    if (top == NULL) {
        printf("Stack is empty\n");
        return;
    }
    struct Node* temp = top;
    printf("Stack elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Node* top = NULL;
 int choice, value;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Push\n");
```

```
        printf("2. Pop\n");
        printf("3. Traverse\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to push: ");
                scanf("%d", &value);
                push(&top, value);
                break;
            case 2:
                pop(&top);
                break;
            case 3:
                traverse(top);
                break;
            case 4:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

RESULT: Output is obtained and the result is verified.

OUTPUT:

```
Menu:
1. Push
2. Pop
3. Traverse
4. Exit
Enter your choice: 1
Enter the value to push: 22
Inserted 22

Menu:
1. Push
2. Pop
3. Traverse
4. Exit
Enter your choice: 1
Enter the value to push: 24
Inserted 24

Menu:
1. Push
2. Pop
3. Traverse
4. Exit
Enter your choice: 1
```

```
Menu:
1. Push
2. Pop
3. Traverse
4. Exit
Enter your choice: 1
Enter the value to push: 26
Inserted 26

Menu:
1. Push
2. Pop
3. Traverse
4. Exit
Enter your choice: 3
Stack elements: 26 24 22

Menu:
1. Push
2. Pop
3. Traverse
4. Exit
Enter your choice: 2
Popped 26
```

```
Menu:
1. Push
2. Pop
3. Traverse
4. Exit
Enter your choice: 2
Popped 24

Menu:
1. Push
2. Pop
3. Traverse
4. Exit
Enter your choice: 2
Popped 22

Menu:
1. Push
2. Pop
3. Traverse
4. Exit
Enter your choice: 2
Stack Underflow
```

Experiment No: 4

Date: 15-10-2024

b) DOUBLY LINKED LIST

AIM: To implement doubly linked list

ALGORITHM:

step1:start

Step2:define a structure node

Step3:initialize data andpointers*previous,*next and assign *head=NULL

step4:read the choice

Step5:if choice=1 then do insertion

Step6:read the value to be inserted and assign to value

step7:Select the insertion position

Step8:if choice=1 then insertion

step9:if choice=2 then do deletion

step10:read the value to be deleted and assign to value

Step11:if choice=3 then Search a with position

Step12:read the value to be searched and assign it with the position

step13:if choice=4 then do display list

step14:if(head==NULL)the print List is empty

step15:else initialize*temp=head

step16:while(temp->next!=NULL)

step17:printemp->data

Step18:if choice=5 then exit

step19:stop

SOURCE CODE:

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

struct Node{
 int data;
 struct Node* prev;
 struct Node* next;
};

struct Node* createNode(int data)
 {
  struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
  newNode->data=data;
  newNode->prev=NULL;
  newNode->next=NULL;
  return newNode;
}
void insertEnd(struct Node **head,int data)
 {
  struct Node *newNode=createNode(data);
  struct Node*temp=*head;
if(*head==NULL)
{
 *head=newNode;
 return;
}
while(temp->next!=NULL)
```

```
{
 temp=temp->next;
}
temp->next=newNode;
newNode->prev=temp;
}
void deleteNode(struct Node **head, int key)
{
 struct Node *temp=*head;
 if(temp==NULL)
 {
  printf("\n list is empty");
  return;
}
while(temp!=NULL && temp->data!=key)
{
 temp=temp->next;
}
if(temp==NULL)
{
 printf("\n value %d not found in the key",key);
 return;
}
if(temp==*head)
 {
  *head=temp->next;
}
if(temp->next!=NULL)
{
 temp->next->prev=temp->prev;
}
if(temp->prev!=NULL)
{
 temp->prev->next=temp->next;
}
free(temp);
printf("\n value %d deleted from the list",key);
}
void searchNode(struct Node *head,int key)
{
  struct Node *temp=head;
```

```
 int pos=1;
 while(temp!=NULL)
{
 if(temp->data==key)
  {
   printf("\n %d found at position in the list",pos, key);
   return;
  }
  temp=temp->next;
  pos++;
}
printf("\n %d is not found in the list",key);
}
void display(struct Node *head){
struct Node *temp=head;
printf("\n Doubly linked list ");
while(temp!=NULL)
{
 printf("\n %d",temp->data);
 temp=temp->next;
}
printf("\n");
}
void main()
{
 struct Node *head=NULL;
 int choice,value;
 clrscr();
 while(1)
 {
 printf("\n 1.Insertnode");
 printf("\n 2.Deletenode");
 printf("\n 3.Searchnode");
 printf("\n 4.Displaylist");
 printf("\n 5.Exit");
 printf("\n Enter your choice:");
 scanf("\n %d",&choice);
 switch(choice)
 {
   case 1:printf("\n Enter the elements to insert");
          scanf("\n%d",&value);
```

```
        insertEnd(&head,value);
        break;
    case 2:printf("\n Enter the elements to delete");
        scanf("\n%d",&value);
        deleteNode(&head,value);
        break;
    case 3:printf("\n Enter the elements to search");
        scanf("\n%d",&value);
        searchNode(head,value);
        break;
    case 4:display(head);
        break;
    case 5:exit(0);
        break;
    default:printf("\n Invalid choice");
}
}
getch();
}
```

RESULT: Output is obtained and the result is verified.

OUTPUT:

```
1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
Enter your choice: 1

Enter the element to insert: 24

1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
Enter your choice: 1

Enter the element to insert: 27

1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
Enter your choice: 1
```

```
Enter your choice: 1

Enter the element to insert: 30

1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
Enter your choice: 1

Enter the element to insert: 33

1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
Enter your choice: 3

Enter the element to search: 11

 11 is not found in the list
```

```
1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
Enter your choice: 3

Enter the element to search: 27

 27 found at position 2 in the list
1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
Enter your choice: 4

