**Experiment 1 Date:16/10/2024**

**Advanced Use of GCC**

**Aim:**

Create appropriate C program. Compile and generate output using gcc command and its important options-o, -c, -D, -I, -g, -O, -save-temps

**About GCC Compiler**

GCC is a Linux-based c compiler released by the free software foundation which is usually operated via the command line. It often comes distributed freely with a Linux installation, so if you are running Unix or a Linux variant you will probably have it on your system. You can invoke gcc on a source code file simply by typing:-

# gcc filename

The default executable output of gcc is "a.out", which can be run by typing"./a.out". It is also possible to specify a name for the executable file at the command line by using the syntax " -o outputfile", as shown in the following example: -

# gcc filename -o outputfile

Again, you can run your program with "./outputfile". (the ./ is there to ensure to run the program for the current working directory.)

Note: if you need to use functions from the math library (generally functions from math.h" such as sin or sqrt), then you need to explicitly ask it to link with that library with the "-1" flag and the library "m":

# gcc filename -o outputfile -lm

**Program 1**

// Write a C program 'sum.c' to add two numbers. Read the input from Standard Input and write output to Standard output

#include<stdio.h>

void main()

{

int a,b ,sum;

printf("enter num 1:\n");

scanf("%d",&a);

printf("enter num 2:\n");

scanf("%d",&b);

sum=a+b;

printf("sum of %d and %d is %d\n",a,b,sum);

}

**Output**

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc sum.c mits@mits-H610M-H-V2-DDR4:~/gokul ds$ ./a.out enter num 1:

40

enter num 2:

20

sum of 40 and 20 is 60

**Important Options in GCC**

# Option: -o

To write and build output to output file.

**Output**

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc sum.c -o sum\_out

Here, GCC compiles the sum.c file and generates an executable named sum\_out.

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ ./sum\_out enter num 1:

40

enter num 2:

20

sum of 40 and 20 is 60

# Option: -save-temps

To save temporary files generated during program execution.

**Output**

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc -save-temps sum.c

This will generate intermediate files, like sum.i (pre-processed source) and sum.s (assembly code), in addition to the final executable.

# Option: -g

gcc -g generates debug information to be used by GDB debugger.

**Output**

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc -g sum.c -o sum\_out mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gdb sum\_out

GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1 Reading symbols from sum\_out...

(gdb) run

Starting program: /home/mits/gokul ds/sum\_out

enter num 1:

15

enter num 2:

10

sum of 15 and 10 is 25

[Inferior 1 (process 6667) exited normally]

(gdb) quit

This compiles sum.c with debug information, enabling you to debug the resulting executable file.

**Option: -c**

gcc -c compiles source files to object files without linking.When we compile a program the ‘C’ compiler will generate object files “.o” files. After that linker will generate a“.out” file. That means, it is a two step process; one step is to compile the program and another step is to link the object files and generate the executable file.

**Output**

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc -c sum.c mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc sum.o -o a\_out mits@mits-H610M-H-V2-DDR4:~/gokul ds$ ./a\_out

enter num 1:

20

enter num 2:

15

sum of 20 and 15 is 35

This will generate an object file sum.o that can be linked separately. This is the way to create an object file and use the functions in object files from different code modules.

**Program 2**

/\* Write a program with preprocessor directives. #ifdef is used to include a section of code if a certain macro is defined by #define.\*/

// myfile.c

#include <stdio.h>

//#define SAMPLE 10

void main()

{

#ifdef SAMPLE

printf("With preprocessor directive= %d\n",SAMPLE);

#else

printf("With out #ifdef\n");

#endif

}

**Important Option in GCC**

**Option: -D**

gcc -D defines a macro to be used by the preprocessor.

## Output

1. **Build *myfile.c* and run it with the macro, SAMPLE defined:**

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc -D SAMPLE myfile.c -o myfile

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ ./myfile With preprocessor directive

1. **Build *myfile.c* and run it without the macro, SAMPLE defined:** mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc myfile.c -o myfile mits@mits-H610M-H-V2-DDR4:~/gokul ds$ ./myfile

With out #ifdef

## Program 3

/\* Create a program with macro and saved this as header file. Create another C program which include this header file. \*/

// myheader.h

#define NUM1 5

save this file as **src/myheader.h**

// myfile.c

#include <stdio.h>

#include "myheader.h"

void main()

{

int num = NUM1; printf("num=%d\n", num);

}

save this file as myfile.c

**Important Option in GCC**

**Option: -I**

gcc -I include directory of header files. This flag is used to specify additional directories where header files are located. It helps the preprocessor find the necessary headers when compiling the code.

## Output

1. **Build *myfile.c* without include directory *src* :**

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc myfile1.c myfile1.c:3:18: fatal error: myheader.h: No such file or directory

3 | #include "myheader.h"

| ^~~~~~~~~~~~ compilation terminated.

1. **Build *myfile.c* with include directory *src* :**

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc -Isrc myfile1.c -o myfile\_out mits@mits-H610M-H-V2-DDR4:~/gokul ds$ ./myfile\_out

num=5

**Experiment 2 Date:22/10/2024**

**Familiarisation with GDB**

**Aim:**

Write a C program ‘mul.c’ to multipy two numbers. Read the input from Standard Input and write output to Standard output. Compile and generate output which is then debug with gdb and use the important commands break, run, next, print, display

**Program** #include<stdio.h> void main()

{

int a,b,mul;

printf("enter num 1:\n"); scanf("%d",&a); printf("enter num 2:\n"); scanf("%d",&b); mul=a\*b;

printf("product of %d and %d is %d\n",a,b,mul);

}

## Output

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc mul.c -o mul\_out

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ ./mul\_out

enter num 1:

8

enter num 2:

6

product of 8 and 6 is 48

# Important Commands in GDB

## option -g

gcc -g generates debug information to be used by GDB debugger.

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc -g mul.c -o mul\_out mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gdb mul\_out

## Output

GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1 Copyright (C) 2022 Free Software Foundation, Inc. License GPLv3+: GNU GPL version 3 or later

<<http://gnu.org/licenses/gpl.html>>

This is free software: you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law.

Type "show copying" and "show warranty" for details. This GDB was configured as "x86\_64-linux-gnu".

Type "show configuration" for configuration details. For bug reporting instructions, please see:

<[https://www.](http://www.gnu.org/software/gdb/bugs/)gnu.org[/software/gdb/bu](http://www.gnu.org/software/gdb/bugs/)g[s/](http://www.gnu.org/software/gdb/bugs/)>.

Find the GDB manual and other documentation resources online at:

<[http://www.gnu.org/software/gdb/documentation/>](http://www.gnu.org/software/gdb/documentation/).

For help, type "help".

Type "apropos word" to search for commands related to "word"...

Reading symbols from mul\_out....

(gdb)

## a. Command: run

Executes the program from start to end.

## Output

(gdb) run

Starting program: /home/mits/gokul ds/mul\_out [Thread debugging using libthread\_db enabled]

Using host libthread\_db library "/lib/x86\_64-linux-gnu/libthread\_db.so.1". enter num 1:

8

enter num 2:

6

product of 8 and 6 is 48

[Inferior 1 (process 4195) exited normally]

## b. Command: l - for Display the code

Type “l” at gdb prompt to display the code.

## Output

(gdb) l

1. #include<stdio.h>
2. void main()
3. {
4. int a,b,mul;
5. printf("enter num 1:\n");
6. scanf("%d",&a);
7. printf("enter num 2:\n");
8. scanf("%d",&b);
9. mul=a\*b;
10. printf("product of %d and %d is %d\n",a,b,mul);
11. }

## c. Command: break

Sets a breakpoint on a particular line.

## Output

(gdb) break mul.c:4

Breakpoint 1 at 0x5555555551c4: file mul.c, line 5. (gdb) run

Starting program: /home/mits/gokul ds/mul\_out [Thread debugging using libthread\_db enabled]

Using host libthread\_db library "/lib/x86\_64-linux-gnu/libthread\_db.so.1".

Breakpoint 1, main () at mul.c:5

1. printf("enter num 1:\n");

## d. Command: next

Executes the next line of code without diving into functions.

**Output (**gdb) next enter num 1:

1. scanf("%d",&a); (gdb) next

4

1. printf("enter num 2:\n");

## e. Command: clear

Clears all breakpoints.

## Output

(gdb) clear

No breakpoint at this line.

## f. Command: print

Displays the value of a variable.

## Output

(gdb) print a

$1 = 4

## g. Command: display

Displays the current values of the specified variable after every step.

## Output

(gdb) display a 1: a = 4

## h. Command: quit

Exits out of GDB.

## Output

(gdb) quit

A debugging session is active.

Inferior 1 [process 5092] will be killed.

Quit anyway? (y or n)

**Experiment 3 Date:23/10/2024**

**Familiarisation with gprof**

**Aim:**

Write a program for finding the sum of two numbers using a function. Then profile the executable with gprof.

**gprof**

Gprof is a profiling program which collects and arranges statistics on our programs.Basically, it looks into each of our functions and inserts code at the head and tail of each one to collect timing information. Then, when we run our program normally, it creates "gmon.out". In order to produce profiling information, the program must be compiled with specific options to tell it to record this information. The important options are: ‘-pg' (profile graph) and '-g' (debug information) must be included.

**Program**

#include<stdio.h>

int sum(int x, int y){

return x+y;

}

void main(){

int a,b;

printf("Enter 2 numbers : ");

scanf("%d %d",&a,&b);

printf("Sum : %d",sum(a,b));

}

**Output**

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc sum.c

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ ./a.out

Enter 2 numbers : 9 7

Sum : 16

**Option: -pg**

To compile a source file for profiling, specify the ‘-pg’ option when we run the compiler. -pg’, alters either compilation or linking to do what is necessary for profiling.

**Output**

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc -pg -o sum.out sum.c

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ ./sum.out

Enter 2 numbers : 8 6

Sum : 14

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gprof ./sum.out gmon.out > pgm3.txt

**pgm3.txt**

Flat profile:

Each sample counts as 0.01 seconds.

no time accumulated

% cumulative self self total

time seconds seconds calls Ts/call Ts/call name

0.00 0.00 0.00 1 0.00 0.00 sum

**Experiment 4 Date:29/10/2024**

**Different types of functions**

**Aim:**

Write a program for finding the sum of two numbers using different types of functions.

a) Function with argument and return type

b) Function with argument and no return type

c) Function without argument and return type

d) Function without argument and no return type

This program should have menu driven options.

**Algorithm:**

**main()**

1. Start
2. Declare ch, num1,num2
3. Display choices
4. Read option ch

if ch==1 input num1,num2 and call sum1(num1,num2)

and print the result from sum1()

if ch==2 input num1,num2 and call sum2(num1, num2)

if ch==3 call sum3() and print the result from sum3()

if ch==4 call sum4()

if ch==5 exit

1. Repeat steps 3 & 4 while ch>0&&ch<5
2. Stop.

**int sum1(int a, int b)**

1. Start
2. sum=a+b
3. Return sum
4. Exit.

**void sum2(int a, int b)**

1. Start
2. sum=a+b
3. Print sum
4. Exit

**int sum3()**

1. Start
2. Declare a and b
3. Read a and b
4. sum=a+b
5. Return sum
6. Exit

**void sum4()**

1. Start
2. Declare a and b
3. Read a and b
4. sum=a+b
5. Print sum
6. Exit

**Program**

#include<stdio.h>

int sum1(int,int);

void sum2(int,int);

int sum3();

void sum4();

void main()

{

int ch;

do{

printf("1.function with argument and return type\n");

printf("2.function with argument and no return type\n");

printf("3.function without argument and return type\n");

printf("4.function without argument and no return type\n");

printf("5.exit\n");

printf("choose an option\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

{

printf("using function with argument and return type\n");

int x,y,r;

printf("enter num 1\n");

scanf("%d",&x);

printf("enter num 2\n");

scanf("%d",&y);

r=sum1(x,y);

printf("sum of %d and %d is %d\n",x,y,r);

break;

}

case 2:

{

printf("using function with argument and no return type\n");

int x,y;

printf("enter num 1\n");

scanf("%d",&x);

printf("enter num 2\n");

scanf("%d",&y);

sum2(x,y);

break;

}

case 3:

{

printf("using function without argument and return type\n");

int r;

r=sum3();

printf("sum is %d\n",r);

break;

}

case 4:

{

printf("using function without argument and no return type\n");

sum4();

break;

}

case 5:printf("exit\n");

break;

default:printf("enter valid option\n");

}

}while(ch!=5);

}

int sum1(int a,int b)

{

int sum;

sum=a+b;

return sum;

}

void sum2(int a,int b)

{

int sum;

sum=a+b;

printf("sum of %d and %d is %d\n",a,b,sum);

}

int sum3()

{

int sum,x,y;

printf("enter num 1\n");

scanf("%d",&x);

printf("enter num 2\n");

scanf("%d",&y);

sum=x+y;

return sum;

}

void sum4()

{

int sum,x,y;

printf("enter num 1\n");

scanf("%d",&x);

printf("enter num 2\n");

scanf("%d",&y);

sum=x+y;

printf("sum is %d\n",sum);;

}

**Output**

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc functioneg.c

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ ./a.out

1.function with argument and return type

2.function with argument and no return type

3.function without argument and return type

4.function without argument and no return type

5.exit

choose an option

1

using function with argument and return type

enter num 1

6

enter num 2

8

sum of 6 and 8 is 14

1.function with argument and return type

2.function with argument and no return type

3.function without argument and return type

4.function without argument and no return type

5.exit

choose an option

2

using function with argument and no return type

enter num 1

8

enter num 2

6

sum of 8 and 6 is 14

1.function with argument and return type

2.function with argument and no return type

3.function without argument and return type

4.function without argument and no return type

5.exit

choose an option

3

using function without argument and return type

enter num 1

7

enter num 2

4

sum is 11

1.function with argument and return type

2.function with argument and no return type

3.function without argument and return type

4.function without argument and no return type

5.exit

choose an option

4

using function without argument and no return type

enter num 1

9

enter num 2

1

sum is 10

1.function with argument and return type

2.function with argument and no return type

3.function without argument and return type

4.function without argument and no return type

5.exit

choose an option

5

exit

**Experiment 5 Date:30/10/2024**

**Array Operations**

**Aim:**

To implement a menu driven program to perform following array operations

a) Insert an element to particular location

b) Delete an element to particular location

c) Traverse

**Algorithm:**

**main()**

1. Start

2 Declare a[20],i,n,ch

3 Enter limit n

4 for i=0 to n-1

Enter array a[i]

5 for i=0 to n-1

Print array a[i]

6 Display Choices

7 Read option ch.

If ch==1 call insert(a,n)

If ch==2 call delete(a,n)

If ch==3 call traverse(a,n)

If ch==4 exit

8 Repeat steps 6 & 7 while ch>0&&ch<4

9 Stop

**void insert(\*a,\*n)**

1 Start

2 Declare j,num,pos

3 Read pos and num

4 j=n-1

5 while j >= pos-1

a[j+1]=a[j]

j=j-1

6 a[pos-1]=num

7 n=n+1

8 Exit

**void delete(\*a,\*n)**

1 Start

2 Declare j,pos

3 Read pos

4 j=pos-1

5 while j < n-1

a[j]=a[j+1]

j=j+1

6 n=n-1

7 Exit

**void traverse(\*a,n)**

1 Start

2 Declare i

3 for i=0 to i<n

Print a[j]

i=i+1

4 Exit

**Program**

#include<stdio.h>

void insert(int \*,int \*);

void delete(int \*,int \*);

void traverse(int \*,int );

void main()

{

int a[20],i,n,ch;

printf("enter limit\n");

scanf("%d",&n);

printf("enter elements\n");

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

{

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

}

printf("array elements\n");

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

{

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

}

printf("\n");

do

{

printf("\n1.insert element at particular position\n2.delete element from particular position\n3.traverse\n4.exit\n");

printf("choose an operation\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

{

printf("insertion\n");

insert(a,&n);

break;

}

case 2:

{

printf("deletion\n");

delete(a,&n);

break;

}

case 3:

{

printf("traverse\n");

traverse(a,n);

break;

}

case 4:

{

printf("exit\n");

break;

}

default:printf("choose valid option\n");

break;

}

}while(ch!=4);

}

void insert (int \*a,int \*n)

{

int pos,num,j;

printf("enter position\n");

scanf("%d",&pos);

printf("enter number\n");

scanf("%d",&num);

if(\*n==20)

printf("array overflow\n");

else

{

j=\*n-1;

while(j>=pos-1)

{

a[j+1]=a[j];

j=j-1;

}

a[pos-1]=num;

\*n=\*n+1;

}

}

void delete (int \*a,int \*n)

{

int pos,j;

printf("enter position\n");

scanf("%d",&pos);

j=pos-1;

while(j<=\*n-1)

{

a[j]=a[j+1];

j=j+1;

}

\*n=\*n-1;

}

void traverse (int \*a,int n)

{

int i;

printf("array elements\n");

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

{

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

}

printf("\n");

}

**Output**

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ gcc array1.c

mits@mits-H610M-H-V2-DDR4:~/gokul ds$ ./a.out

enter limit

5

enter elements

2

4

6

8

10

array elements

2 4 6 8 10

1.insert element at particular position

2.delete element from particular position

3.traverse

4.exit

choose an operation

1

insertion

enter position

3

enter number

5

1.insert element at particular position

2.delete element from particular position

3.traverse

4.exit

choose an operation

3

traverse

array elements

2 4 5 6 8 10

1.insert element at particular position

2.delete element from particular position

3.traverse

4.exit

choose an operation

2

deletion

enter position

4

1.insert element at particular position

2.delete element from particular position

3.traverse

4.exit

choose an operation

3

traverse

array elements

2 4 5 8 10

1.insert element at particular position

2.delete element from particular position

3.traverse

4.exit

choose an operation

4

exit

**Experiment 6 Date:05/11/2024**

**Array Sorting**

**Aim:**

Program to sort an integer array.