Doubly linked list: 24 27 30 33

1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
```

```
Enter your choice: 2

Enter the element to delete: 23

Value 23 not found in the list
1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
Enter your choice: 2

Enter the element to delete: 24

Value 24 deleted from the list
1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
Enter your choice: 2

Enter the element to delete: 27

Value 27 deleted from the list
```

```
Enter your choice: 2

Enter the element to delete: 33

Value 33 deleted from the list
1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
Enter your choice: 4

Doubly linked list:

1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
Enter your choice: 2

Enter the element to delete: 23

List is empty
```

```
List is empty
1. Insert node
2. Delete node
3. Search node
4. Display list
5. Exit
Enter your choice: 5


=== Code Execution Successful ===
```

Experiment No: 5

Date:22-10-2024

SET OPERATIONS USING BIT STRING

AIM: To perform Union, Intersection and Difference operations using Bitstring.

ALGORITHM:

input()

STEP 1 : Read size and elements of Set 1 and Set 2

union()

STEP 1 : for i 1 to 9 do
STEP 1.1 : if(a[i] != b[i]) then set c[i]=1
STEP 1.2 : else set c[i] = a[i]
STEP 2 : End for
STEP 3 : Display c

INTERSECTION()

STEP 1 : for i 1 to 9 do
STEP 1.1 : if(a[i] == b[i])then set c[i]=a[i]
STEP 1.2 : else set c[i] = 0
STEP 2 : End for
STEP 3 : Display c

Difference()

STEP 1 : for i 1 to 9 do
STEP 1.1 : if(a[i] == 1 && b[i] == 0) then set c[i]=1
STEP 1.2 : else set c[i] = 0
STEP 2 : End for
STEP 3 : Display c

SOURCE CODE:

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


void input();
void setunion();
void intersection();
void difference();
void output(int c[]);


int a[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int b[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

int main() {
   int ch, wish;
   clrscr();
   do {
         printf("\nMENU\n");
         printf("1. Input\n2. Union\n3. Intersection\n4. Difference\n");
         printf("Enter your choice: ");
         scanf("%d", &ch);

         switch (ch) {
            case 1: input(); break;
            case 2: setunion(); break;
            case 3: intersection(); break;
            case 4: difference(); break;
            default: printf("Wrong choice");
         }

         printf("\nDo you wish to continue? (1 for yes / 0 for no): ");
         scanf("%d", &wish);
   } while (wish == 1);
```

```
    getch();
    return 0;
}

void input() {
    int n, e, i;

    printf("Enter the size of first set: ");
    scanf("%d", &n);
    printf("Enter the elements (1 to 9 only):\n");
    for (i = 1; i <= n; i++) {
        scanf("%d", &e);
        if (e >= 1 && e <= 9) {
            a[e] = 1;
        } else {
         printf("Invalid element, only numbers 1-9 are allowed.\n");
       }
    }

    printf("Enter the size of second set: ");
    scanf("%d", &n);
    printf("Enter the elements (1 to 9 only):\n");
    for (i = 1; i <= n; i++) {
        scanf("%d", &e);
        if (e >= 1 && e <= 9) {
            b[e] = 1;
        } else {
            printf("Invalid element, only numbers 1-9 are allowed.\n");
        }
    }

    printf("\nFirst set: ");
    for (i = 1; i <= 9; i++) {
        printf("%d ", a[i]);
    }

    printf("\nSecond set: ");
    for (i = 1; i <= 9; i++) {
        printf("%d ", b[i]);
    }
```

```
}

void output(int c[]) {
   int i;
   printf("\nResulting set: ");
   for (i = 1; i <= 9; i++) {
         if (c[i] != 0) {
            printf("%d ", i);
         }
   }
   printf("\n");
}

void setunion() {
   int i, c[10];
   for (i = 1; i <= 9; i++) {
         c[i] = a[i] | b[i];
   }
   printf("\nUnion: ");
   for (i = 1; i <= 9; i++) {
         printf("%d ", c[i]);
   }
   output(c);
}

void intersection() {
   int i, c[10];
   for (i = 1; i <= 9; i++) {
      c[i] = a[i] & b[i];
   }
   printf("\nIntersection: ");
   for (i = 1; i <= 9; i++) {
      printf("%d ", c[i]);
   }
   output(c);
}

void difference() {
   int i, c[10];
   for (i = 1; i <= 9; i++) {
      c[i] = (a[i] == 1 && b[i] == 0) ? 1 : 0;
```

```
    }
    printf("\nDifference (A - B): ");
    for (i = 1; i <= 9; i++) {
        printf("%d ", c[i]);
    }

    output(c);
}
```

RESULT: Output is obtained and the result is verified.