**Algorithm:**

**main()**

1. Start

2 Declare a[20],i,n,j,temp

3 Enter limit n

4 for i=0 to n-1

Enter array a[i]

5 for i=0 to n-1

Print array a[i]

6 for i=0 to n-1

for j=0 to j-i-1

if a[j] > a[j+1]

temp=a[j]

a[j]=a[j+1]

temp=a[j+1]

7 for i=0 to n-1

Print array a[i]

8 Stop

**Program**

#include<stdio.h>void main(){int a[30],i,n,j,temp;printf("enter limit:\n");scanf("%d",&n);printf("enter elements:\n");for(i=0;i<n;i++){scanf("%d",&a[i]);}printf("array elements:\n");for(i=0;i<n;i++){printf("%d\t",a[i]);}printf("\n");for(i=0;i<n-1;i++){for(j=0;j<n-i-1;j++){if(a[j]>a[j+1]){temp=a[j];a[j]=a[j+1];a[j+1]=temp;}}}printf("sorted array:\n");for(i=0;i<n;i++){printf("%d\t",a[i]);}printf("\n");}

**Output**

enter limit:6enter elements:12815379array elements:12 8 15 3 7 9sorted array:3 7 8 9 12 15

**Experiment 7 Date:05/11/2024**

**Array Searching**

**Aim:**

Program to implement linear search and binary search.

**Algorithm:**

**main()**

1. Start

2 Declare a[20],i,n,ch

3 Enter limit n

4 for i=0 to n-1

Enter array a[i]

5 for i=0 to n-1

Print array a[i]

6 Display Choices.

7 Read option ch.

If ch==1 call lsearch(a,n)

If ch==2 call bsearch(a,n)

If ch==3 exit

8 Repeat steps 6 & 7 while ch>0&&ch<3

9 Stop

**void lsearch(\*a,n)**

1 Start

2 Declare i,num,f=0

3 Read num

4 for i=0 to n-1

if num==a[i]

f=1

break

5 if f==0

Print num found at position i+1

else

Print num not found

6 Exit

**void bsearch(\*a,n)**

1 Start

2 Declare pos,ub,lb,mid,num,i,j,temp

3 Read num

4 for i=0 to n-1

for j=0 to j-i-1

if a[j] > a[j+1]

temp=a[j]

a[j]=a[j+1]

temp=a[j+1]

5 for i=0 to n-1

Print array a[i]

6 Set pos=-1,lb=0,ub=n-1

7 while lb < ub

mid=lb+ub/2

if num == a[mid]

pos=mid

else if num < a[mid]

ub=mid-1

else

lb=mid+1

8 if pos==-1

Print num not found

else

Print num found at position pos+1

9 Exit

**Program**

#include<stdio.h>void lsearch(int \*,int);void bsearch(int \*,int);void main(){int a[30],i,n,ch;printf("enter limit of array:\n");scanf("%d",&n);printf("enter array elements:\n");for(i=0;i<n;i++){scanf("%d",&a[i]); }printf("array elements:\n");for(i=0;i<n;i++){printf("%d\t",a[i]);}printf("\n");do{printf("\n1.using linear search\n2.using binary search\n3.exit\n");printf("choose an operation:\n");scanf("%d",&ch);switch(ch){case 1:lsearch(a,n);break;case 2:bsearch(a,n);break;case 3:printf("exit\n");break;default:printf("enter correct value\n");break;}}while(ch!=3);}void lsearch(int \*a,int n){int i,f=0,num;printf("enter element to search:\n");scanf("%d",&num);for(i=0;i<n;i++){if(num==a[i]){f=1;break;}}if(f==1)printf("element %d found at position %d\n",num,i+1);elseprintf("element not found\n");}void bsearch(int \*a,int n){int pos,ub,lb,mid,num,i,j,temp;printf("enter element to search:\n");scanf("%d",&num);for(i=0;i<n-1;i++){for(j=0;j<n-i-1;j++){if(a[j]>a[j+1]){temp=a[j];a[j]=a[j+1];a[j+1]=temp;}}}printf("sorted array:\n");for(i=0;i<n;i++){printf("%d\t",a[i]);}printf("\n");pos=-1;lb=0;ub=n-1;while(lb<=ub){mid=(lb+ub)/2;if(num==a[mid]){pos=mid;break;}else if(num<a[mid]){ub=mid-1;}else{lb=mid+1;}}if(pos==-1)printf("element not found\n");elseprintf("element %d found at position %d\n",num,pos+1);}

**Output**

enter limit of array:6enter array elements:894215array elements:8 9 4 2 1 51.using linear search2.using binary search3.exitchoose an operation:1enter element to search:4element 4 found at position 31.using linear search2.using binary search3.exitchoose an operation:1enter element to search:10element not found1.using linear search2.using binary search3.exitchoose an operation:2enter element to search:9sorted array:1 2 4 5 8 9element 9 found at position 6 1.using linear search2.using binary search3.exitchoose an operation:3exit

**Experiment 8 Date:07/11/2024**

**Matrix Operations**

**Aim:**

Perform addition, subtraction and multiplication of two matrices using switch.

**Algorithm:**

**main()**

1 Start

2 Declare a[20][20],b[20][20],i,j,m,n,o,p,ch

3 Read m,n

4 for i=0 to m-1

for j=0 to n-1

Enter matrix a[i][j]

5 for i=0 to m-1

for j=0 to n-1

Print matrix a[i][j]

6 for i=0 to o-1

for j=0 to p-1

Enter matrix b[i][j]

7 for i=0 to o-1

for j=0 to -p1

Print matrix b[i][j]

8 Display Choices.

9 Read option ch.

If ch==1 call addmatrix(a,b,m,n,o,p)

If ch==2 call submatrix(a,b,m,n,o,p)

If ch==3 call mulmatrix(a,b,m,n,o,p)

If ch==4 exit

10 Repeat steps 8 & 9 while ch>0&&ch<4.

11 Stop.

**void addmatrix(a[20][20],b[20][20],m,n,o,p)**

1 Start

2 Declare c[20][20],i,j

3 if m==o and n==p

for i=0 to m-1

for j=0 to o-1

c[i][j]=a[i][j]+b[i][j]

for i=0 to m-1

for j=0 to n-1

Print c[i][j]

else

Print Addition Cannot Be Performed.

4 Exit

**void submatrix(a[20][20],b[20][20],m,n,o,p)**

1 Start

2 Declare c[20][20],i,j

3 if m==o and n==p

for i=0 to m-1

for j=0 to o-1

c[i][j]=a[i][j]-b[i][j]

for i=0 to m-1

for j=0 to n-1

Print c[i][j]

else

Print Substraction Cannot Be Performed.

4 Exit

**void mulmatrix(a[20][20],b[20][20],m,n,o,p)**

1 Start

2 Declare c[20][20],i,j,k

3 if n==o

for i=0 to m-1

for j=0 to o-1

c[i][j]=0

for k=0 to n-1

c[i][j]=c[i][j]+a[i][k]\*b[k][j]

for i=0 to m-1

for j=0 to p-1

Print c[i][j]

else

Print Multiplication Cannot Be Performed.

4 Exit

**Program**

#include<stdio.h>void addmatrix(int a[20][20],int b[20][20],int m,int n,int o,int p);void submatrix(int a[20][20],int b[20][20],int m,int n,int o,int p);void mulmatrix(int a[20][20],int b[20][20],int m,int n,int o,int p);void main(){int a[20][20],b[20][20],i,j,m,n,o,p,ch;printf("enter no of rows and columns of matrix 1:\n");scanf("%d%d",&m,&n);printf("enter matrix 1:\n");for(i=0;i<m;i++){for(j=0;j<n;j++){scanf("%d",&a[i][j]);}}printf("matrix 1:\n");for(i=0;i<m;i++){for(j=0;j<n;j++){printf("%d\t",a[i][j]);}printf("\n");}printf("\n");printf("enter no of rows and columns of matrix 2:\n");scanf("%d%d",&o,&p);printf("enter matrix 2:\n");for(i=0;i<o;i++){for(j=0;j<p;j++){scanf("%d",&b[i][j]);}}printf("matrix 2:\n");for(i=0;i<o;i++){for(j=0;j<p;j++){printf("%d\t",b[i][j]);}printf("\n");}printf("\n");do{printf("\n1.matrixaddition\n2.matrixsubstraction\n3.matrix multiplication\n4.exit\n");printf("choose an operation:\n");scanf("%d",&ch);switch(ch){case 1:addmatrix(a,b,m,n,o,p);break;case 2:submatrix(a,b,m,n,o,p);break;case 3:mulmatrix(a,b,m,n,o,p);break;case 4:printf("exit\n");break;default:printf("enter correct value\n");break;}}while(ch!=4);}void addmatrix(int a[20][20],int b[20][20],int m,int n,int o,int p){int c[20][20],i,j;if(m==o && n==p){for(i=0;i<m;i++){for(j=0;j<o;j++){c[i][j]=a[i][j]+b[i][j];}}printf("after addition:\n");for(i=0;i<m;i++){for(j=0;j<n;j++){printf("%d\t",c[i][j]);}printf("\n");}}elseprintf("addition cannot be performed\n");}void submatrix(int a[20][20],int b[20][20],int m,int n,int o,int p){int c[20][20],i,j;if(m==o && n==p){for(i=0;i<m;i++){for(j=0;j<o;j++){c[i][j]=a[i][j]-b[i][j];}}printf("after substraction:\n");for(i=0;i<m;i++){for(j=0;j<n;j++){printf("%d\t",c[i][j]);}printf("\n");}}elseprintf("substraction cannot be performed\n");}void mulmatrix(int a[20][20],int b[20][20],int m,int n,int o,int p){int c[20][20],i,j,k;if(n==o){for(i=0;i<m;i++){for(j=0;j<o;j++){c[i][j]=0;for(k=0;k<n;k++){c[i][j]=c[i][j]+a[i][k]\*b[k][j];}}}printf("after multiplication:\n");for(i=0;i<m;i++){for(j=0;j<p;j++){printf("%d\t",c[i][j]);}printf("\n");}}elseprintf("multiplication cannot be performed\n");}

**Output**

enter no of rows and columns of matrix 1:22enter matrix 1:841216matrix 1:8 412 16enter no of rows and columns of matrix 2:22enter matrix 2:2247matrix 2:2 24 71.matrix addition2.matrix substraction3.matrix multiplication4.exitchoose an operation:1after addition:10 616 231.matrix addition2.matrix substraction3.matrix multiplication4.exitchoose an operation:2after substraction:6 28 9

1.matrix addition2.matrix substraction3.matrix multiplication4.exitchoose an operation:3after multiplication:32 4488 1361.matrix addition2.matrix substraction3.matrix multiplication4.exitchoose an operation:4exit

**Experiment 9 Date:07/11/2024**

**Stack Using Arrays**

**Aim:**

Program to implement stack operations using arrays.

**Algorithm:**

**main()**

1. Start

2 Initialise size=10,top=-1

3 Declare stack[10],ch

4 Display Choices.

5 Read option ch.

If ch==1 call push(stack)

If ch==2 call pop(stack)

If ch==3 call display(stack)

If ch==4 exit

6 Repeat steps 4 & 5 while ch>0&&ch<4.

7 Stop

**void push(\*stack)**

1 Start

2 Declare num

3 if top==size-1

Print Stack Overflow

else

Read num

top=top+1

stack[top]=num

4 Exit

**void pop(\*stack)**

1 Start

2 if top < 0

Print Stack Empty

else

top=top-1

3 Exit

**void display(\*stack)**

1 Start

2 Declare i

3 if top < 0

Print Stack Empty

else

for i=top to 0,i- -

Print stack[i]

4 Exit

**Program**

#include<stdio.h>void push(int \*);void pop(int \*);void display(int \*);int size=10;int top=-1;void main(){int stack[20],ch;do{printf("Stack Operations\n1.Push\n2.Pop\n3.Display\n4.Exit\n");printf("choose an operation:\n");scanf("%d",&ch);switch(ch){case 1:push(stack);break;case 2:pop(stack);break;case 3:display(stack);break;case 4:printf("exit\n");break;default:printf("enter correct value\n");break;}}while(ch!=4);}void push(int \*stack){int num;if(top==size-1){printf("stack overflow\n");}else{printf("enter number\n");scanf("%d",&num);top=top+1;stack[top]=num;printf("value pushed\n");}}void pop(int \*stack){if(top<0){printf("stack empty\n");}else{top=top-1;printf("value poped\n");}}void display(int \*stack){int i; if(top<0){printf("stack empty\n");}else{printf("stack elements:\n");for(i=top;i>=0;i--){printf("%d",stack[i]);printf("\n");}}}

**Output**

Stack Operations1.Push2.Pop3.Display4.Exitchoose an operation:1enter number8value pushedStack Operations1.Push2.Pop3.Display4.Exitchoose an operation:1enter number7value pushedStack Operations1.Push2.Pop3.Display4.Exitchoose an operation:1enter number6value pushedStack Operations1.Push2.Pop3.Display4.Exitchoose an operation:1enter number5value pushedStack Operations1.Push2.Pop3.Display4.Exitchoose an operation:1enter number4value pushedStack Operations1.Push2.Pop3.Display4.Exitchoose an operation:3stack elements:45678Stack Operations1.Push2.Pop3.Display4.Exitchoose an operation:2value popedStack Operations1.Push2.Pop3.Display4.Exitchoose an operation:2value popedStack Operations1.Push2.Pop3.Display4.Exitchoose an operation:3stack elements:678Stack Operations1.Push2.Pop3.Display4.Exitchoose an operation:4exit

**Experiment 10 Date:18/11/2024**

**Queue Using Arrays**

**Aim:**

Program to implement queue operations using arrays.

**Algorithm:**

**main()**

1. Start

2 Initialise size=5,front=-1,rear=-1

3 Declare queue[10],ch

4 Display Choices.

5 Read option ch.

If ch==1 call enqueue(queue)

If ch==2 call dequeue(queue)

If ch==3 call display(queue)

If ch==4 exit

6 Repeat steps 4 & 5 while ch>0&&ch<4.

7 Stop

**void enqueue(\*queue)**

1 Start

2 Declare item

3 if rear==size-1

Print Queue Overflow

else

Read item

if front==-1 && rear==-1

front=rear=0

else

rear=rear+1

queue[rear]=item

4 Exit

**void dequeue(\*queue)**

1 Start

2 if front==-1 && rear==-1

Print Queue Underflow

else

if front==rear

front=rear-1

else

front=front+1

3 Exit

**void display(\*queue)**

1 Start

2 Declare i

3 if rear==-1

Print Queue Empty

else

for i=front to rear ,i++

Print queue[i]

4 Exit

**Program**

#include<stdio.h>void enqueue(int \*);void dequeue(int \*);void display(int \*);int size=5;int front=-1,rear=-1;void main(){int queue[20],ch;do{printf("Queue Operations\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");printf("choose an operation:\n");scanf("%d",&ch);switch(ch){case 1:enqueue(queue);break;case 2:dequeue(queue);break;case 3:display(queue);break;case 4:printf("exit\n");break;default:printf("enter correct value\n");break;}}while(ch!=4);}void enqueue(int \*queue){int item;if(rear==size-1){printf("Queue Overflow\n");}

else {printf("enter item:\n");scanf("%d",&item);if(front==-1 && rear==-1){front=rear=0;}else{rear=rear+1;}queue[rear]=item;printf("value inserted\n");}printf("\n");}void dequeue(int \*queue){if(front==-1 && rear==-1){printf("Queue underflow\n");}else {if(front==rear){front=rear=-1;}else{front=front+1;}printf("value deleted\n");}printf("\n");}void display(int \*queue){int i; if(rear==-1){printf("queue empty\n");}else{printf("queue elements:\n");for(i=front;i<=rear;i++){printf("%d\t",queue[i]);}}printf("\n");}

**Output**

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter item:

8

value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter item:

7

value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

queue elements:

8 7

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter item:

6

value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter item:

5

value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

queue elements:

8 7 6 5

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter item:

9

value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

Queue Overflow

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

queue elements:

8 7 6 5 9

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

2

value deleted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

queue elements:

7 6 5 9

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

2

value deleted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

queue elements:

6 5 9

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

4

exit

**Experiment 11 Date:18/11/2024**

**Circular Queue Using Arrays**

**Aim:**

Program to implement circular queue using array.

**Algorithm:**

**main()**

1. Start

2 Initialise size=5,front=-1,rear=-1,count=0

3 Declare queue[10],ch

4 Display Choices.

5 Read option ch.

If ch==1 call enqueue(queue)

If ch==2 call dequeue(queue)

If ch==3 call display(queue)

If ch==4 exit

6 Repeat steps 4 & 5 while ch>0&&ch<4.

7 Stop

**void enqueue(\*queue)**

1 Start

2 Declare item

3 if count==size

Print Queue Overflow

else

Read item

if front==-1 && rear==-1

front=rear=0

else

rear=(rear+1)%size

queue[rear] = item

count=count+1

4 Exit

**void dequeue(\*queue)**

1 Start

2 if count==0

Print Queue Underflow

else

if front==rear

front=rear-1

else

front=(front+1)%size

count=count-1

3 Exit

**void display(\*queue)**

1 Start

2 Declare i

3 if count==0

Print Queue Empty

else

i=front

while i !=rear

print queue[i]

i = (i+1) % size

print queue[rear]

4 Exit

**Program**

#include<stdio.h>void enqueue(int \*);void dequeue(int \*);void display(int \*);int size=5;int front=-1,rear=-1,count=0;void main(){int queue[20],ch;do{printf("Queue Operations\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");printf("choose an operation:\n");scanf("%d",&ch);switch(ch){case 1:enqueue(queue);break;case 2:dequeue(queue);break;case 3:display(queue);break;case 4:printf("exit\n");break;default:printf("enter correct value\n\n");break;}}while(ch!=4);}void enqueue(int \*queue){int item;if(count==size){printf("Queue Overflow\n");}else {printf("enter item:\n");scanf("%d",&item);if(front==-1 && rear==-1){front=rear=0;}elseif(front==-1&&rear==-1) {front=rear=0;} else {rear=(rear+1)%size;}queue[rear] = item;count=count+1;printf("Value inserted\n");}printf("\n");}void dequeue(int \*queue){if(count==0){printf("Queue underflow\n");}else {if(front==rear){front=rear=-1;}else{front=(front+1)%size;}count=count-1;printf("value deleted\n");}printf("\n");}

void display(int \*queue){int i; if(count==0){printf("queue empty\n");}else{printf("queue elements:\n");int i=front;while(i!=rear) {printf("%d\t", queue[i]);i = (i+1) % size;}printf("%d\n", queue[rear]);}printf("\n");}

**Output**

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter item:

8

Value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter item:

7

Value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

queue elements:

8 7

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter item:

6

Value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter item:

5

Value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

queue elements:

8 7 6 5

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter item:

4

Value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

Queue Overflow

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

queue elements:

8 7 6 5 4

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

2

value deleted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

2

value deleted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

queue elements:

6 5 4

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter item:

8

Value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter item:

7

Value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

queue elements:

6 5 4 8 7

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

4

exit

**Experiment 12 Date:20/11/2024**

**Singly Linked List-Insertion**

**Aim:**

To implement the following operations on a singly linked list

i. Creation

ii. Insert a new node at front

iii. Insert an element after a particular node

iv. Insert a new node at end

v. Searching

vi. Traversal.

**Algorithm:**

**Declare the Structure node**

struct node

1.declare data, struct node\* next

**main()**

1. Start

2 Declare ch,struct node\* head = NULL

3 call createnode(head)

4 Display Choices.

5 Read option ch.

If ch==1 call insertatfront(head)

If ch==2 call insertafterkey(head)

If ch==3 call insertatlast(head)

If ch==4 call valuesearch(head)

If ch==5 call traverse(head)

If ch==6 exit

6 Repeat steps 4 & 5 while ch>0&&ch<6

7 Stop

**struct node\* createnode(struct node\* head)**

1 Start

2 Declare n,value,struct node\* p

3 Read n

4 if n <= 0

Print Size Must Be Greater Than Zero

5 for i=1 to n

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

Read value

temp->data=value

temp->next=NULL

if head==NULL

head=temp

else

p=head

while p->next != NULL

p=p->next

p->next=temp

6 Return head

7 Exit

**struct node\* insertatfront(struct node\* head)**

1 Start

2 Declare value

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

4 Read value

5 newnode->data=value

6 newnode->next=NULL

7 if head==NULL

newnode->next=NULL

head=newnode

else

newnode->next=head;

head=newnode;

8 Return head

9 Exit

**struct node\* insertafterkey(struct node\* head)**

1 Start

2 Declare value,key,struct node\* ptr

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

4 Read value

5 newnode->data=value

6 newnode->next=NULL

7 Read key

8 if head==NULL

newnode->next=NULL

head=newnode

else

ptr=head

while ptr != NULL

if ptr->data == key

newnode->next=ptr->next;

ptr->next=newnode;

else

ptr=ptr->next

if ptr==NULL

Print Node With Key Not Exist

9 Return head

10 Exit

**struct node\* insertatlast(struct node\* head)**

1 Start

2 Declare value,struct node\* ptr

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

4 Read value

5 newnode->data=value

6 newnode->next=NULL

7 if head==NULL

newnode->next=NULL

head=newnode

else

ptr=head

while ptr->next != NULL

ptr=ptr->next

8 ptr->next=newnode

9 Return head

10 Exit

**void valuesearch(struct node\* head)**

1 Start

2 Declare value,flag=0,pos=1,struct node\* ptr

3 Read value

4 ptr=head

5 while ptr != NULL

if ptr->data==value

flag=1

Print Value Present At Position pos

else

ptr=ptr->next

pos=pos+1

6 if flag==0

Print Value Not Found

7 Exit

**void traverse(struct node\* head)**

1 Start

2 Declare struct node\* ptr

3 ptr=head

4 if head==NULL

Print List Empty

else

while ptr != NULL

Print ptr->data

ptr=ptr->next

Print Null

5 Exit

**Program**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*next;

};

struct node\* createnode(struct node\* head){

struct node \*p;

int value,n;

printf("enter size \n");

scanf("%d",&n);

if(n <= 0) {

printf("List size must be greater than 0.\n");

return 0;

}

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

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

printf("enter value to insert\n");

scanf("%d",&value);

temp->data=value;

temp->next=NULL;

if (head==NULL){

head=temp;

}

else{

p=head;

while(p->next!=NULL){

p=p->next;

}

p->next=temp;

}

}

return head;

}

struct node\* insertatfront(struct node\* head);

struct node\* insertatlast(struct node\* head);

struct node\* insertafterkey(struct node\* head);

void traverse(struct node\* head);

void valuesearch(struct node\* head);

void main()

{

int ch,data,pos,val;

struct node\* head = NULL;

printf("Creating a linked list:\n");

head=createnode(head);

do

{

printf("Linked List Operations\n1.Insert Node At Front\n2.Insert After Particular Node\n3.Insert Node At Last\n4.Searching\n5.Traversal\n6.Exit\n");

printf("choose an operation:\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

{

head=insertatfront(head);

printf("value inserted at front\n");

break;

}

case 2:

{

head=insertafterkey(head);

break;

}

case 3:

{

head=insertatlast(head);

printf("value inserted at last\n");

break;

}

case 4:

{

printf("searching\n");

valuesearch(head);

break;

}

case 5:

{

printf("traversing\n");

traverse(head);

break;

}

case 6:printf("exit\n");

break;

default:printf("enter correct value\n\n");

break;

}

}while(ch!=6);

}

struct node\* insertatfront(struct node\* head)

{

int value;

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

printf("enter value to insert\n");

scanf("%d",&value);

newnode->data=value;

newnode->next=NULL;

if(head==NULL){

newnode->next=NULL;

head=newnode;

}

else{

newnode->next=head;

head=newnode;

}

return head;

}

struct node\* insertatlast(struct node\* head)

{

int value;

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

printf("enter value to insert\n");

scanf("%d",&value);

newnode->data=value;

newnode->next=NULL;

struct node \*ptr;

if(head==NULL){

newnode->next=NULL;

head=newnode;

}

else{

ptr=head;

while(ptr->next!=NULL){

ptr=ptr->next;

}

}

ptr->next=newnode;

return head;

}

struct node\* insertafterkey(struct node\* head)

{

int value,key;

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

printf("enter value to insert\n");

scanf("%d",&value);

newnode->data=value;

newnode->next=NULL;

printf("enter key\n");

scanf("%d",&key);

struct node \*ptr;

if(head==NULL){

newnode->next=NULL;

head=newnode;

}

else{

ptr=head;

while (ptr != NULL){

if (ptr->data == key){

newnode->next=ptr->next;

ptr->next=newnode;

printf("Node inserted after key\n");

return head;

}

else{

ptr = ptr->next;

}

}

if (ptr == NULL){

printf("Node with key does not exist\n");

}

}

return head;

}

void traverse(struct node\* head)

{

struct node \*ptr;

ptr=head;

if(head==NULL){

printf("list empty\n");

}

else{

while(ptr!=NULL){

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

ptr=ptr->next;

}

printf("NULL\n");

}

}

void valuesearch(struct node\* head)

{

int val;

printf("enter value to search\n");

scanf("%d",&val);

struct node \*ptr;

ptr=head;

int flag=0,pos=1;

while(ptr!=NULL){

if(ptr->data==val){

flag=1;

printf("Item Present at position %d\n",pos);

break;

}

else{

ptr=ptr->next;

pos=pos+1;

}

}

if(flag==0){

printf("Item Not Found\n");

}

}

**Output**

Creating a linked list:

enter size

5

enter value to insert

4

enter value to insert

5

enter value to insert

6

enter value to insert

7

enter value to insert

8

Linked List Operations

1.Insert Node At Front

2.Insert After Particular Node

3.Insert Node At Last

4.Searching

5.Traversal

6.Exit

choose an operation:

5

traversing

4->5->6->7->8->NULL

Linked List Operations

1.Insert Node At Front

2.Insert After Particular Node

3.Insert Node At Last

4.Searching

5.Traversal

6.Exit

choose an operation:

1

enter value to insert

3

value inserted at front

Linked List Operations

1.Insert Node At Front

2.Insert After Particular Node

3.Insert Node At Last

4.Searching

5.Traversal

6.Exit

choose an operation:

5

traversing

3->4->5->6->7->8->NULL

Linked List Operations

1.Insert Node At Front

2.Insert After Particular Node

3.Insert Node At Last

4.Searching

5.Traversal

6.Exit

choose an operation:

3

enter value to insert

9

value inserted at last

Linked List Operations

1.Insert Node At Front

2.Insert After Particular Node

3.Insert Node At Last

4.Searching

5.Traversal

6.Exit

choose an operation:

5

traversing

3->4->5->6->7->8->9->NULL

Linked List Operations

1.Insert Node At Front

2.Insert After Particular Node

3.Insert Node At Last

4.Searching

5.Traversal

6.Exit

choose an operation:

2

enter value to insert

7

enter key

5

Node inserted after key

Linked List Operations

1.Insert Node At Front

2.Insert After Particular Node

3.Insert Node At Last

4.Searching

5.Traversal

6.Exit

choose an operation:

5

traversing

3->4->5->7->6->7->8->9->NULL

Linked List Operations

1.Insert Node At Front

2.Insert After Particular Node

3.Insert Node At Last

4.Searching

5.Traversal

6.Exit

choose an operation:

2

enter value to insert

10

enter key

10

Node with key does not exist

Linked List Operations

1.Insert Node At Front

2.Insert After Particular Node

3.Insert Node At Last

4.Searching

5.Traversal

6.Exit

choose an operation:

4

searching

enter value to search

8

Item Present at position 7

Linked List Operations

1.Insert Node At Front

2.Insert After Particular Node

3.Insert Node At Last

4.Searching

5.Traversal

6.Exit

choose an operation:

4

searching

enter value to search

12

Item Not Found

Linked List Operations

1.Insert Node At Front

2.Insert After Particular Node

3.Insert Node At Last

4.Searching

5.Traversal

6.Exit

choose an operation:

6

exit

**Experiment 13 Date:02/12/2024**

**Singly Linked List-Deletion**

**Aim:**

To implement the following operations on a singly linked list

i. Creation

ii. Deletion from beginning

iii. Deletion from the end

iv. Deletion from particular location

v. Traversal.