OUTPUT:

```
MENU
1. Input
2. Union
3. Intersection
4. Difference
Enter your choice: 1
Enter the size of first set: 4
Enter the elements (1 to 9 only):
1 2 3 5
Enter the size of second set: 4
Enter the elements (1 to 9 only):
4 5 6 7

First set: 1 1 1 0 1 0 0 0 0
Second set: 0 0 0 1 1 1 1 0 0
Do you wish to continue? (1 for yes / 0 for no): 1
```

```
MENU
1. Input
2. Union
3. Intersection
4. Difference
Enter your choice: 2

Union: 1 1 1 1 1 1 1 0 0
Resulting set: 1 2 3 4 5 6 7
```

```
Do you wish to continue? (1 for yes / 0 for no): 1

MENU
1. Input
2. Union
3. Intersection
4. Difference
Enter your choice: 3

Intersection: 0 0 0 0 1 0 0 0 0
Resulting set: 5

Do you wish to continue? (1 for yes / 0 for no): 1

MENU
1. Input
2. Union
3. Intersection
4. Difference
Enter your choice:
```

```
Do you wish to continue? (1 for yes / 0 for no): 1

MENU
1. Input
2. Union
3. Intersection
4. Difference
Enter your choice: 4

Difference (A - B): 1 1 1 0 0 0 0 0 0
Resulting set: 1 2 3

Do you wish to continue? (1 for yes / 0 for no): 0
```

Experiment No: 6

Date:04-11-2024

DISJOINT SET OPERATIONS

AIM: To perform Create , Union , Find operations of Disjoint Set.

ALGORITHM:

makeSet()

STEP 1 : For I to dis.n

STEP 2 : parent[x]=x

STEP 3 : rank[x]=0

STEP 4 : end for

Union()

STEP 1 : LINK(findSet(x), FindSet(y)

STEP 2 : LINK(x,y)

STEP 2.1 : if rank[x]>rank[y] then p[y]=x

STEP 2.2 : else p[x]=y

STEP 2.3 : if rank[x] == rank[y]

STEP 2.4 : then rank[y]==rank[y]+1

findSet(x)

STEP 1 : if x == p[x] then p[x] = findSet(p[x])

STEP 2 : return p[x]

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct DisJoint{
        int parent[10];
        int rank[10];
        int n;
}dis;
void makeset()
{
        int i;
        for(i=0;i<dis.n;i++){
                dis.parent[i]=i;
                dis.rank[i]=0;
        }
}
void display()
{
        int i;
        printf("\nParent");
        for(i=0;i<dis.n;i++)
        {
                printf("%d\t",dis.parent[i]);
```

```
	}
	printf("\nRank");
	for(i=0;i<dis.n;i++)
	{
		printf("%d\t",dis.rank[i]);
	}
}

int find(int x)
{
	if(dis.parent[x]!=x)
	{
		dis.parent[x]=find(dis.parent[x]);
	}
	return dis.parent[x];
}

void Union(int x,int y)
{
	int setx=find(x);
	int sety=find(y);
	if(setx==sety)
		return;
	if(dis.rank[setx]>dis.rank[sety])
```

```
        {
                dis.parent[sety]=setx;
                dis.rank[sety]=-1;
        }
        else if(dis.rank[setx]<dis.rank[sety])
        {
                dis.parent[setx]=sety;
                dis.rank[setx]=-1;
        }
        else
        {
                dis.parent[sety]=setx;
                dis.rank[setx]+=1;
                dis.rank[sety]=-1;
        }
}
void main()
{
        int choice,x,y;
        while(1)
        {
                printf("\nDisjoint Set Operations");
                printf("\n------------------------");
                printf("\n1.Make Sets");
```

```
printf("\n2.Find");
printf("\n3.Union");
printf("\n4.Display");
printf("\n5.Exit");
printf("\nEnter your choices:");
scanf("%d",&choice);

switch(choice)
{
    case 1:printf("\nEnter no of elements in a set:");
           scanf("%d",&dis.n);
           makeset();
           break;
    case 2:printf("\nEnter two elements to find they are connected:");
           scanf("%d%d",&x,&y);
           if(find(x)==find(y))
                printf("\n%d and %d are connected to a single leader
%d",x,y,find(x));
           else
                printf("\nDisconnected components");break;
    case 3: printf("\nEnter two elements to perform union:");
           scanf("%d%d",&x,&y);
           Union(x,y); break;
    case 4:display();
            break;
```

```
                    case 5:exit(1);
                }
        }
}
```

RESULT: Output is obtained and the result is verified.

OUTPUT:

```
Disjoint Set Operations
-----------------------
1.Make Sets
2.Find
3.Union
4.Display
5.Exit
Enter your choices:1

Enter no of elements in a set:6

Disjoint Set Operations
-----------------------
1.Make Sets
2.Find
3.Union
4.Display
5.Exit
Enter your choices:4

Parent0 1   2   3   4   5
Rank0   0   0   0   0   0
Disjoint Set Operations
-----------------------
```

```
1.Make Sets
2.Find
3.Union
4.Display
5.Exit
Enter your choices:3

Enter two elements to perform union:3 5

Disjoint Set Operations
-----------------------
1.Make Sets
2.Find
3.Union
4.Display
5.Exit
Enter your choices:4
```

```
-------------------------
1.Make Sets
2.Find
3.Union
4.Display
5.Exit
Enter your choices:4

Parent0 1   2   3   4   3
Rank0   0   0   1   0   -1
Disjoint Set Operations
------------------------
1.Make Sets
2.Find
3.Union
4.Display
5.Exit
```