**Algorithm:**

**Declare the Structure node**

struct node

1.declare data, struct node\* next

**main()**

1. Start

2 Declare ch,struct node\* head = NULL

3 call createnode(head)

4 Display Choices.

5 Read option ch.

If ch==1 call deleteatfront(head)

If ch==2 call deleteatlast(head)

If ch==3 call deleteatpos(head)

If ch==4 call traverse(head)

If ch==5 exit

6 Repeat steps 4 & 5 while ch>0&&ch<5

7 Stop

**struct node\* createnode(struct node\* head)**

1 Start

2 Declare n,value,struct node\* p

3 Read n

4 if n <= 0

Print Size Must Be Greater Than Zero

5 for i=1 to n

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

Read value

temp->data=value

temp->next=NULL

if head==NULL

head=temp

else

p=head

while p->next != NULL

p=p->next

p->next=temp

6 Return head

7 Exit

**struct node\* deleteatfront(struct node\* head)**

1 Start

2 Declare struct node\* ptr

3 if head==NULL

Print Linked List Underflow

else

ptr=head

head=ptr->next

free(ptr)

4 Return head

5 Exit

**struct node\* deleteatlast(struct node\* head)**

1 Start

2 Decalre struct node\* ptr,struct node\* ptr1

3 if head==NULL

Print Linked List Underflow

else

if ptr->next == NULL

head=NULL

free(head)

else

ptr=head

while ptr->next != NULL

ptr1=ptr

ptr=ptr->next

ptr1->next=NULL

free(ptr)

4 Return head

5 Exit

**struct node\* deleteatpos(struct node\* head)**

1 Start

2 Declare i,pos,struct node\* ptr,struct node\* ptr1

3 Read pos

4 if head==NULL

Print Linked List Underflow

5 if pos==1

ptr=head

head=ptr->next

free(ptr)

else

ptr=head

for i=1 to i<pos-1 && ptr!==NULL,i++

ptr=ptr->next

if ptr==NULL||ptr->next==NULL

Print Position Out Of Range

ptr1 = ptr->next;

ptr->next = ptr1->next;

free(ptr1);

6 Return head

7 Exit

**void traverse(struct node\* head)**

1 Start

2 Declare struct node \*ptr

3 ptr=head

4 if head==NULL

Print List Empty

else

while ptr != NULL

Print ptr->data

ptr=ptr->next

Print Null

5 Exit

**Program**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*next;

};

struct node\* createnode(struct node\* head){

struct node \*p;

int value,n;

printf("enter size \n");

scanf("%d",&n);

if(n <= 0) {

printf("List size must be greater than 0.\n");

return 0;

}

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

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

printf("enter value to insert\n");

scanf("%d",&value);

temp->data=value;

temp->next=NULL;

if (head==NULL){

head=temp;

}

else{

p=head;

while(p->next!=NULL){

p=p->next;

}

p->next=temp;

}

}

return head;

}

struct node\* deleteatfront(struct node\* head);

struct node\* deleteatlast(struct node\* head);

struct node\* deleteatpos(struct node\* head);

void traverse(struct node\* head);

void main()

{

int ch,data,pos,val;

struct node \*head=NULL;

printf("Creating a linked list:\n");

head=createnode(head);

do

{

printf("Linked List Operations\n1.Delete From Front\n2.Delete From Last\n3.Delete From Particular Position\n4.Traversal\n5.Exit\n");

printf("choose an operation:\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

{

head=deleteatfront(head);

printf("value deleted from front\n");

break;

}

case 2:

{

head=deleteatlast(head);

printf("value deleted from last\n");

break;

}

case 3:

{

head=deleteatpos(head);

break;

}

case 4:

{

printf("traversing\n");

traverse(head);

break;

}

case 5:printf("exit\n");

break;

default:printf("enter correct value\n\n");

break;

}

}while(ch!=5);

}

struct node\* deleteatfront(struct node\* head)

{

if(head==NULL){

printf("linked list underflow\n");

}

else{

struct node \*ptr;

ptr=head;

head=ptr->next;

free(ptr);

}

return head;

}

struct node\* deleteatlast(struct node\* head)

{

if(head==NULL){

printf("linked list underflow\n");

}

else{

if(head->next==NULL){

head=NULL;

free(head);

}

else{

struct node \*ptr;

struct node \*ptr1;

ptr=head;

while(ptr->next!=NULL){

ptr1=ptr;

ptr=ptr->next;

}

ptr1->next=NULL;

free(ptr);

}

}

return head;

}

struct node\* deleteatpos(struct node\* head)

{

struct node \*ptr;

struct node \*ptr1;

int i,pos;

printf("enter position\n");

scanf("%d",&pos);

if (head == NULL){

printf("Linked list underflow\n");

return head;

}

if (pos==1){

ptr=head;

head=ptr->next;

free(ptr);

printf("Value deleted from position\n");

}

else{

ptr = head;

for(i=1;i<pos-1&&ptr!=NULL;i++)  {

ptr = ptr->next;

}

if (ptr==NULL||ptr->next==NULL){

printf("Position out of range\n");

return head;

}

ptr1 = ptr->next;

ptr->next = ptr1->next;

free(ptr1);

printf("Value deleted from position\n");

}

return head;

}

void traverse(struct node\* head)

{

struct node \*ptr;

ptr=head;

if(head==NULL){

printf("list empty\n");

}

else{

while(ptr!=NULL)s{

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

ptr=ptr->next;

}

printf("NULL\n");

}

}

**Output**

Creating a linked list:

enter size

7

enter value to insert

2

enter value to insert

3

enter value to insert

4

enter value to insert

5

enter value to insert

6

enter value to insert

7

enter value to insert

8

Linked List Operations

1.Delete From Front

2.Delete From Last

3.Delete From Particular Position

4.Traversal

5.Exit

choose an operation:

4

traversing

2->3->4->5->6->7->8->NULL

Linked List Operations

1.Delete From Front

2.Delete From Last

3.Delete From Particular Position

4.Traversal

5.Exit

choose an operation:

1

value deleted from front

Linked List Operations

1.Delete From Front

2.Delete From Last

3.Delete From Particular Position

4.Traversal

5.Exit

choose an operation:

4

traversing

3->4->5->6->7->8->NULL

Linked List Operations

1.Delete From Front

2.Delete From Last

3.Delete From Particular Position

4.Traversal

5.Exit

choose an operation:

2

value deleted from last

Linked List Operations

1.Delete From Front

2.Delete From Last

3.Delete From Particular Position

4.Traversal

5.Exit

choose an operation:

4

traversing

3->4->5->6->7->NULL

Linked List Operations

1.Delete From Front

2.Delete From Last

3.Delete From Particular Position

4.Traversal

5.Exit

choose an operation:

3

enter position

3

Value deleted from position

Linked List Operations

1.Delete From Front

2.Delete From Last

3.Delete From Particular Position

4.Traversal

5.Exit

choose an operation:

4

traversing

3->4->6->7->NULL

Linked List Operations

1.Delete From Front

2.Delete From Last

3.Delete From Particular Position

4.Traversal

5.Exit

choose an operation:

3

enter position

6

Position out of range

Linked List Operations

1.Delete From Front

2.Delete From Last

3.Delete From Particular Position

4.Traversal

5.Exit

choose an operation:

5

exit

**Experiment 14 Date:02/12/2024**

**Stack Using Singly Linked List**

**Aim:**

To implement a menu driven program to perform following stack operations using

linked list

i. Push

ii. pop

iii.Traversal

**Algorithm:**

**Declare the Structure node**

struct node

1.declare data, struct node\* next

**main()**

1. Start

2 Declare ch,struct node\* head = NULL

3 Display Choices.

4 Read option ch.

If ch==1 call push(head)

If ch==2 call pop(head)

If ch==3 call display(head)

If ch==4 exit

5 Repeat steps 3 & 4 while ch>0&&ch<4

6 Stop

**struct node\* push(struct node\* head)**

1 Start

2 Declare value,struct node\* ptr

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

4 Read value

5 newnode->data=value

6 newnode->next=NULL

7 if head==NULL

newnode->next=NULL

head=newnode

else

ptr=head

while ptr->next != NULL

ptr=ptr->next

8 ptr->next=newnode

9 Return head

10 Exit

**struct node\* pop(struct node\* head)**

1 Start

2 Decalre struct node\* ptr,struct node\* ptr1

3 if head==NULL

Print Stack Underflow

else

if ptr->next == NULL

head=NULL

free(head)

else

ptr=head

while ptr->next != NULL

ptr1=ptr

ptr=ptr->next

ptr1->next=NULL

free(ptr)

4 Return head

5 Exit

**void display(struct node\* head)**

1 Start

2 Declare struct node \*ptr

3 ptr=head

4 if head==NULL

Print Stack Empty

else

while ptr != NULL

Print ptr->data

ptr=ptr->next

Print Null

5 Exit

**Program**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*next;

};

struct node\* push(struct node\* head);

struct node\* pop(struct node\* head);

void display(struct node\* head);

void main()

{

int ch;

struct node\* head = NULL;

do

{

printf("Stack Operations\n1.Push\n2.Pop\n3.Display\n4.Exit\n");

printf("choose an operation:\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

{

head=push(head);

break;

}

case 2:

{

head=pop(head);

break;

}

case 3:

{

printf("display\n");

display(head);

break;

}

case 4:printf("exit\n");

break;

default:printf("enter correct value\n\n");

break;

}

}while(ch!=4);

}

struct node\* push(struct node\* head)

{

int value;

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

printf("enter value to insert\n");

scanf("%d",&value);

newnode->data=value;

newnode->next=NULL;

struct node \*ptr;

if(head==NULL){

newnode->next=NULL;

head=newnode;

}

else{

ptr=head;

while(ptr->next!=NULL){

ptr=ptr->next;

}

}

ptr->next=newnode;

printf("value pushed\n");

return head;

}

struct node\* pop(struct node\* head)

{

if(head==NULL){

printf("stack underflow\n");

}

else{

if(head->next==NULL){

head=NULL;

free(head);

}

else{

struct node \*ptr;

struct node \*ptr1;

ptr=head;

while(ptr->next!=NULL){

ptr1=ptr;

ptr=ptr->next;

}

ptr1->next=NULL;

free(ptr);

}

}

printf("value poped\n");

return head;

}

void display(struct node\* head)

{

struct node \*ptr;

ptr=head;

if(head==NULL){

printf("stack empty\n");

}

else{

while(ptr!=NULL){

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

ptr=ptr->next;

}

printf("NULL\n");

}

}

**Output**

Stack Operations

1.Push

2.Pop

3.Display

4.Exit

choose an operation:

1

enter value to insert

4

value pushed

Stack Operations

1.Push

2.Pop

3.Display

4.Exit

choose an operation:

1

enter value to insert

5

value pushed

Stack Operations

1.Push

2.Pop

3.Display

4.Exit

choose an operation:

3

display

4->5->NULL

Stack Operations

1.Push

2.Pop

3.Display

4.Exit

choose an operation:

1

enter value to insert

6

value pushed

Stack Operations

1.Push

2.Pop

3.Display

4.Exit

choose an operation:

3

display

4->5->6->NULL

Stack Operations

1.Push

2.Pop

3.Display

4.Exit

choose an operation:

2

value poped

Stack Operations

1.Push

2.Pop

3.Display

4.Exit

choose an operation:

3

display

4->5->NULL

Stack Operations

1.Push

2.Pop

3.Display

4.Exit

choose an operation:

2

value poped

Stack Operations

1.Push

2.Pop

3.Display

4.Exit

choose an operation:

3

display

4->NULL

Stack Operations

1.Push

2.Pop

3.Display

4.Exit

choose an operation:

4

exit

**Experiment 15 Date:02/12/2024**

**Queue Using Singly Linked List**

**Aim:**

To implement a menu driven program to perform following queue operations using

linked list

i. Enqueue

ii. Dequeue

iii.Traversal

**Algorithm:**

**Declare the Structure node**

struct node

1.declare data, struct node\* next

**main()**

1. Start

2 Declare ch,struct node\* head = NULL

3 Display Choices.

4 Read option ch.

If ch==1 call enqueue(head)

If ch==2 call dequeue(head)

If ch==3 call display(head)

If ch==4 exit

5 Repeat steps 3 & 4 while ch>0&&ch<4

6 Stop

**struct node\* enqueue(struct node\* head)**

1 Start

2 Declare value,struct node\* ptr

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

4 Read value

5 newnode->data=value

6 newnode->next=NULL

7 if head==NULL

newnode->next=NULL

head=newnode

else

ptr=head

while ptr->next != NULL

ptr=ptr->next

8 ptr->next=newnode

9 Return head

10 Exit

**struct node\* dequeue(struct node\* head)**

1 Start

2 Declare struct node\* ptr

3 if head==NULL

Print Queue Underflow

else

ptr=head

head=ptr->next

free(ptr)

4 Return head

5 Exit

**void display(struct node\* head)**

1 Start

2 Declare struct node\* ptr

3 ptr=head

4 if head==NULL

Print Stack Empty

else

while ptr != NULL

Print ptr->data

ptr=ptr->next

Print Null

5 Exit

**Program**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*next;

};

struct node\* enqueue(struct node\* head);

struct node\* dequeue(struct node\* head);

void display(struct node\* head);

void main()

{

int ch;

struct node\* head = NULL;

do

{

printf("Queue Operations\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");

printf("choose an operation:\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

{

head=enqueue(head);

break;

}

case 2:

{

head=dequeue(head);

break;

}

case 3:

{

printf("display\n");

display(head);

break;

}

case 4:printf("exit\n");

break;

default:printf("enter correct value\n\n");

break;

}

}while(ch!=4);

}

struct node\* enqueue(struct node\* head)

{

int value;

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

printf("enter value to insert\n");

scanf("%d",&value);

newnode->data=value;

newnode->next=NULL;

struct node \*ptr;

if(head==NULL){

newnode->next=NULL;

head=newnode;

}

else{

ptr=head;

while(ptr->next!=NULL){

ptr=ptr->next;

}

}

ptr->next=newnode;

printf("value inserted\n");

return head;

}

struct node\* dequeue(struct node\* head)

{

if(head==NULL){

printf("underflow\n");

}

else{

struct node \*ptr;

ptr=head;

head=ptr->next;

free(ptr);

printf("value deleted\n");

}

return head;

}

void display(struct node\* head)

{

struct node \*ptr;

ptr=head;

if(head==NULL){

printf("stack empty\n");

}

else{

while(ptr!=NULL){

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

ptr=ptr->next;

}

printf("NULL\n");

}

}

**Output**

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter value to insert

2

value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter value to insert

3

value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

display

2->3->NULL

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

1

enter value to insert

4

value inserted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

display

2->3->4->NULL

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

2

value deleted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

display

3->4->NULL

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

2

value deleted

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

3

display

4->NULL

Queue Operations

1.Enqueue

2.Dequeue

3.Display

4.Exit

choose an operation:

4

exit

**Experiment 16 Date:04/12/2024**

**Doubly Linked List**

**Aim:**

To implement the following operations on a Doubly linked list.

i. Creation

ii. Count the number of nodes

iii. Searching

iv.Traversal

**Algorithm:**

**Declare the Structure node**

struct node

1.declare data, struct node\* prev,struct node \*next

**main()**

1. Start

2 Declare ch,struct node\* head = NULL

3 call createnode(head)

4 Display Choices.

5 Read option ch.

If ch==1 call noofnode(head)

If ch==2 call searchnode(head)

If ch==3 call traverse(head)

If ch==4 exit

6 Repeat steps 4 & 5 while ch>0&&ch<5

7 Stop

**struct node \*createnode(struct node\* head)**

1 Start

2 Declare n,value,struct node \*p

3 Read n

4 if n <= 0

Print Size Must Be Greater Than Zero

5 for i=1 to n

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

Read value

temp->data=value

temp->prev=NULL

temp->next=NULL

if head==NULL

head=temp

else

p=head

while p->next != NULL

p=p->next

p->next=temp

temp->prev=p

6 Return head

7 Exit

**struct node \*noofnode(struct node\* head)**

1 Start

2 Declare count=0,struct node \*ptr

3 ptr=head

4 while ptr != NULL

ptr=ptr->next

count=count+1

5 Print count

6 Exit

**void searchnode(struct node\* head)**

1 Start

2 Declare value,flag=0,pos=1,struct node \*ptr

3 Read value

4 ptr=head

5 while ptr != NULL

if ptr->data==value

flag=1

Print Value Present At Position pos

else

ptr=ptr->next

pos=pos+1

6 if flag==0

Print Value Not Found

7 Exit

**void traverse(struct node\* head)**

1 Start

2 Declare struct node \*ptr

3 ptr=head

4 if head==NULL

Print List Empty

else

while ptr != NULL

Print ptr->data

ptr=ptr->next

Print Null

5 Exit

**Program**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*next;

struct node \*prev;

};

struct node\* createnode(struct node\* head){

struct node \*p;

int value,n;

printf("enter size \n");

scanf("%d",&n);

if(n <= 0) {

printf("List size must be greater than 0.\n");

return 0;

}

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

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

printf("enter value to insert\n");

scanf("%d",&value);

temp->data=value;

temp->prev=NULL;

temp->next=NULL;

if (head==NULL){

head=temp;

}

else{

p=head;

while(p->next!=NULL){

p=p->next;

}

p->next=temp;

temp->prev=p;

}

}

return head;

}

void searchnode(struct node\* head);

void noofnodes(struct node\* head);

void traverse(struct node\* head);

void main()

{

int ch,val;

struct node \*head=NULL;

printf("Creating a linked list:\n");

head=createnode(head);

do

{

printf("Linked List Operations\n1.Count No Of Nodes\n2.Searching a node\n3.Traversal\n4.Exit\n");

printf("choose an operation:\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