```
1.Make Sets
2.Find
3.Union
4.Display
5.Exit
Enter your choices:2

Enter two elements to find they are connected:3 4

Disconnected components
Disjoint Set Operations
----------------------
1.Make Sets
2.Find
3.Union
4.Display
5.Exit
```

Experiment No: 7

Date: 11-11-2024

BINARY SEARCH TREES

AIM: To perform stack binary search tree operations

ALGORITHM:

Step 1.1: Create a Node

Step 1.2: Allocate memory for a new node.

Step 1.3: Set the node’s data field to the given value.

Step 1.4: Set the node’s left and right pointers to NULL.

Step 1.5: Return the created node.

INSERTION

Step 2.1: Insert a Node

Step 2.2: If the tree is empty (root is NULL), create and return a new node as the root.

Step 2.3: If the data to insert is smaller than the current node's data, recursively insert into the left subtree.

Step 2.4: If the data to insert is greater than or equal to the current node's data, recursively insert into the right subtree.

Step 2.5: Return the updated root after insertion.

Find the Minimum Node in the Right Subtree

Step 3.1: Traverse down the left side of the right subtree.

Step 3.2: Keep moving left until you reach a node with no left child.

Step 3.3: Return the node with the smallest value in the right subtree.

Delete a Node

Step 4.1: If the root is NULL, return NULL (node not found).

Step 4.2: If the data to delete is smaller than the current node's data, recursively search the left subtree.

Step 4.3: If the data to delete is larger than the current node's data, recursively search the right subtree.

Step 4.4: If the data matches the current node's data:

Step 4.5: If the node has no children (leaf node), delete the node and return NULL.

Step 4.6: If the node has one child, replace the node with its child and return the child node.

Step 4.7: If the node has two children:

Step 4.7.1: Find the minimum node in the right subtree.

Step 4.7.2: Copy the minimum node's data to the current node.

Step 4.7.3: Recursively delete the minimum node from the right subtree.

Step 4.8: Return the updated tree after deletion.

Search for a Node

Step 5: If the current node is NULL, return false (node not found).

Step 5.1: If the data matches the current node's data, return true (node found).

Step 5.2: If the data is smaller than the current node's data, search the left subtree.

Step 5.3: If the data is larger than the current node's data, search the right subtree.

Step 5.4: Return true if the node is found, false if not.

Display the Tree (In-order Traversal):

Step 6: Recursively traverse the right subtree first.

Step 6.1: Print the current node's data with indentation to indicate depth.

Step 6.2: Recursively traverse the left subtree.

Main Program:

Step 1:Display a menu with options: Insert a node, Delete a node, Search for a node, Display the tree, Exit the program.

Step 2: Based on the user's input, perform the corresponding operation:

Step 3: For Insert: Prompt for data and insert the node into the tree.

Step 4: For Delete: Prompt for data and delete the node.

Step 5: For Search: Prompt for data and search the tree.

Step 6: For Display: Print the tree structure.

Step 7: For Exit: End the program.

SOURCE CODE:

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

// Structure for a node in the binary search tree
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Function for inserting a node in the binary search tree
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
```

```
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else {
        root->right = insert(root->right, data);
    }
    return root;
}

// Function to find the minimum value node in the right subtree
struct Node* findMin(struct Node* root) {
    while (root->left != NULL) {
        root = root->left;
    }
    return root;
}

// Function to delete a node in the binary search tree
struct Node* delete(struct Node* root, int data) {
  struct Node* temp = findMin(root->right);
       if (root == NULL) {
           printf("Node not found.\n");
           return root;
```

```
    }

    if (data < root->data) {
        root->left = delete(root->left, data);
    } else if (data > root->data) {
        root->right = delete(root->right, data);
    } else {
        // Node with one or no child
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }

        // Node with two children
      root->data = temp->data;
      root->right = delete(root->right, temp->data);
    }
    return root;
}
```

```
// Function to search a node in the binary search tree
int search(struct Node* root, int key) {
    if (root == NULL) {
        return 0;
    }
    if (root->data == key) {
        return 1;
    } else if (key < root->data) {
        return search(root->left, key);
    } else {
        return search(root->right, key);
    }
}

// Function to print the tree structure (in-order traversal)
void display(struct Node* root, int space) {
    int i;
    if (root == NULL) {
          return;
    }
    space += 10;
    display(root->right, space);
    printf("\n");
```

```
    for (i = 10; i < space; i++) {
        printf(" ");
    }
    printf("%d\n", root->data);
    display(root->left, space);
}

// Main function to interact with the user and perform operations
int main() {
    struct Node* root = NULL;
    int choice, data;

    do {
        printf("\nBinary Search Tree Operations:\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display Tree\n");
        printf("4. Search\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

```
    printf("Enter data to insert: ");
    scanf("%d", &data);
    root = insert(root, data);
    printf("Node inserted successfully.\n");
    break;

case 2:
    printf("Enter data to delete: ");
    scanf("%d", &data);
    root = delete(root, data);
    printf("Node deleted successfully.\n");
    break;

case 3:
    printf("Displaying the tree:\n");
    display(root, 0);
    break;

case 4:
    printf("Enter data to search: ");
    scanf("%d", &data);
    if (search(root, data)) {
        printf("Node found in the tree.\n");
    } else {
```

```
                printf("Node not found in the tree.\n");
            }
            break;

        case 5:
            printf("Exiting...\n");
            break;

        default:
            printf("Invalid choice! Please try again.\n");
    }
  } while (choice != 5);

  return 0;
}
```

RESULT: Output is obtained and the result is verified.

OUTPUT:

```
Binary Search Tree Operations:
1. Insert
2. Delete
3. Display Tree
4. Search
5. Exit
Enter your choice: 1
Enter data to insert: 12
Node inserted successfully.

Binary Search Tree Operations:
1. Insert
2. Delete
3. Display Tree
4. Search
5. Exit
Enter your choice: 3
Displaying the tree:

12
```

```
Binary Search Tree Operations:
1. Insert
2. Delete
3. Display Tree
4. Search
5. Exit
Enter your choice: 1
Enter data to insert: 15
Node inserted successfully.

Binary Search Tree Operations:
1. Insert
2. Delete
3. Display Tree
4. Search
5. Exit
Enter your choice: 3
Displaying the tree:

         15

12
```

```
Binary Search Tree Operations:
1. Insert
2. Delete
3. Display Tree
4. Search
5. Exit
Enter your choice: 1
Enter data to insert: 7
Node inserted successfully.

Binary Search Tree Operations:
1. Insert
2. Delete
3. Display Tree
4. Search
5. Exit
Enter your choice: 1
Enter data to insert: 27
Node inserted successfully.
```

```
Binary Search Tree Operations:
1. Insert
2. Delete
3. Display Tree
4. Search
5. Exit
Enter your choice: 3
Displaying the tree:

                    27

          15

12

          7
```

```
Binary Search Tree Operations:
1. Insert
2. Delete
3. Display Tree
4. Search
5. Exit
Enter your choice: 4
Enter data to search: 15
Node found in the tree.

Binary Search Tree Operations:
1. Insert
2. Delete
3. Display Tree
4. Search
5. Exit
Enter your choice: 4
Enter data to search: 11
Node not found in the tree.
```

```
Binary Search Tree Operations:
1. Insert
2. Delete
3. Display Tree
4. Search
5. Exit
Enter your choice: 2
Enter data to delete: 15
Node deleted successfully.

Binary Search Tree Operations:
1. Insert
2. Delete
3. Display Tree
4. Search
5. Exit
Enter your choice: 3
Displaying the tree:

          27


12


          7
```

```
Binary Search Tree Operations:
1. Insert
2. Delete
3. Display Tree
4. Search
5. Exit
Enter your choice: 5
Exiting...
```

Experiment No: 8

Date:18-11-2024

RED BLACK TREE

AIM: To implement a Red Black Tree and insert a node in it.

ALGORITHM:

Insert ():

Step 1.1: Create a new node z with the value to insert, colored Red.

Step 1.2: Traverse the tree to find the correct position for z.

Step 1.3: Insert z as the left or right child of its parent.

Step 1.4: Call the color function to restore Red-Black properties.

Restore Red-Black Properties (color):

Step 2.1: While z's parent is Red:

Step 2.2: Check if the uncle of z is Red:

Step 2.3: Case 1 (Uncle Red)**:** Recolor the parent, uncle, and grandparent.

Step 2.4: If the uncle is Black:

Step 2.5: Case 2 (Uncle Black): Perform a rotation (left or right) and recolor to fix violations.

Step 2.6: Ensure the root is always Black.

Rotations (leftRotate and rightRotate):

Step 3.1: Adjust pointers to perform:

Step 3.2: Left Rotate-Shift nodes left around a given node.

Step 3.3: Right Rotate-Shift nodes right around a given node.

Display the Tree (printTree):

Step 4: Recursively print the tree structure with node values and colors.

Main Function:

Step 1: Insert an element.

Step 2: Display the tree structure.

Step 3: Exit the program.

SOURCE CODE:

```
#include <stdio.h>

#include <stdlib.h>

#define RED 'R'

#define BLACK 'B'

struct Node

{

   int data;

   char color;

   struct Node *left, *right, *parent;

};

struct Node *root = NULL;

// Function prototypes
```

```
void leftRotate(struct Node *x);
void rightRotate(struct Node *y);
void color(struct Node *z);
void insert(int val);
void printTree(struct Node *root, int level, int space);
// Function to perform a left rotation
void leftRotate(struct Node *x)
{
    struct Node *y = x->right;
    x->right = y->left;
    if (y->left != NULL)
        y->left->parent = x;
    y->parent = x->parent;
    if (x->parent == NULL)
        root = y;
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;

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

```
}
// Function to perform a right rotation
void rightRotate(struct Node *y)
{
    struct Node *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 == y->parent->left)
        y->parent->left = x;
    else
        y->parent->right = x;
    x->right = y;
    y->parent = x;
}
// Function to adjust the color of the nodes after insertion
void color(struct Node *z)
{
```

```
while (z->parent != NULL && z->parent->color == RED)
{
   if (z->parent == z->parent->parent->left)
   {
      struct Node *y = z->parent->parent->right;

      if (y != NULL && y->color == RED)
      {
         z->parent->color = BLACK;
         y->color = BLACK;
         z->parent->parent->color = RED;
         z = z->parent->parent;
      }
      else
      {
         if (z == z->parent->right)
         {
            z = z->parent;
            leftRotate(z);
         }
         z->parent->color = BLACK;
         z->parent->parent->color = RED;
```

```
            rightRotate(z->parent->parent);
        }
    }
    else
    {
        struct Node *y = z->parent->parent->left;

        if (y != NULL && y->color == RED)
        {
            z->parent->color = BLACK;
            y->color = BLACK;
            z->parent->parent->color = RED;
            z = z->parent->parent;
        }
        else
        {
            if (z == z->parent->left)
            {
                z = z->parent;
                rightRotate(z);
            }
            z->parent->color = BLACK;
```

```
            z->parent->parent->color = RED;
            leftRotate(z->parent->parent);
        }
    }
    root->color = BLACK;
}
// Function to insert a node into the Red-Black Tree
void insert(int val)
{
    struct Node *z = (struct Node *)malloc(sizeof(struct Node));
     struct Node *y = NULL;
    struct Node *x = root;
    z->data = val;
    z->left = z->right = z->parent = NULL;
    z->color = RED;
    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);
}
// Function to print the Red-Black Tree in a hierarchical structure
void printTree(struct Node *root, int level, int space)
{
    int i;
    if (root == NULL)
        return;
    space += 5;
    printTree(root->right, level + 1, space);
    printf("\n");
    for (i = 5; i < space; i++)
```

```
        printf(" ");
    printf("%d%c\n", root->data, root->color);
    printTree(root->left, level + 1, space);
}
void main()
{
    int ch, val;
   clrscr();
    while (1)
    {
        printf("\nRed Black Tree Operation");
        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:
            printf("Red-Black Tree Structure:\n");
```

```
            printf("------------------------\n");
            printTree(root, 1, 0);
            printf("------------------------\n");
            break;
        case 3:
            exit(0);
        }
    }
}
```

RESULT: Output is obtained and the result is verified.