{

printf("Node Count\n");

noofnodes(head);

break;

}

case 2:

{

printf("searching\n");

searchnode(head);

break;

}

case 3:

{

printf("traversing\n");

traverse(head);

break;

}

case 4:printf("exit\n");

break;

default:printf("enter correct value\n\n");

break;

}

}while(ch!=4);

}

void searchnode(struct node\* head)

{

struct node \*ptr;

ptr=head;

int flag=0,pos=1,val;

printf("enter value to search\n");

scanf("%d",&val);

while(ptr!=NULL){

if(ptr->data==val){

flag=1;

printf("Item Present at position %d\n",pos);

break;

}

else{

ptr=ptr->next;

pos=pos+1;

}

}

if(flag==0){

printf("Item Not Found\n");

}

}

void noofnodes(struct node\* head)

{

struct node \*ptr;

ptr=head;

int count=0;

while(ptr!=NULL){

ptr=ptr->next;

count=count+1;

}

printf("No of nodes is %d\n",count);

}

void traverse(struct node\* head)

{

struct node \*ptr;

ptr=head;

if(head==NULL){

printf("list empty\n");

}

else{

while(ptr!=NULL){

printf("%d<->",ptr->data);

ptr=ptr->next;

}

printf("NULL\n");

}

}

**Output**

Creating a linked list:

enter size

4

enter value to insert

2

enter value to insert

3

enter value to insert

4

enter value to insert

5

Linked List Operations

1.Count No Of Nodes

2.Searching a node

3.Traversal

4.Exit

choose an operation:

3

traversing

2<->3<->4<->5<->NULL

Linked List Operations

1.Count No Of Nodes

2.Searching a node

3.Traversal

4.Exit

choose an operation:

1

Node Count

No of nodes is 4

Linked List Operations

1.Count No Of Nodes

2.Searching a node

3.Traversal

4.Exit

choose an operation:

2

searching

enter value to search

4

Item Present at position 3

Linked List Operations

1.Count No Of Nodes

2.Searching a node

3.Traversal

4.Exit

choose an operation:

2

searching

enter value to search

10

Item Not Found

Linked List Operations

1.Count No Of Nodes

2.Searching a node

3.Traversal

4.Exit

choose an operation:

4

exit

**Experiment 17 Date:04/12/2024**

**Doubly Linked List-Insertion & sDeletion**

**Aim:**

To implement the following operations on a Doubly linked list.

i. Creation

ii. Insert a node at first position

iii. Insert a node at last

iv. Delete a node from the first position

v. Delete a node from last

vi.Traversal

**Algorithm:**

**Declare the Structure node**

struct node

1.declare data, struct node\* prev,struct node \*next

**main()**

1. Start

2 Declare ch,struct node\* head = NULL

3 call createnode(head)

4 Display Choices.

5 Read option ch.

If ch==1 call insertatfront(head)

If ch==2 call insertatlast(head)

If ch==3 call deleteatfront(head)

If ch==4 call deleteatlast(head)

If ch==5 call traverse(head)

If ch==6 exit

6 Repeat steps 4 & 5 while ch>0&&ch<6

7 Stop

**struct node \*createnode(struct node\* head)**

1 Start

2 Declare n,value,struct node \*p

3 Read n

4 if n <= 0

Print Size Must Be Greater Than Zero

5 for i=1 to n

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

Read value

temp->data=value

temp->prev=NULL

temp->next=NULL

if head==NULL

head=temp

else

p=head

while p->next != NULL

p=p->next

p->next=temp

temp->prev=p

6 Return head

7 Exit

**struct node \*insertatfront(struct node\* head)**

1 Start

2 Declare value

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

4 Read value

5 newnode->data=value

6 newnode->prev=NULL

7 newnode->next=NULL

8 if head==NULL

newnode->prev=NULL

newnode->next=NULL

head=newnode

else

newnode->prev=NULL

newnode->next=head;

head=newnode;

9 Return head

10 Exit

**struct node \*insertatlast(struct node\* head)**

1 Start

2 Declare value,struct node \*ptr

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

4 Read value

5 newnode->data=value

6 newnode->prev=NULL

7 newnode->next=NULL

8 if head==NULL

newnode->next=NULL

newnode->prev=NULL

head=newnode

else

ptr=head

while ptr->next != NULL

ptr=ptr->next

9 ptr->next=newnode

10 newnode->prev=ptr

11 Return head

12 Exit

**struct node \*deleteatfront(struct node\* head)**

1 Start

2 Declare struct node \*ptr

3 if head==NULL

Print Linked List Underflow

else

ptr=head

head=ptr->next

head->prev=NULL

free(ptr)

4 Return head

5 Exit

**struct node \*deleteatlast(struct node\* head)**

1 Start

2 Decalre struct node \*ptr,struct node \*ptr1

3 if head==NULL

Print Linked List Underflow

else

if ptr->next == NULL

head=NULL

free(head)

else

ptr=head

while ptr->next != NULL

ptr1=ptr

ptr=ptr->next

ptr1->next=NULL

ptr->prev=NULL;

free(ptr)

4 Return head

5 Exit

**void traverse(struct node\* head)**

1 Start

2 Declare struct node \*ptr

3 ptr=head

4 if head==NULL

Print List Empty

else

while ptr != NULL

Print ptr->data

ptr=ptr->next

Print Null

5 Exit

**Program**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*next;

struct node \*prev;

};

struct node\* createnode(struct node\* head){

struct node \*p;

int value,n;

printf("enter size \n");

scanf("%d",&n);

if(n <= 0) {

printf("List size must be greater than 0.\n");

return 0;

}

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

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

printf("enter value to insert\n");

scanf("%d",&value);

temp->data=value;

temp->prev=NULL;

temp->next=NULL;

if (head==NULL){

head=temp;

}

else{

p=head;

while(p->next!=NULL){

p=p->next;

}

p->next=temp;

temp->prev=p;

}

}

return head;

}

struct node\* insertatfront(struct node\* head);

struct node\* insertatlast(struct node\* head);

struct node\* deleteatfront(struct node\* head);

struct node\* deleteatlast(struct node\* head);

void traverse(struct node\* head);

void main()

{

int ch,data;

struct node \*head=NULL;

printf("Creating a linked list:\n");

head=createnode(head);

do

{

printf("Linked List Operations\n1.Insert Node At Front\n2.Insert Node At Last\n3.Delete Node At Front\n4.Delete Node At Last\n5.Traversal\n6.Exit\n");

printf("choose an operation:\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

{

head=insertatfront(head);

printf("value inserted at front\n");

break;

}

case 2:

{

head=insertatlast(head);

printf("value inserted at last\n");

break;

}

case 3:

{

head=deleteatfront(head);

printf("value deleted from front\n");

break;

}

case 4:

{

head=deleteatlast(head);

printf("value deleted from last\n");

break;

}

case 5:

{

printf("traversing\n");

traverse(head);

break;

}

case 6:printf("exit\n");

break;

default:printf("enter correct value\n\n");

break;

}

}while(ch!=6);

}

struct node\* insertatfront(struct node\* head)

{

int value;

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

printf("enter value to insert\n");

scanf("%d",&value);

newnode->data=value;

newnode->prev=NULL;

newnode->next=NULL;

if(head==NULL){

newnode->next=NULL;

newnode->prev=NULL;

head=newnode;

}

else{

newnode->next=head;

newnode->prev=NULL;

head=newnode;

}

return head;

}

struct node\* insertatlast(struct node\* head)

{

int value;

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

printf("enter value to insert\n");

scanf("%d",&value);

newnode->data=value;

newnode->prev=NULL;

newnode->next=NULL;

struct node \*ptr;

if(head==NULL){

newnode->next=NULL;

newnode->prev=NULL;

head=newnode;

}

else{

ptr=head;

while(ptr->next!=NULL){

ptr=ptr->next;

}}

ptr->next=newnode;

newnode->prev=ptr;

return head;

}

struct node\* deleteatfront(struct node\* head)

{

if(head==NULL){

printf("linked list underflow\n");

}

else{

struct node \*ptr;

ptr=head;

head=ptr->next;

head->prev=NULL;

free(ptr);

}

return head;

}

struct node\* deleteatlast(struct node\* head)

{

if(head==NULL){

printf("linked list underflow\n");

}

else{

if(head->next==NULL){

head=NULL;

free(head);

}

else{

struct node \*ptr;

struct node \*ptr1;

ptr=head;

while(ptr->next!=NULL){

ptr1=ptr;

ptr=ptr->next;

}

ptr1->next=NULL;

ptr->prev=NULL;

free(ptr);

}}

return head;

}

void traverse(struct node\* head)

{

struct node \*ptr;

ptr=head;

if(head==NULL){

printf("list empty\n");

}

else{

while(ptr!=NULL){

printf("%d<->",ptr->data);

ptr=ptr->next;

}

printf("NULL\n");

}}

**Output**

Creating a linked list:

enter size

3

enter value to insert

6

enter value to insert

7

enter value to insert

8

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Delete Node At Front

4.Delete Node At Last

5.Traversal

6.Exit

choose an operation:

5

traversing

6<->7<->8<->NULL

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Delete Node At Front

4.Delete Node At Last

5.Traversal

6.Exit

choose an operation:

1

enter value to insert

5

value inserted at front

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Delete Node At Front

4.Delete Node At Last

5.Traversal

6.Exit

choose an operation:

5

traversing

5<->6<->7<->8<->NULL

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Delete Node At Front

4.Delete Node At Last

5.Traversal

6.Exit

choose an operation:

2

enter value to insert

12

value inserted at last

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Delete Node At Front

4.Delete Node At Last

5.Traversal

6.Exit

choose an operation:

5

traversing

5<->6<->7<->8<->12<->NULL

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Delete Node At Front

4.Delete Node At Last

5.Traversal

6.Exit

choose an operation:

3

value deleted from front

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Delete Node At Front

4.Delete Node At Last

5.Traversal

6.Exit

choose an operation:

5

traversing

6<->7<->8<->12<->NULL

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Delete Node At Front

4.Delete Node At Last

5.Traversal

6.Exit

choose an operation:

4

value deleted from last

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Delete Node At Front

4.Delete Node At Last

5.Traversal

6.Exit

choose an operation:

5

traversing

6<->7<->8<->NULL

Linked List Operations

1.Insert Node At Front

2.Insert Node At Last

3.Delete Node At Front

4.Delete Node At Last

5.Traversal

6.Exit

choose an operation:

6

exit

**Experiment 18 Date:09/12/2024**

**Binary Search Tree**

**Aim:**

Menu Driven program to implement Binary Search Tree (BST) Operations- Insertion

of node, Deletion of a node, In-order traversal, Pre-order traversal and post-order

traversal.

**Algorithm:**

**Declare the Structure node**

struct node

1.declare data, struct node\* left,struct node \*right

**main()**

1. Start

2 Declare ch,struct node\* root = NULL,item,value

3 Display Choices.

4 Read option ch.

If ch==1 Read item andcall insertnode(root,item)

If ch==2 Read value and call deletenode(root,value)

If ch==3 call inorder(root)

If ch==4 call preorder(root)

If ch==5 call postorder(root)

If ch==6 exit

5 Repeat steps 3 & 4 while ch>0&&ch<6

6 Stop

**struct node \*create(int item)**

1 Start

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

3 temp->data = item;

4 temp->left = NULL;

5 temp->right = NULL;

6 Return temp;

7 Exit

**struct node \*insertnode(struct node\* root, int item)**

1 Start

2 if root == NULL

call create(item)

3 if (item < root->data)

root->left =call insertnode(root->left, item)

else

root->right =call insertnode(root->right, item);

4 Return root

5 Exit

**struct node \*inorderpredecessor(struct node\* root)**

1 Start

2 root = root->left;

3 while root->right != NULL

root = root->right

4 return root;

5 Exit

**struct node \*deletenode(struct node\* root,int value)**

1 Start

2 if root == NULL

Print Node not found

Return root

3 struct node\* temp

4 if value < root->data

root->left =call deletenode(root->left, value)

else if value > root->data

root->right =call deletenode(root->right, value)

else

if root->left == NULL && root->right == NULL

free(root)

Return NULL

else if root->left == NULL

temp = root->right

free(root)

Return temp

else if root->right == NULL

temp = root->left

free(root)

Return temp

else

temp =call inorderpredecessor(root)

root->data = temp->data

root->left = deletenode(root->left, temp->data)

5 Return root

6 Exit

**void inorder(struct node\* root)**

1 Start

2 if root == NULL

return

3 inorder(root->left)

4 printf("%d\t", root->data)

5 inorder(root->right)

6 Exit

**void preorder(struct node\* root)**

1 Start

2 if root == NULL

return

3 printf("%d\t", root->data)

4 preorder(root->left)

5 preorder(root->right)

6 Exit

**void postorder(struct node\* root)**

1 Start

2 if root == NULL

return

3 postorder(root->left);

4 postorder(root->right);

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

6 Exit

**Program**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\* left;

struct node\* right;

};

struct node\* create(int item)

{

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

temp->data = item;

temp->left = NULL;

temp->right = NULL;

return temp;

}

struct node\* insertnode(struct node\* root, int item);

struct node\* deletenode(struct node\* root, int value);

struct node\* inorderpredecessor(struct node\* root);

void inorder(struct node\* root);

void preorder(struct node\* root);

void postorder(struct node\* root);

void main()

{

int ch,item,value;

struct node\* root = NULL;

do

{

printf("Binary Search Tree Operations\n1.Insert a Node\n2.Delete a Node\n3.In-order Traversal\n4.Pre-order Traversal\n5.Post-order Traversal\n6.Exit\n");

printf("choose an operation:\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

{

printf("Enter item for new node: ");

scanf("%d", &item);

root = insertnode(root, item);

printf("value inserted\n");

break;

}

case 2:

{

if(root==NULL)

{

printf("Tree is empty\n");

}

else

{

printf("Enter value to delete: ");

scanf("%d", &value);

root = deletenode(root, value);

printf("value deleted\n");

}

break;

}

case 3:

{

if(root==NULL)

{

printf("Tree is empty\n");

}

else

{

printf("In Order Traversal\n");

inorder(root);

printf("\n");

}

break;

}

case 4:

{

if(root==NULL)

{

printf("Tree is empty\n");

}

else

{

printf("Pre Order Traversal\n");

preorder(root);

printf("\n");

}

break;

}

case 5:

{

if(root==NULL)

{

printf("Tree is empty\n");

}

else

{

printf("Post Order Traversal\n");

postorder(root);

printf("\n");

}

break;

}

case 6:printf("exit\n");

break;

default:printf("enter correct value\n\n");

break;

}

}while(ch!=6);

}

struct node\* insertnode(struct node\* root, int item)

{

if (root == NULL)

{

return create(item);

}

if (item < root->data)

{

root->left = insertnode(root->left, item);

}

else

{

root->right = insertnode(root->right, item);

}

return root;

}

struct node\* inorderpredecessor(struct node\* root)

{

root = root->left;

while (root->right != NULL)

{

root = root->right;

}

return root;

}

struct node\* deletenode(struct node\* root, int value)

{

if (root == NULL)

{

printf("Node not found\n");

return root;

}

struct node\* temp;

if (value < root->data)

{

root->left = deletenode(root->left, value);

}

else if (value > root->data)

{

root->right = deletenode(root->right, value);

}

else

{

if (root->left == NULL && root->right == NULL)

{

free(root);

return NULL;

}

else if (root->left == NULL)

{

temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL)

{

temp = root->left;

free(root);

return temp;

}

else

{

temp = inorderpredecessor(root);

root->data = temp->data;

root->left = deletenode(root->left, temp->data);

}

}

return root;

}

void inorder(struct node\* root)

{

if(root == NULL) return;

inorder(root->left);

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

inorder(root->right);

}

void preorder(struct node\* root)

{

if(root == NULL) return;

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

preorder(root->left);

preorder(root->right);

}

void postorder(struct node\* root)

{

if(root == NULL) return;

postorder(root->left);

postorder(root->right);

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

}

**Output**

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

1

Enter item for new node: 100

value inserted

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

1

Enter item for new node: 20

value inserted

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

1

Enter item for new node: 200

value inserted

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

1

Enter item for new node: 10

value inserted

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

1

Enter item for new node: 30

value inserted

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

1

Enter item for new node: 150

value inserted

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

1

Enter item for new node: 300

value inserted

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

3

In Order Traversal

10 20 30 100 150 200 300

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

4

Pre Order Traversal

100 20 10 30 200 150 300

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

5

Post Order Traversal

10 30 20 150 300 200 100

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

2

Enter value to delete: 50

Node not found

value deleted

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

2

Enter value to delete: 10

value deleted

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

3

In Order Traversal

20 30 100 150 200 300

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

4

Pre Order Traversal

100 20 30 200 150 300

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

5

Post Order Traversal

30 20 150 300 200 100

Binary Search Tree Operations

1.Insert a Node

2.Delete a Node

3.In-order Traversal

4.Pre-order Traversal

5.Post-order Traversal

6.Exit

choose an operation:

6

exit

**Experiment 19 Date: 09/12/2024**

**Red Black Tree Operations**

**Aim**: Create Red Black Tree and perform the following operation.

i. Create

ii. Insert a new node

iii. Left rotate

iv. Right rotate

v. Inorder traversal

**Algorithm:**

## Declare the structure node

typedef struct Node

* 1. Declare data, struct Node \*parent, \*left, \*right, color typedef struct RedBlackTree
  2. Declare Node\* root

## main()

1. Start
2. Declare choice, value,.
3. Call RedBlackTree\* tree= createRedBlackTree
4. Display choices.
5. Read option choice.