OUTPUT:

```
1.Insert
2.Display
3.Exit
Enter Choice:1
Enter element:22

1.Insert
2.Display
3.Exit
Enter Choice:2
Red Black Tree Structure:
--------------------------
22B
--------------------------
```

```
1.Insert
2.Display
3.Exit
Enter Choice:1
Enter element:26

1.Insert
2.Display
3.Exit
Enter Choice:2
Red Black Tree Structure:
--------------------------
     26R

22B
--------------------------
```

```
1.Insert
2.Display
3.Exit
Enter Choice:1
Enter element:17

1.Insert
2.Display
3.Exit
Enter Choice:2
Red Black Tree Structure:
--------------------------
     26R

22B

     17R
--------------------------
```

```
1.Insert
2.Display
3.Exit
Enter Choice:1
Enter element:37

1.Insert
2.Display
3.Exit
Enter Choice:2
Red Black Tree Structure:
-------------------------
         37R

    26B

22B

    17B
-------------------------
```

```
1.Insert
2.Display
3.Exit
Enter Choice:3


=== Code Execution Successful ===
```

Experiment No: 9

Date: 25-11-2024

GRAPH TRAVERSAL TECHNIQUES :BFS, DFS, TOPOLOGICAL SORTING

AIM: To perform BFS , DFS , Topological sorting.

ALGORITHM:

BFS (Breadth-First Search)

Step 1.1: Initialize all vertices as unvisited (visited[i] = 0).

Step 1.2: Input the starting vertex v.

Step 1.3: Mark v as visited and enqueue it.

Step 1.4: While the queue is not empty:

Step 1.5: Dequeue a vertex, print it, and enqueue all its unvisited adjacent vertices.

Step 1.6: Repeat until all reachable vertices are visited.

DFS (Depth-First Search)

Step 2.1: Initialize all vertices as unvisited (reach[i] = 0).

Step 2.2: Input the starting vertex v.

Step 2.3: For each adjacent vertex of v:

Step 2.4: If unvisited, recursively visit the vertex and mark it as reached.

Step 2.5: Print all visited vertices.

Step 2.6: After traversal:

Step 2.7: Check if all vertices were reached.

Step 2.8: If all vertices are visited, the graph is connected. Otherwise, it is

disconnected.

Topological Sort

Step 3.1: Calculate the in-degree for all vertices:

Step 3.2: For each vertex j, sum the values in column j of the adjacency matrix to compute indegree[j].

Step 3.3: Push all vertices with in-degree 0 into a stack.

Step 3.4: While the stack is not empty:

Step 3.5: Pop a vertex u from the stack and add it to the topological order.

Step 3.6: For each adjacent vertex v of u:

Step 3.7: Decrement its in-degree.

Step 3.8: If indegree[v] == 0, push it onto the stack.

Step 3.9: Print the topological order.

Exit

Step 4.1: Exit the program.

SOURCE CODE:

```
#include<stdio.h>

#include<conio.h>

#define MAX 100

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1,indegree[10];

int adj[MAX][MAX],reach[20];
```

```
int queue[MAX], front = -1,rear = -1;
void main()
{
 int ch,v,j,G[MAX][MAX],count=0;
 clrscr();
 do
 {
        Printf(“\nGraph Traversal”);
        printf("\n1 BFS\n2 DFS\n3 Topological Sort\n4 Exit \nEnter Choice:");
  scanf("%d",&ch);
  switch(ch)
  {
    case 1:
          printf("Enter the number of vertices:");
          scanf("%d",&n);
          for(i=1;i<=n;i++)
          {
           q[i]=0;
           visited[i]=0;
          }
          printf("\nEnter graph data in matrix form:\n");
          for(i=1;i<=n;i++)
```

```
        for(j=1;j<=n;j++)
        scanf("%d",&a[i][j]);
        printf("\nEnter the starting vertex:");
        scanf("%d",&v);
        visited[v]=1;
        printf("%d\t",v);
        bfs(v);
        break;
case 2:
        printf("\n Enter the 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");
        break;
case 3:
    printf("\n Enter the number of vertices:");
    scanf("%d",&n);
    printf("Enter the adjacency matrix:\n ");
    for(i=0;i<n;i++)
    {
    for(j=0;j<n;j++)
    scanf("%d",&a[i][j]);
    }
    topology();
    getch();
```

```
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid input");
        }
    }while(ch!=5);
    getch();
}
int 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;
   printf("%d\t",q[f]);
   bfs(q[f++]);
   }
 return 0;
}
```

```
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);
}
     printf("\n%d",v);
return 0;
}
void find_indegree()
{
 int i,j,sum;
 for(j=0;j<n;j++)
 {
  sum=0;
  for(i=0;i<n;i++)
   sum+=a[i][j];
 indegree[j]=sum;
```

```
 }
}
int topology()
{
 int i,u,v,t[10],s[10],top=-1,k=0;
 delay(1000);
 find_indegree();
 for(i=0;i<n;i++)
 {
  if(indegree[i]==0)s[++top]=i;
 }
 while(top!=-1)
 {
  u=s[top--];
  t[k++]=u;
  for(v=0;v<n;v++)
  {
   if(a[u][v]==1)
   {
    indegree[v]--;
    if(indegree[v]==0)s[++top]=v;
   }
```

```
  }
 }
 printf("The Topological Sequence is:");
 for(i=0;i<n;i++)
 printf("\t%d",t[i]+1);
 return 0;
}
```

RESULT: Output is obtained and the result is verified.