If ch==1

Read the value to insert call insert(tree, value)

If ch==2

Read the value to left rotate

Node\* leftRotNode=search(tree->root,value) call leftRotate(tree, leftRotNode)

If ch==3 call

Read the value to right rotate

Node\* rightRotNode=search(tree->root,value) call leftRotate(tree, rightRotNode)

If ch==4 call inOrderTraversal(root) If ch==5 exit from the program

1. Repeat steps 4 and 5 while(1)
2. Stop

## Node\* createNode(int data)

1. Start.
2. Allocate memory for newNode
3. If (newNode=NULL)

Display “Memory allocation error” Go to step 8

1. Set newNode->data=data
2. Set newNode->parent= newNode->left=newNode->right=NULL
3. newNode->color=1
4. Return newNode
5. Exit.

## RedBlackTree\* createRedBlackTree()

1. Start.
2. Allocate memory for newTree
3. If (newTree =NULL)

Display “Memory allocation error” Go to step 6

1. newTree ->root=NULL
2. Return newTree
3. Exit.

## Void leftRotate(RedBlackTree\* tree, Node\* x)

1. Start.
2. If (x =NULL || x->right=NULL) Display “Left rotation not possible” Go to step 10
3. Set Node\* y=x->right
4. Set x->right=y->left
5. If (y->left != NULL)

Set y->left->parent = x

1. Set y->parent = x->parent
2. If (x->parent = NULL) Set tree->root = y

Else if (x == x->parent->left) Set x->parent->left = y

Else

Set x->parent->right = y

1. Set y->left = x
2. Set x->parent = y
3. Exit.

## Void rightRotate(RedBlackTree\* tree, Node\* y)

1. Start.
2. If (y == NULL || y->left == NULL) Display “Right rotation not possible” Go to step 10
3. Set Node\* x=y->right
4. Set y->right=x->left
5. If (x->right!= NULL)

Set x->right ->parent = y

1. Set x->parent = y->parent;
2. If (y->parent = NULL) Set tree->root = x

Else if (y == y->parent->left) Set y->parent->left = x

Else

Set y->parent->right = x

1. Set x->right= y
2. Set y->parent = x
3. Exit.

## Void insertFixup(RedBlackTree\* tree, Node\* z)

1. Start.
2. While (z->parent != NULL && z->parent->color == 1) If (z->parent == z->parent->parent->left)

Set Node\* y = z->parent->parent->right If (y != NULL && y->color == 1)

Set z->parent->color = 0 Set y->color = 0

Set z->parent->parent->color = 1 Set z = z->parent->parent

Else

If (z == z->parent->right) Set z = z->parent

Call leftRotate(tree, z) Set z->parent->color = 0

Set z->parent->parent->color = 1

Call rightRotate(tree, z->parent->parent)

Else

Set Node\* y = z->parent->parent->left If (y != NULL && y->color == 1)

Set z->parent->color = 0 Set y->color = 0

Set z->parent->parent->color = 1 Set z = z->parent->parent

Else

If (z == z->parent->left) Set z = z->parent;

Call rightRotate(tree, z) Set z->parent->color = 0

Set z->parent->parent->color = 1

Call leftRotate(tree, z->parent->parent)

1. Set tree->root->color = 0
2. Exit.

## Void insert(RedBlackTree\* tree, int data)

1. Start.
2. Call Node\* z = createNode(data)
3. Set Node\* y = NULL
4. Set Node\* x = tree->root;
5. While (x != NULL) Set y=x

If (z->data < x->data) Set x = x->left

Else

Set x = x->right

1. Set z->parent = y
2. If (y= NULL)

Set tree->root = z

Else if (z->data < y->data) Set y->left = z

Else

Set y->right = z

1. Call insertFixup(tree, z)
2. Exit.

## Node\* search(Node\* root, int value)

1. Start.
2. If (root == NULL || root->data == value) Return root
3. If (value < root->data)

Return search(root->left, value)

1. Return search(root->right, value)
2. Exit.

## void inOrderTraversal(Node\* root)

1. Start.
2. char\* colorStr[2] = {"BLACK", "RED"}
3. If (root != NULL)

Call inOrderTraversal(root->left); Display root->data

Call inOrderTraversal(root->right)

1. Exit.

## void freeMemory(Node\* root)

1. Start.
2. If (root = NULL)

Go to step 6

1. Call freeMemory(root->left)
2. Call freeMemory(root->right)
3. Free(root)
4. Exit

**Program**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node

{

int data;

struct Node\* parent;

struct Node\* left;

struct Node\* right;

int color; // 0 for black, 1 for red

} Node;

typedef struct RedBlackTree

{ Node\* root;

} RedBlackTree;

Node\* createNode(int data);

RedBlackTree\* createRedBlackTree();

void leftRotate(RedBlackTree\* tree, Node\* x);

void rightRotate(RedBlackTree\* tree, Node\* y);

void insertFixup(RedBlackTree\* tree, Node\* z);

void insert(RedBlackTree\* tree, int data);

Node\* search(Node\* root, int value);

void inOrderTraversal(Node\* root);

void freeMemory(Node\* root);

int main() {

int choice, value;

RedBlackTree\* tree = createRedBlackTree();

while (1)

{

printf("\n1. Insertion\n2. Left Rotation\n3. Right Rotation\n4. Display\n5.

Exit");

printf("\nEnter your choice: ");

switch (choice) {

case 1:

printf("Enter the value to be inserted: ");

if (scanf("%d", &value) != 1)

{

printf("Invalid input. Try again.\n");

while (getchar() != '\n'); // Clear input buffer break;

}

insert(tree, value);

break;

case 2:

printf("Enter the value to be left rotated: ");

if (scanf("%d", &value) != 1)

{

printf("Invalid input. Try again.\n");

while (getchar() != '\n'); // Clear input buffer break;

}

{

Node\* leftRotNode = search(tree->root, value);

if (leftRotNode != NULL)

{

printf("Performing left rotation on node %d\n", value);

leftRotate(tree, leftRotNode);

} else

printf("Node not found!\n");

}

break;

case 3:

printf("Enter the value to be right rotated: ");

if (scanf("%d", &value) != 1) {

printf("Invalid input. Try again.\n");

while (getchar() != '\n'); // Clear input buffer break;

}

{

Node\* rightRotNode = search(tree->root, value);

if (rightRotNode != NULL) {

printf("Performing right rotation on node %d\n", value);

rightRotate(tree, rightRotNode);

} else

printf("Node not found!\n");

}

break;

case 4:

printf("In-order traversal of the tree:\n");

inOrderTraversal(tree->root);

printf("\n");

break;

case 5:

freeMemory(tree->root);

free(tree);

printf("Exiting the program.\n");

exit(0);

default:

printf("\nWrong selection!!! Try again!!!\n");

}}

return 0;

}

Node\* createNode(int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

if (newNode == NULL) {

printf("Memory allocation error\n");

exit(1);

}

newNode->data = data;

newNode->parent = newNode->left = newNode->right = NULL;

newNode->color = 1; // New nodes are initially red

printf("Created node with value %d\n", data);

return newNode;

}

RedBlackTree\* createRedBlackTree() {

RedBlackTree\* newTree = (RedBlackTree\*)malloc(sizeof(RedBlackTree));

if (newTree == NULL) {

printf("Memory allocation error\n"); exit(1);

}

newTree->root = NULL; return newTree;

}

void leftRotate(RedBlackTree\* tree, Node\* x) {

if (x == NULL || x->right == NULL) {

printf("Left rotation not possible for node %d\n", x ? x->data : -1);

return;

}

Node\* y = x->right;

x->right = y->left;

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

y->parent = x->parent;

if (x->parent == NULL)

tree->root = y;

else if (x == x->parent->left) x->parent->left = y;

else

x->parent->right = y;

y->left = x;

x->parent = y;

printf("Left rotation performed on node %d\n", x->data);

}

void rightRotate(RedBlackTree\* tree, Node\* y) {

if (y == NULL || y->left == NULL) {

printf("Right rotation not possible for node %d\n", y ? y->data : -1);

return;

}

Node\* x = y->left; y->left = x->right;

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

x->parent = y->parent;

if (y->parent == NULL)

tree->root = x;

else if (y == y->parent->left)

y->parent->left = x; else

y->parent->right = x; x->right = y;

y->parent = x;

printf("Right rotation performed on node %d\n", y->data);

}

void insertFixup(RedBlackTree\* tree, Node\* z) {

while (z->parent != NULL && z->parent->color == 1) {

if (z->parent == z->parent->parent->left) {

Node\* y = z->parent->parent->right;

if (y != NULL && y->color == 1) {

z->parent->color = 0; // Black y->color = 0; // Black

z->parent->parent->color = 1; // Red z = z->parent->parent;

}

else {

if (z == z->parent->right) {

z = z->parent; leftRotate(tree, z);

}

z->parent->color = 0; // Black

z->parent->parent->color = 1; // Red rightRotate(tree, z->parent->parent);

}

} else {

Node\* y = z->parent->parent->left;

if (y != NULL && y->color == 1)

{ z->parent->color = 0; // Black

y->color = 0; // Black

z->parent->parent->color = 1; // Red z = z->parent->parent;

}

else {

if (z == z->parent->left) {

z = z->parent; rightRotate(tree, z);

}

z->parent->color = 0; // Black

z->parent->parent->color = 1; // Red leftRotate(tree, z->parent->parent);

}}}

tree->root->color = 0; // Root must be black printf("Tree fixed after insertion\n");

}

void insert(RedBlackTree\* tree, int data) {

Node\* z = createNode(data);

Node\* y = NULL;

Node\* x = tree->root;

while (x != NULL) {

y = x;

if (z->data < x->data) x = x->left;

else

x = x->right;

}

z->parent = y; if (y == NULL)

tree->root = z;

else if (z->data < y->data) y->left = z;

else

y->right = z;

printf("Inserted node with value %d\n", data);

insertFixup(tree, z);

}

Node\* search(Node\* root, int value) {

if (root == NULL || root->data == value)

return root;

if (value < root->data)

return search(root->left, value);

return search(root->right, value);

}

void inOrderTraversal(Node\* root) {

char\* colorStr[2] = {"BLACK", "RED"};

if (root != NULL) {

inOrderTraversal(root->left);

printf("%d,%s->",root->data,colorStr[root->color]); inOrderTraversal(root->right);

}}

void freeMemory(Node\* root) {

if (root == NULL)

return; freeMemory(root->left);

freeMemory(root->right);

free(root);

}

**Output**

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 1

Enter the value to be inserted: 10 Inserted node with value 10

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 1

Enter the value to be inserted: 5 Inserted node with value 5

Tree fixed after insertion

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 1

Enter the value to be inserted: 3 Inserted node with value 3

Right rotation performed on node 10 Tree fixed after insertion

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 1

Enter the value to be inserted: 4 Inserted node with value 4

Tree fixed after insertion

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 1

Enter the value to be inserted: 9 Inserted node with value 9

Tree fixed after insertion

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 1

Enter the value to be inserted: 8 Inserted node with value 8

Right rotation performed on node 10 Tree fixed after insertion

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 4

In-order traversal of the tree:

3, BLACK -> 4, RED -> 5, BLACK -> 8, RED -> 9, BLACK -> 10, RED ->

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 2

Enter the value to be left rotated: 9 Left rotation performed on node 9

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 4

In-order traversal of the tree:

3, BLACK -> 4, RED -> 5, BLACK -> 8, RED -> 9, BLACK -> 10, RED ->

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 3

Enter the value to be right rotated: 3 Right rotation not possible for node 3

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 3

Enter the value to be right rotated: 5 Right rotation performed on node 5

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 4

In-order traversal of the tree:

3, BLACK -> 4, RED -> 5, BLACK -> 8, RED -> 9, BLACK -> 10, RED ->

1. Insertion
2. Left Rotation
3. Right Rotation
4. Display
5. Exit

Enter your choice: 5

**Experiment 20 Date:11/12/2024**

**B Tree**

**Aim:**

Write a program to implement the following operation on B Tree.

1.Creation

2. Insertion

3.Searching

4.In-order Traversal

**Algorithm:**

## Declare the structure node and define variables

* 1. #define MAX\_KEYS 3
  2. #define MIN\_KEYS 1
  3. #define MAX\_CHILDREN typedef struct BTreeNode
  4. Declare keys[MAX\_KEYS], numKeys, isLeaf
  5. Declare BTreeNode\* children[MAX\_CHILDREN]

**main()**

1. Start
2. Call BTreeNode\* root = createNode(1)
3. Declare choice, key.
4. Display choices.
5. Read option choice.

If ch==1

Read the key to insert call insert(&root, key)

If ch==2

Read the key to search

If (search(root, key) != NULL) Display “Key found in the B-tree”

Else

Display “Key not found in the B-tree” If ch==3 call traverse(root)

If ch==4 exit from the program

1. Repeat steps 4 and 5 while(1)
2. Stop

## BTreeNode\* createNode(int isLeaf)

1. Start.
2. Allocate memory for node
3. Set node->isLeaf = isLeaf
4. Set node->numKeys = 0
5. For i=0 to MAX\_CHILDREN

Set node->children[i] = NULL

1. Return node
2. Exit.

## void splitChild(BTreeNode\* parent, int index, BTreeNode\* child)

1. Start.
2. Call BTreeNode\* newChild = createNode(child->isLeaf)
3. Set newChild->numKeys = MIN\_KEYSSet node->numKeys = 0
4. For i=0 to MIN\_KEYS

Set newChild->keys[i] = child->keys[i + MIN\_KEYS + 1]

1. If (!child->isLeaf)

For i=0 to MIN\_KEYS

Set newChild->children[i] = child->children[i + MIN\_KEYS + 1]

1. Set child->numKeys = MIN\_KEYS
2. For i= parent->numKeys – 1 to index

Set parent->keys[i + 1] = parent->keys[i]

1. Set parent->keys[index] = child->keys[MIN\_KEYS]
2. Set parent->numKeys++
3. Exit.

## void insertNonFull(BTreeNode\* node, int key)

1. Start.
2. Set i = node->numKeys - 1
3. If (node->isLeaf)

While (i >= 0 && node->keys[i] > key) Set node->keys[i + 1] = node->keys[i] Set i=i-1

Set node->keys[i + 1] = key Set node->numKeys++

1. Else

While (i >= 0 && node->keys[i] > key) Set i=i-1

If (node->children[i + 1]->numKeys == MAX\_KEYS) Call splitChild(node, i + 1, node->children[i + 1])

If (key > node->keys[i + 1]) Set i=i+1

Call insertNonFull(node->children[i + 1], key)

1. Exit.

## void insert(BTreeNode\*\* root, int key)

1. Start.
2. Set BTreeNode\* r = \*root
3. If (r->numKeys == MAX\_KEYS)

Call BTreeNode\* newRoot = createNode(0) Set newRoot->children[0] =r

Call splitChild(newRoot, 0, r)

Set i=0

If (newRoot->keys[0] < key) Set i=i+1

Call insertNonFull(newRoot->children[i], key) Set \*root = newRoot

1. Else

Call insertNonFull(r, key)

1. Exit.

## BTreeNode\* search(BTreeNode\* root, int key)

1. Start.
2. Set i=0
3. While (i < root->numKeys && key > root->keys[i]) Set i=i+1
4. If (i < root->numKeys && key == root->keys[i]) Return root
5. If (root->isLeaf) Return NULL
6. Exit.