OUTPUT:

```
1. BFS
2. DFS
3. Topological Sort
4. Exit
Enter your choice: 1
Enter the number of vertices: 4

Enter graph data in matrix form:
0 1 0 1
0 0 1 0
0 0 0 0
0 0 1 0

Enter the starting vertex: 1
1   2   4   3
```

```
1. BFS
2. DFS
3. Topological Sort
4. Exit
Enter your choice: 2

Enter the number of vertices: 4

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

1->2
2->3
1->4

Graph is connected
```

```
1. BFS
2. DFS
3. Topological Sort
4. Exit
Enter your choice: 3

Enter the number of vertices: 4

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

The Topological Sequence is:    1   4   2   3
1. BFS
2. DFS
3. Topological Sort
4. Exit
Enter your choice: 4


=== Code Execution Successful ===
```

Experiment No: 10

Date: 02-12-2024

PRIM'S ALGORITHM

AIM: To perform minimum cost spanning tree using prim's algorithm

ALGORITHM:

1. Input:

Step 1.1: The number of nodes n in the graph.

Step 1.2: The adjacency matrix cost[][] where each element represents the cost of the edge between two nodes. If no edge exists, the value is set to 0 (which is replaced by a large number, typically 999 in your code).

2. Initialization:

Step 2.1: Mark the first node (node 1) as visited.

Step 2.2: Set the mincost variable to 0 to store the total weight of the MST.

Step 2.3: Set the variable ne (number of edges) to 1 (since the first edge will be added).

Step 2.4: Initialize the visited[] array to track the visited nodes, with visited[1] = 1 (first node is visited).

3. Process:

Step 3.1: Repeat the process until ne < n (i.e., until the number of edges in the MST is less than the number of nodes):

Step 3.2: For each node, if it is visited, check its adjacent nodes.

Step 3.3: Find the edge with the minimum cost from the already visited node to any unvisited node. This is the edge that should be added to the MST.

Step 3.4: Once the minimum cost edge is found, mark the node at the other end of the edge as visited and include it in the MST.

Step 3.5: Add the cost of this edge to the mincost and print the edge details (from node a to node b).

Step 3.6: Remove the edge from the cost matrix by setting its value to a very large number (like 999).

4. Termination:

Step 4.1: The algorithm stops when ne == n (the MST includes n-1 edges).

Step 4.2: Finally, print the total minimum cost (mincost) of the MST.

SOURCE CODE:

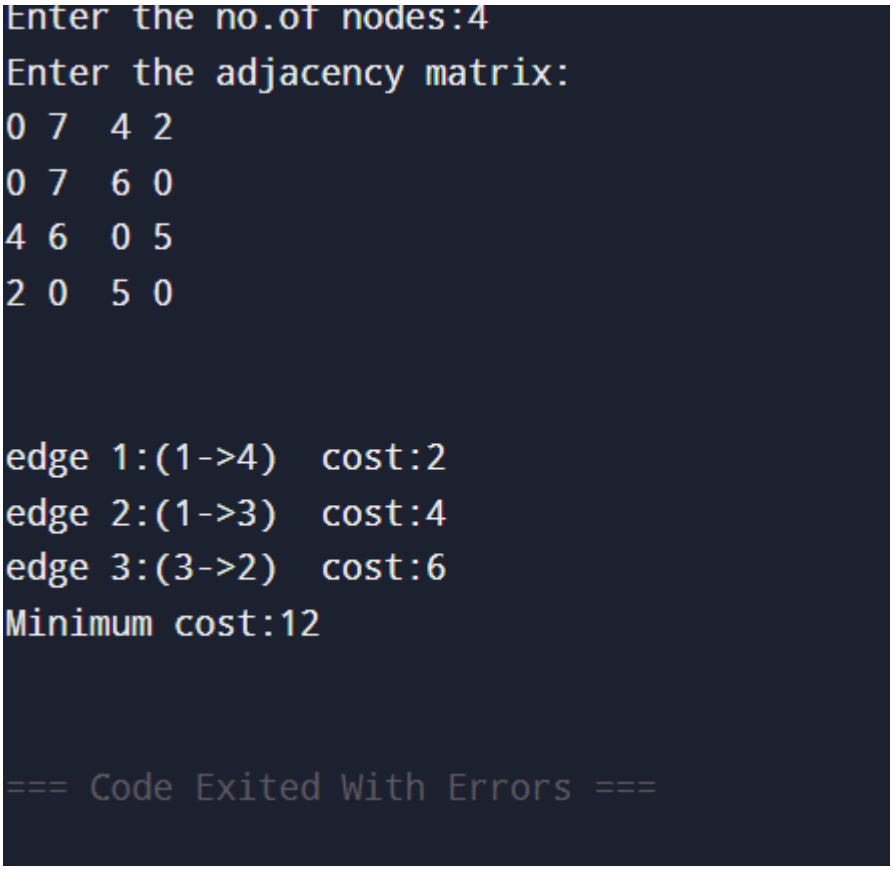
```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
 {
   clrscr();
```

```
printf("Enter the no.of nodes:");
scanf("%d",&n);
printf("Enter 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("\nedge %d:(%d->%d)  cost:%d",ne++,a,b,min);
                   mincost+=min;
                   visited[b]=1;
                 }
                 cost[a][b]=cost[b][a]=999;
    }
    printf("\nMinimum cost:%d\n",mincost);
    getch();
  }
```

RESULT: Output is obtained and the result is verified.

OUTPUT:


Enter the no.of nodes:4
Enter the adjacency matrix:
0 7 4 2
0 7 6 0
4 6 0 5
2 0 5 0
edge 1:(1->4) cost:2
edge 2:(1->3) cost:4
edge 3:(3->2) cost:6
Minimum cost:12
=== Code Exited With Errors ===

Experiment No: 11

Date: 09-12-2024

KRUSKAL'S ALGORITHM

AIM: To perform minimum cost spanning tree using Kruskal's algorithm

ALGORITHM:

Step 1: **Initialization**:All necessary variables (i, j, k, a, b, u, v, n, nedge, min, mincost, cost[9][9], parent[9]) are initialized.

Step 2: **Input the number of vertices (n)**: The user is prompted to enter the number of vertices in the graph.

Step 3: **Input the adjacency matrix**: The user is asked to input the adjacency matrix cost[i][j], and any 0 values (representing no edge between nodes) are replaced with 999 (infinity).

Step 4: **Initialize the parent array**: The parent[] array is initialized to 0, meaning that initially, each vertex is its own parent.

Step 5: **Start the MST**: The process of finding the minimum spanning tree begins. The algorithm will loop until n-1 edges are added to the MST.

Step 6: **Loop through edges**: While the number of edges in the MST (nedge) is less than n-1, the algorithm continues.

Step 7: **Find the minimum weight edge**: The algorithm scans the cost[][] matrix to find the edge with the smallest weight that has not yet been selected.

Step 8: **Cycle check**: The find() function is used to check if adding the selected edge

would form a cycle. If u and v belong to different sets, the edge is added to the MST.

Step 9: **Union operation**: If the edge (a, b) does not form a cycle, the uni() function joins the sets of u and v.

Step 10: **Remove the selected edge**: Once the edge is added to the MST, the corresponding entry in the adjacency matrix is set to 999 to avoid selecting it again.

Step 11: **Output the total minimum cost**: After the MST has been formed, the total minimum cost is displayed.

Step 11.1: **End program**: The program waits for user input before exiting.

Step 11.2: **Find function**: The find() function traces the parent array to find the root of a vertex. It returns the root of the set containing the vertex.

Step 11.3: **Union function**: The uni() function checks if two vertices belong to different sets. If they do, it unites the sets by making one vertex the parent of the other.

SOURCE CODE:

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

```
        clrscr();
        printf("Minimum Cost Spanning Tree\n");
        printf("\n\tImplemenetation of Kruskal's Algorithm\n");
        printf("\nEnter the number of vertices:");
        scanf("%d",&n);
        printf("\nEnter the cost using adjacency martix:\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 edge of Minimum Cost Spanning Tree are\n");
        while(nedge<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("Edge %d(%d->%d)=%d\n",nedge++,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: Output is obtained and the result is verified.

OUTPUT:

```
Minimum Cost Spanning Tree

    Implemenetation of Kruskal's Algorithm

Enter the number of vertices:4

Enter the cost using adjacency martix:
0 7 4 2
7 0 6 0
4 6 0 5
2 0 5 0
The edge of Minimum Cost Spanning Tree are
Edge 1(1->4)=2
Edge 2(1->3)=4
Edge 3(2->3)=6

    Minimum Cost=12


=== Code Exited With Errors ===
```

Experiment No: 12

Date: 17-12-2024

SINGLE SOURCE SHORTEST PATH : DIJKSTRA'S ALGORITHM

AIM: To find the single source shortest path algorithm.

ALGORITHM:

STEP 1 : Create a set shortPath to store vertices that come in the way of the shortest path tree.

STEP 2 : Initialize all distance values as INFINITE and assign distance values as 0 for source vertex so that it is picked first.

STEP 3 : Loop until all vertices of the graph are in the shortPath.

STEP 3.1 : Take a new vertex that is not visited and is nearest.

STEP 3.2 : Add this vertex to shortPath.

STEP 3.3 : For all adjacent vertices of this vertex update distances. Now check every adjacent vertex of V, if sum of distance of u and weight of edge is else then update it.

SOURCE CODE:

```
#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10
```

```
void dijkstra(int g[MAX][MAX],int n,int startnode);

void main()

{

        int g[MAX][MAX],i,j,n,u;

        clrscr();

        printf("\nSingle Shortest Path\n");

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

        scanf("%d",&n);

        printf("\nEnter the adjacenncy Matrix:\n");

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

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

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

        printf("\nEnter the starting vertex:");

        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: Output is obtained and the result is verified.

OUTPUT:

```
Single Shortest Path
Enter the number of vertices:5

Enter the adjacenncy Matrix:
0  10  30  50  100
10  0  50  0   100

0   50 0   20  10
30  0  20  0   60
100 0  10  60  0

Enter the starting vertex:0

Distance of node 1=10
Path=1<-0
Distance of node 2=30
Path=2<-0
Distance of node 3=50
Path=3<-0
Distance of node 4=40
Path=4<-2<-0

=== Code Exited With Errors ===
```