## void traverse(BTreeNode\* root)

1. Start.
2. If (root == NULL) Go to step
3. For i=0 to root->numKeys If (!root->isLeaf)

Call traverse(root->children[i]) Display root->keys[i]

1. If (!root->isLeaf)

Call traverse(root->children[i])

1. Exit.

**Program**

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

#define MAX\_KEYS 3 // Maximum keys in a node (t-1 where t is the minimum degree)

#define MIN\_KEYS 1 // Minimum keys in a node (ceil(t/2) - 1)

#define MAX\_CHILDREN (MAX\_KEYS + 1) // Maximum children in a node (t)

typedef struct BTreeNode { int keys[MAX\_KEYS];

struct BTreeNode\* children[MAX\_CHILDREN];

int numKeys;

int isLeaf;

} BTreeNode;

BTreeNode\* createNode(int isLeaf) {

BTreeNode\* node = (BTreeNode\*)malloc(sizeof(BTreeNode)); node->isLeaf = isLeaf;

node->numKeys = 0;

for (int i = 0; i < MAX\_CHILDREN; i++) { node->children[i] = NULL;

}

return node;

}

void splitChild(BTreeNode\* parent, int index, BTreeNode\* child) { BTreeNode\* newChild = createNode(child->isLeaf);

newChild->numKeys = MIN\_KEYS;

for (int i = 0; i < MIN\_KEYS; i++) {

newChild->keys[i] = child->keys[i + MIN\_KEYS + 1];

}

if (!child->isLeaf) {

for (int i = 0; i < MIN\_KEYS + 1; i++) {

newChild->children[i] = child->children[i + MIN\_KEYS + 1];

}}

child->numKeys = MIN\_KEYS;

for (int i = parent->numKeys; i >= index + 1; i--) { parent->children[i + 1] = parent->children[i];

}

parent->children[index + 1] = newChild;

for (int i = parent->numKeys - 1; i >= index; i--) { parent->keys[i + 1] = parent->keys[i];

}

parent->keys[index] = child->keys[MIN\_KEYS]; parent->numKeys++;

}

void insertNonFull(BTreeNode\* node, int key) { int i = node->numKeys - 1;

if (node->isLeaf) {

while (i >= 0 && node->keys[i] > key) { node->keys[i + 1] = node->keys[i];

i--;

}

node->keys[i + 1] = key; node->numKeys++;

} else {

while (i >= 0 && node->keys[i] > key) { i--;

}

if (node->children[i + 1]->numKeys == MAX\_KEYS) {

splitChild(node, i + 1, node->children[i + 1]); if (key > node->keys[i + 1]) {

i++;

}}

insertNonFull(node->children[i + 1], key);

}}

void insert(BTreeNode\*\* root, int key) { BTreeNode\* r = \*root;

if (r->numKeys == MAX\_KEYS) { BTreeNode\* newRoot = createNode(0); newRoot->children[0] = r; splitChild(newRoot, 0, r);

int i = 0;

if (newRoot->keys[0] < key) { i++;

}

insertNonFull(newRoot->children[i], key);

\*root = newRoot;

} else {

insertNonFull(r, key);

}}

BTreeNode\* search(BTreeNode\* root, int key) { int i = 0;

while (i < root->numKeys && key > root->keys[i]) { i++;

}

if (i < root->numKeys && key == root->keys[i]) { return root;

}

if (root->isLeaf) { return NULL;

}

return search(root->children[i], key);

}

void traverse(BTreeNode\* root) { if (root == NULL) return;

int i;

for (i = 0; i < root->numKeys; i++) { if (!root->isLeaf) {

traverse(root->children[i]);

}

printf("%d ", root->keys[i]);

}

if (!root->isLeaf) { traverse(root->children[i]);

}}

int main() {

BTreeNode\* root = createNode(1); int choice, key;

while (1) {

printf("\nB-Tree Operations:\n"); printf("1. Insert\n");

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

case 1:

printf("Enter the key to insert: "); scanf("%d", &key); insert(&root, key);

break; case 2:

printf("Enter the key to search: "); scanf("%d", &key);

if (search(root, key) != NULL) {

printf("Key %d found in the B-tree.\n", key);

} else {

printf("Key %d not found in the B-tree.\n", key);

}

break; case 3:

printf("Traversal of the B-tree: \n"); traverse(root);

printf("\n"); break;

case 4:

exit(0); default:

printf("Invalid choice. Please try again.\n");

}}

return 0;

}

**Output**

B-Tree Operations:

1. Insert
2. Search
3. Traverse
4. Exit

Enter your choice: 1 Enter the key to insert: 5

B-Tree Operations:

1. Insert
2. Search
3. Traverse
4. Exit

Enter your choice: 1 Enter the key to insert: 7 B-Tree Operations:

1. Insert
2. Search
3. Traverse
4. Exit

Enter your choice: 1 Enter the key to insert: 4

B-Tree Operations:

1. Insert
2. Search
3. Traverse
4. Exit

Enter your choice: 1 Enter the key to insert: 9

B-Tree Operations:

1. Insert
2. Search
3. Traverse
4. Exit

Enter your choice: 3 Traversal of the B-tree: 4 5 7 9

B-Tree Operations:

1. Insert
2. Search
3. Traverse
4. Exit

Enter your choice: 2 Enter the key to search: 9 Key 9 found in the B-tree.

B-Tree Operations:

1. Insert
2. Search
3. Traverse
4. Exit

Enter your choice: 2

Enter the key to search: 10 Key 10 not found in the B-tree.

B-Tree Operations:

1. Insert
2. Search
3. Traverse
4. Exit

Enter your choice: 1 Enter the key to insert: 10

B-Tree Operations:

1. Insert
2. Search
3. Traverse
4. Exit

Enter your choice: 3 Traversal of the B-tree:

4 5 7 9 10

B-Tree Operations:

1. Insert
2. Search
3. Traverse
4. Exit

Enter your choice: 4

**Experiment 21 Date:17/12/2024**

# Different Operations of Set using Bit Strings

**Aim:**

Create the Abstract Data Type (ADT) using Set and perform the operations Union, Intersection and Difference operations. Implement using Bit Strings.

**Algorithm:**

## Declare the sets

1 Declare a[11], b[11], res[11]

2 Declare and initialize universal set U[11] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

**main()**

1. Start
2. Declare n, choice.
3. Read the limit of set A
4. Call input(a,n)
5. Read the limit of set B
6. Call input(b,n)
7. Display choices.
8. Read option choice.

If ch==1 call set\_union()

If ch==2 call set\_intersection() If ch==3 call set\_difference() If ch==4 call set\_equality())

If (set\_equality())

Display “Sets are equal” Else

Display “Sets are not equal” If ch==5 exit from the program

1. Repeat steps 7 and 8 while(1)
2. Stop

## void display(int bs[])

1. Start.
2. For i=1 to 11

Display bs[i]

1. Exit.

## void input(int bs[], int n)

1. Start.
2. Declare x

3 Read the elements

For i=0 n

If (x >= 1 && x <= 10) Set bs[x]=1

Else

Display “Invalid element. Please enter a number between 1 and 10.” Set i=i-1

1. Exit.

## void set\_union()

1. Start.
2. For i=1 to 11

Set res[i] = a[i] | b[i]

1. Display Union Set
2. Exit.

## void set\_intersection()

1. Start.
2. For i=1 to 11

Set res[i] = a[i] & b[i]

1. Display Intersection Set
2. Exit.

## void set\_difference()

1. Start.
2. For i=1 to 11

Set res[i] = a[i] & ~b[i]

1. Display Difference Set
2. Exit.

## bool set\_equality()

1. Start.
2. For i=1 to 11

If (a[i] != b[i]) Return false

1. Return true
2. Exit.

**Program**

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

int a[11] = {0}; // Set A int b[11] = {0}; // Set B

int res[11] = {0}; // Result Set

int U[11] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; // Universal Set void display(int bs[]) {

for (int i = 1; i < 11; i++) {

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

}

printf("\n");

}

void input(int bs[], int n) { int x;

printf("Enter the elements: "); for (int i = 0; i < n; i++) {

scanf("%d", &x);

if (x >= 1 && x <= 10) { bs[x] = 1;

} else {

printf("Invalid element. Please enter a number between 1 and 10.\n"); i--;

}}}

void set\_union() {

for (int i = 1; i < 11; i++) { res[i] = a[i] | b[i];

}

printf("\nUnion Set: "); display(res);

}

void set\_intersection() {

for (int i = 1; i < 11; i++) { res[i] = a[i] & b[i];

}

printf("\nIntersection Set: "); display(res);

}

void set\_difference() {

for (int i = 1; i < 11; i++) { res[i] = a[i] & ~b[i];

}

printf("\nDifference Set: "); display(res);

}

bool set\_equality() {

for (int i = 1; i < 11; i++) { if (a[i] != b[i]) {

return false;

}}

return true;

}

void main() { int n;

printf("Enter the number of elements in set A: "); scanf("%d", &n);

input(a, n);

printf("Enter the number of elements in set B: "); scanf("%d", &n);

input(b, n); int choice; while(1) {

printf("\nMENU");

printf("\n1. Union\n2. Intersection\n3. Difference\n4. Check Equality\n5.

Exit");

printf("\nEnter your choice: "); scanf("%d", &choice);

switch (choice) { case 1:

set\_union(); break;

case 2:

set\_intersection(); break;

case 3:

set\_difference(); break;

case 4:

if (set\_equality()) {

printf("\nSets A and B are equal.\n");

} else {

printf("\nSets A and B are not equal.\n");

}

break; case 5:

exit(0); default:

printf("\nInvalid choice! Please enter a number between 1 and 5.\n");

}} }

**Output**

Enter the number of elements in set A: 4 Enter the elements: 1

2

3

4

Enter the number of elements in set B: 4 Enter the elements: 3

4

5

6

MENU

1. Union
2. Intersection
3. Difference
4. Check Equality
5. Exit

Enter your choice: 1

Union Set: 1 1 1 1 1 1 0 0 0 0

MENU

1. Union
2. Intersection
3. Difference
4. Check Equality
5. Exit

Enter your choice: 2

Intersection Set: 0 0 1 1 0 0 0 0 0 0

MENU

1. Union
2. Intersection
3. Difference
4. Check Equality
5. Exit

Enter your choice: 3

Difference Set: 1 1 0 0 0 0 0 0 0 0

MENU

1. Union
2. Intersection
3. Difference
4. Check Equality
5. Exit

Enter your choice: 4

Sets A and B are not equal.

MENU

1. Union
2. Intersection
3. Difference
4. Check Equality
5. Exit

Enter your choice: 5

**Experiment 22 Date:30/12/2024**

# Disjoint Set Operations

**Aim:**

Implement the Disjoint set ADT with Create, Union and Find operations.

**Algorithm:**

## initSets()

1. declare ;
2. i = 0,i<numElements
3. sets[i].parent = i
4. sets[i].rank = 0

## find(int)

1. if (sets[element].parent != element)
2. sets[element].parent = find(sets[element].parent)
3. return sets[element].parent

## unionSets(int, int )

1. int set1 = find(element1);
2. int set2 = find(element2);
3. if (set1 != set2)
4. if (sets[set1].rank> sets[set2].rank)
5. sets[set2].parent = set1
6. else if (sets[set1].rank< sets[set2].rank)
7. sets[set1].parent = set2
8. else
9. sets[set2].parent = set1;
10. sets[set1].rank++;

## displaySets()

1. declare i;
2. i = 0, i<numElements
3. print i
4. for (i = 0; i<numElements; i++) {
5. print sets[i].parent
6. i = 0, i<numElements
7. print sets[i].rank

## main()

1. declare i
2. numElements = 6
3. unionSets(0, 1);
4. unionSets(1, 2);
5. unionSets(3, 4);
6. unionSets(4, 5);
7. unionSets(2, 4);
8. set i = 0
9. if i < numElements

Print find(i)

**Program**

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

#define MAX\_ELEMENTS 1000 typedef struct Set {

int parent; int rank;

} Set;

Set sets[MAX\_ELEMENTS]; int numElements;

void initSets() { int i;

for (i = 0; i<numElements; i++) { sets[i].parent = i;

sets[i].rank = 0;

}}

int find(int element) {

if (sets[element].parent != element) {

sets[element].parent = find(sets[element].parent); // Path compression

}

return sets[element].parent;

}

void unionSets(int element1, int element2) { int set1 = find(element1);

int set2 = find(element2);

if (set1!= set2) {

if (sets[set1].rank> sets[set2].rank) { sets[set2].parent = set1;

} else if (sets[set1].rank< sets[set2].rank) { sets[set1].parent = set2;

} else {

sets[set2].parent = set1; sets[set1].rank++;

}}}

void displaySets() { int i;

printf("Element:\t");

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

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

printf("\nParent:\t");

for (i = 0; i<numElements; i++) { printf("%d\t", sets[i].parent);

}

printf("\nRank:\t");

for (i = 0; i<numElements; i++) { printf("%d\t", sets[i].rank);

}

printf("\n\n");

}

int main() { int i;

numElements = 6; initSets(); displaySets(); unionSets(0, 1);

unionSets(1, 2);

unionSets(3, 4);

unionSets(4, 5);

unionSets(2, 4); displaySets();

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

printf("The representative element of element %d is %d\n", i, find(i));

}

return 0;

}

**Output**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Element: | 0 | 1 | 2 | 3 | 4 | 5 |
| Parent: | 0 | 1 | 2 | 3 | 4 | 5 |
| Rank: | 0 | 0 | 0 | 0 | 0 | 0 |
| Element: | 0 | 1 | 2 | 3 | 4 | 5 |
| Parent: | 0 | 0 | 0 | 0 | 3 | 3 |
| Rank: | 2 | 0 | 0 | 1 | 0 | 0 |

The representative element of element 0 is 0 The representative element of element 1 is 0 The representative element of element 2 is 0 The representative element of element 3 is 0 The representative element of element 4 is 0 The representative element of element 5 is 0

**Experiment 23 Date:30/12/2024**

# Max-Heap and Min-Heap

**Aim:**

Create a Max-Heap and Min-Heap from the array.

**Algorithm:**

## main()

* 1. Start
  2. Declare n, arr[n].
  3. Read the limit of array
  4. Read the elements of the array
  5. Call buildMaxHeap(arr, n)
  6. Call printArray(arr, n)
  7. Call buildMinHeap(arr, n)
  8. Call printArray(arr, n)
  9. Stop

## void maxHeapify(int arr[], int n, int i)

1. Start.
2. Set largest=i
3. Set left = 2 \* i + 1
4. Set right = 2 \* i + 2
5. If (left < n && arr[left] > arr[largest]) Set largest = left
6. If (right < n && arr[right] > arr[largest]) Set largest = right
7. If (largest != i)

Set temp = arr[i]

Set arr[i] = arr[largest] Set arr[largest] = temp

Call maxHeapify(arr, n, largest)

1. Exit.

## void buildMaxHeap(int arr[], int n)

1. Start.
2. For i=n/2-1 to 0

Call maxHeapify(arr, n, i)

1. Exit.

## void minHeapify(int arr[], int n, int i)

1. Start.
2. Set smallest=i
3. Set left = 2 \* i + 1
4. Set right = 2 \* i + 2
5. If (left < n && arr[left] < arr[smallest]) Set smallest= left
6. If (right < n && arr[right] < arr[smallest]) Set smallest= right
7. If (smallest!= i)

Set temp = arr[i]

Set arr[i] = arr[smallest] Set arr[smallest] = temp

Call minHeapify(arr, n, smallest)

1. Exit.

## void buildMinHeap(int arr[], int n)

1. Start.
2. For i=n/2-1 to 0

Call minHeapify(arr, n, i)

1. Exit.

## void printArray(int arr[], int n)

1. Start
2. For i = 0 to n

Display arr[i]

1. Exit.

**Program**

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

void maxHeapify(int arr[], int n, int i) { int largest = i;

int left = 2 \* i + 1; int right = 2 \* i +2;

if (left < n && arr[left] > arr[largest]) largest = left;

if (right < n && arr[right] > arr[largest]) largest = right;

if (largest != i) { int temp = arr[i];

arr[i] = arr[largest]; arr[largest] = temp; maxHeapify(arr, n, largest);

}}

void buildMaxHeap(int arr[], int n) { for (int i = n / 2 - 1; i >= 0; i--)

maxHeapify(arr, n, i);

}

void minHeapify(int arr[], int n, int i) { int smallest = i;

int left = 2 \* i + 1; int right = 2 \* i +2;

if (left < n && arr[left] < arr[smallest]) smallest = left;

if (right < n && arr[right] < arr[smallest]) smallest = right;

if (smallest != i) { int temp = arr[i];

arr[i] = arr[smallest]; arr[smallest] = temp; minHeapify(arr, n, smallest);

}

}

void buildMinHeap(int arr[], int n) { for (int i = n / 2 - 1; i >= 0; i--)

minHeapify(arr, n, i);

}

void printArray(int arr[], int n) { for (int i = 0; i < n; ++i)

printf("%d ", arr[i]); printf("\n");

}

void main(){ int n;

printf("Enter the number of elements in the array: "); scanf("%d", &n);

int arr[n];

printf("Enter the elements of the array: "); for (int i = 0; i < n; ++i)

scanf("%d",&arr[i]; buildMaxHeap(arr, n); printf("Max-Heap: "); printArray(arr, n); buildMinHeap(arr, n); printf("Min-Heap: "); printArray(arr, n);

}

**Output**

Enter the number of elements in the array: 8

Enter the elements of the array: 4

8

7

6

2

1

9

11

Max-Heap: 11 8 9 6 2 1 7 4

Min-Heap: 1 2 7 4 8 9 11 6

**Experiment 24 Date:01/01/2025**

# BFS and DFS

**Aim:**

Write a program to implement BFS and DFS on a connected undirected graph.

**Algorithm:**

## Declare stack, queue and global variables

1 Declare q[20], stack[20], a[20][20], vis[20]

2 Declare top = -1, front = -1, rear = -1

## main()

1. Start
2. Declare n, i, s, ch, j, c, dummy.
3. Read the number of vertices.
4. Read adjacency matrix.
5. For i=1 to n

Set vis[i]=0

1. Display choices
2. Read the source vertex
3. Read option choice

If ch==1 call bfs(s, n) If ch==2 dfs(s, n)

1. Read whether the user want to continue or not
2. Repeat steps 5 to 9 while (((c == 'y') || (c == 'Y')))
3. Stop

## void bfs(int s, int n)

1. Start.
2. Declare p, i
3. Call enqueue(s)
4. Set vis[s] = 1
5. Call p = dequeue()
6. If (p != 0)

Display p

1. While (p != 0)

For i=1 to n

if ((a[p][i] != 0) && (vis[i] == 0)) call enqueue(i)

set vis[i] = 1 Call p = dequeue() If (p != 0)

Display p

1. For i=1 to n

If (vis[i] == 0) Call bfs(i, n)

1. Exit.

## void enqueue(int item)

1. Start.
2. If ((rear == 19))

Display “Queue is full”

1. Else

If (rear == -1)

Set q[++rear] = item Set front=front+1

Else

Set q[++rear] = item

1. Exit.

## int dequeue()

1. Start.
2. Declare k
3. If ((front > rear) || (front == -1)) Return k
4. Else

Set k = q[front++] Return k

1. Exit.

## void dfs(int s, int n)

1. Start.
2. Declare k, i
3. Call push(s)
4. Set vis[s] = 1
5. Call k = pop()
6. If (k != 0)

Display k

1. While (k != 0)

For i=1 to n

if ((a[k][i] != 0) && (vis[i] == 0)) call push(i)

set vis[i] = 1 Call k = pop()

If (k != 0)

Display k

1. For i=1 to n

If (vis[i] == 0) Call dfs(i, n)

1. Exit.

## void push(int item)

1. Start.
2. If ((top == 19))

Display “stack is full”

1. Else

Set stack[++top] = item

1. Exit.

## int pop()

1. Start.
2. Declare k
3. If (top == -1)

Go to step 5

1. Else

Set k = stack[top--] Return k

1. Exit.

**Program**

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

int q[20], top = -1, front = -1, rear = -1, a[20][20], vis[20], stack[20]; int dequeue();

void enqueue(int item); void bfs(int s, int n); void dfs(int s, int n); void push(int item);

int pop(); int main() {

int n, i, s, ch, j; char c, dummy;

printf("Enter the number of vertices:"); scanf("%d", &n);

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

printf("Enter 1 if %d has an adge with %d else 0: ", i, j); scanf("%d", &a[i][j]);

}}

printf("The Adjacency Matrix is\n"); for (i = 1; i <= n; i++) {

for (j = 1; j <= n; j++) { printf(" %d", a[i][j]);

}

printf("\n");

}

do {

for (i = 1; i <= n; i++) vis[i] = 0;

printf("\nMENU"); printf("\n1. B.F.S");

printf("\n2. D.F.S"); printf("\nEnter your choice: "); scanf("%d", &ch);

printf("Enter the source vertex: "); scanf("%d", &s);

switch (ch) {

case 1: bfs(s, n);

break;

case 1: dfs(s, n);

break;

}

printf("Do you want to continue (Y/N)? "); scanf(" %c", &c);

} while ((c == 'y') || (c == 'Y')); return 0;

}

void bfs(int s, int n) {

int p, i; enqueue(s);

vis[s] = 1;

p = dequeue(); if (p != 0)

printf(" %d", p); while (p != 0) {

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

if ((a[p][i] != 0) && (vis[i] == 0)) { enqueue(i);

vis[i] = 1;

}

p = dequeue(); if (p != 0)

printf(" %d ", p);

}

for (i = 1; i <= n;i++) if (vis[i] == 0)

bfs(i, n);

}

void enqueue(int item) { if (rear == 19)

printf("QUEUE FULL"); else {

if (rear == -1) { q[++rear] = item; front++;

} else

q[++rear] = item;

}}

int dequeue() { int k;

if ((front > rear) || (front == -1)) return 0;

else {

k = q[front++]; return k;

}}

void dfs(int s, int n) { int i, k;

push(s);

vis[s] = 1;

k = pop();

if (k != 0)

printf(" %d ", k); while (k != 0) {

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

if ((a[k][i] != 0) && (vis[i] == 0)) { push(i);

vis[i] = 1;

}

k = pop();

if (k != 0)

printf(" %d ", k);

}

for (i = 1; i <= n;i++) if (vis[i] == 0)

dfs(i, n);

}

void push(int item) { if (top == 19)

printf("Stack overflow"); else

stack[++top] = item;

}

int pop() { int k;

if (top == 1) return 0;

else {

k = stack[top--];

return k;

}}

**Output**

Enter the number of vertices: 4

Enter 1 if 1 has an adge with 1 else 0: 1

Enter 1 if 1 has an adge with 2 else 0: 1

Enter 1 if 1 has an adge with 3 else 0: 1

Enter 1 if 1 has an adge with 4 else 0: 1

Enter 1 if 2 has an adge with 1 else 0: 0

Enter 1 if 2 has an adge with 2 else 0: 0

Enter 1 if 2 has an adge with 3 else 0: 1

Enter 1 if 2 has an adge with 4 else 0: 1

Enter 1 if 3 has an adge with 1 else 0: 0

Enter 1 if 3 has an adge with 2 else 0: 0

Enter 1 if 3 has an adge with 3 else 0: 1

Enter 1 if 3 has an adge with 4 else 0: 0

Enter 1 if 4 has an adge with 1 else 0: 0

Enter 1 if 4 has an adge with 2 else 0: 1

Enter 1 if 4 has an adge with 3 else 0: 1

Enter 1 if 4 has an adge with 4 else 0: 0

The Adjacency Matrix 1 1 1 1

0 0 1 1

0 0 1 0

0 1 1 0

MENU

1. B.F.S
2. D.F.S

Enter your choice: 1 Enter the source vertex: 3 3 1 2 4

Do you want to continue (Y/N)? y

MENU

1. B.F.S
2. D.F.S

Enter your choice: 2 Enter the source vertex: 1 1 4 3 2

Do you want to continue (Y/N)? n

**Experiment 25 Date:06/01/2025**

# Prim’s Algorithm

**Aim:**

Program to implement Prim’s Algorithm for finding the minimum cost spanning

Tree.

**Algorithm:**

## main()

1. b1.Read n
2. int i=1,i<=n
3. for(int j=1;j<=n;j++)
4. read cost[i][j]
5. cost[i][j]==0
6. cost[i][j]=INF
7. visited[1]=1
8. while (no\_edges<n-1)
9. min=INF
10. a=0
11. b=0
12. i=1,i<=n
13. if(visited[i]==1)
14. j=1,j<=n
15. if (visited[j]==0 && cost[i][j]!=INF)
16. if (cost[i][j]<min)
17. min=cost[i][j];
18. a=i
19. b=j
20. no\_edges
21. visited[b]=1
22. total\_cost=total\_cost+min
23. Print total cost
24. exit

**Program**

#include<stdio.h>

#define INF 999

int cost[10][10],visited[10]={0,0,0,0,0,0,0,0,0,0};

int n,i,j,no\_edges=0,total\_cost=0,min,a,b;

int main()

{

printf("Enter the number of vertices : "); scanf("%d",&n);

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

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

{

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

{

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

if (cost[i][j]==0)

{

cost[i][j]=INF;

}}}

printf("\nThe minimum cost spanning tree edges are:\n"); visited[1]=1;

while (no\_edges<n-1)

{

min=INF;

a=0;

b=0;

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

{

if(visited[i]==1)

{

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

{

if (visited[j]==0 && cost[i][j]!=INF)

{

if (cost[i][j]<min)

{

no\_edges++;

min=cost[i][j]; a=i;

b=j;

}}

}}

}

visited[b]=1;

printf("%d-%d:%d\n",a,b,min);

total\_cost=total\_cost+min;

}

printf("Total cost : %d\n",total\_cost);

}

**Output**

Enter the number of vertices : 3

Enter the cost adjacency matrix : 1

0

0

0

1

1

1

1

1

The minimum cost spanning tree edges are:

0-0:999

0-0:999

Total cost : 1998

**Experiment 26 Date:06/01/2025**

# Kruskal’s Algorithm

**Aim:**

Program to implement Kruskal's algorithm using Disjoint sets.

**Algorithm:**

## main()

1. Start

2. Declare cost[10][10], visited[10]={0,0,0,0,0,0,0,0,0,0}, parent[10]

3. Declare n,i,j,no\_edges=0,total\_cost=0,min,a,b,u,v

4. Read n

5. Set i=1, i<=n

6. Set j=1,j<=n

7. Read cost[i][j]

8. if (cost[i][j] ==0)

9. cost[i][j] =INF

10. visited [1]=1

11. while (no\_edges<n-1)

12. min=INF

13. a=0

14. b=0

15. Set i=1, i<=n

16. Set j=1, j<=n

17. if (cost[i][j]<min)

18. min=cost[i][j]

19. a=u=i

20. b=v=j

21. u=find(u)

22. v=find(v)

23. if(uni(u,v))

24. Print a,b,min

25. no\_edges++

26. total\_cost=total\_cost+min

27. cost[a][b]=cost[b][a]=INF

28. Printtotal\_cost

29. Stop

## 

## find(int)

1. Start

2. while(parent[i])

3. i=parent[i]

4. return i

5. Stop

## unit(int ,int )

* 1. Start
  2. if(i!=j)
  3. parent[j]=i
  4. return 1
  5. return
  6. Stop

**Program**

#include<stdio.h>

#define INF 999

int cost[10][10],visited[10]={0,0,0,0,0,0,0,0,0,0},parent[10];

int n,i,j,no\_edges=0,total\_cost=0,min,a,b,u,v;

int find(int);

int uni(int,int);

int main()

{

printf("Enter the no. of vertices : ");

scanf("%d",&n);

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

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

{

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

{

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

if (cost[i][j]==0)

{

cost[i][j]=INF;

}}}

printf("\nCost of edges\n");

visited[1]=1;

while (no\_edges<n-1)

{

min=INF;

a=0;

b=0;

for(i=1;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-%d:%d\n",a,b,min);

no\_edges++;

total\_cost=total\_cost+min;

}

cost[a][b]=cost[b][a]=INF;

}

printf("Total cost : %d\n",total\_cost);

}

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;

}

**Output**

Enter the no. of vertices: 3

Enter the adjacency matrix:

1

0

0

1

1

1

0

0

1

Cost of edges

2-1:1

2-3:1

Total cost: 2

**Experiment 27 Date:08/01/2025**

# Single Source Shortest Path

**Aim:**

Program for single source shortest path algorithm using Dijkstras algorithm.

**Algorithm:**

## minDistance(int , bool )

1. Start
2. Set v = 0, v < V
3. if (sptSet[v] == false &&dist[v] <= min)
4. min = dist[v], min\_index = v
5. return min\_index
6. Stop

## printSolution( dist[])

1. Start
2. Set i = 0, i< V
3. Print i, dist[i]
4. Stop

## dijkstra(graph[V][V],src)

1. Start
2. declaredist[V]
3. declare sptSet[V]
4. declare i = 0; i< V; i++)
5. Set dist[i] = INT\_MAX, sptSet[i] = false
6. Set dist[src] = 0
7. Set count = 0, count < V - 1
8. Set u = minDistance(dist, sptSet)
9. Set sptSet[u] = true
10. Set v = 0, v< V
11. if!sptSet[v] && graph[u][v]
12. Set dist[u] != INT\_MAX
13. Set dist[u] + graph[u][v] <dist[v])
14. Set dist[v] = dist[u] + graph[u][v]
15. Print Solution(dist)
16. Stop

**Program**

#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

#define V 9

int minDistance(int dist[], bool sptSet[])

{

int min = INT\_MAX, min\_index;

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

if (sptSet[v] == false &&dist[v] <= min)

min = dist[v];

min\_index = v;

return min\_index;

}

void printSolution(int dist[])

{

printf("Vertex \t\t Distance from Source\n");

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

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

}

void dijkstra(int graph[V][V], int src)

{

int dist[V];

bool sptSet[V];

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

dist[i] = INT\_MAX;

sptSet[i] = false;

dist[src] = 0;

for (int count = 0; count < V - 1; count++)

{

int u = minDistance(dist, sptSet);

sptSet[u] = true;

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

if (!sptSet[v] && graph[u][v] &&dist[u] != INT\_MAX &&dist[u] + graph[u][v] <dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist);

}

int main()

{

int graph[V][V] = {

{ 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

{ 0, 0, 2, 0, 0, 0, 6, 7, 0 }

};

dijkstra(graph, 0);

return 0;

}

**Output**

|  |  |
| --- | --- |
| Vertex | Distance from Source |
| 0 | 0 |
| 1 | 4 |
| 2 | 12 |
| 3 | 19 |
| 4 | 21 |
| 5 | 11 |
| 6 | 9 |
| 7 | 8 |
| 8 | 14 |