**A blue and white logo

Description automatically generated Lab no: 3 Date:2024/01/26**

**Title: Write a program to show the basic operation of Queue.**

**Queue:**

A queue is a linear data structure in Data Structures and Algorithms (DSA) that follows the **First In, First Out (FIFO)** principle. This means that the first element added to the queue is the first one to be removed.

**Key Queue Operations:**

1. **Enqueue: Adds an element to the rear of the queue.**
   * **Example: Queue → [A, B],**

**Enqueue C → [A, B, C]**

1. **Dequeue: Removes the front element from the queue.**
   * **Example: Queue → [A, B, C],**

**Dequeue → [B, C] (removes A)**

1. **Front/Peek: Returns the front element without removing it.**
   * **Example: Queue → [A, B, C],**

**Peek → A**

1. **IsEmpty: Checks if the queue is empty.**
   * **Example: Queue → [A, B, C],**

**IsEmpty → false**

1. **Size: Returns the number of elements in the queue.**
   * **Example: Queue → [A, B, C],**

**Size → 3**

**Compiler:** DEV C++

**Language :** C

**Source Code:**

#include<iostream>

using namespace std;

class queue{

int\* arr;

int front;

int rear;

int n;

public:

queue(int size) //construct || initialize of queue

{

cout<<"Queue is Created"<<endl;

arr=new int[size];

front=-1;

rear=-1;

n=size;

}

void enqueue(int x){

if(rear == n-1){ //could have compare with n only but as rear start with 1 so we had to compare with n-1

cout<<"The Queue is full"<<endl;

}

else{

rear++;

arr[rear]=x;

if (front==-1)

{

front++;

}

}

}

void dequeue(){

if(front==-1 || front>rear) {

cout<<"The Queue is Empty"<<endl;

}

else{

front++;

}

}

void Display(){

if(front==-1 || front>rear) {

cout<<"The Queue is Empty"<<endl;

}

else{

cout<<"Here are the entered data:"<<endl;

for (int i=front; i < rear+1; i++)

{

cout<<"=>"<<arr[i]<<endl;

}

}

}

};

int main(){

int size;

int in,n;

cout<<"Programmer -Sarfraj Alam"<<endl;

cout << "Basic Operation of Stack" << endl;

cout << "Enter the size of stack\n=>";

cin >> size;

queue q(size);

while (true)

{

cout<<"Programmer -Sarfraj Alam"<<endl;

cout << "\nEnter your choice\n 1 for enqueue\n 2 for dequeue\n 3 for display\n 4 for exit\n\n=>"; //Menu for the operation

cin >> n;

switch (n)

{

case 1:

cout<<"Enter the value to set in queue: "; //For enqeue operation

cin>>in;

q.enqueue(in);

break;

case 2: //For dequeue Operation

q.dequeue();

break;

case 3: //For Display Operation

q.Display();

break;

case 4:

exit(0);

default:

cout<<"Inavalid choice. Please try again."<<endl;

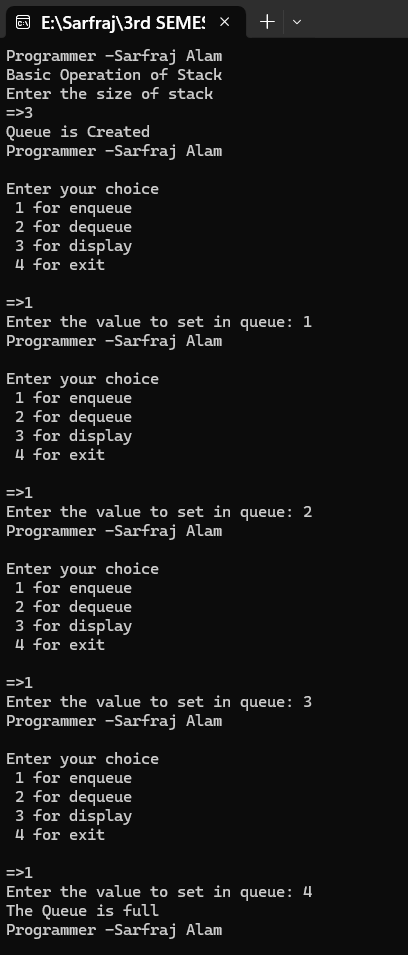
}

}

return 0;

}

**A screenshot of a computer program

Description automatically generatedOutput**

**A screenshot of a computer program

Description automatically generated**

**A blue and white logo

Description automatically generated Lab no: 4 Date:2024/01/26**

**Title: Write a program to switch between recursive programs**

**Recursion & Recursive Program:**

A recursive program is one that solves a problem by calling itself. In recursion, a function calls itself to divide a problem into smaller subproblems. Each recursive call processes a simpler version of the problem, and the base case is used to stop the recursion once the problem is simple enough to solve directly. Different type of recursive programs are :-

**Factorial :**

The factorial of a number n is the product of all positive integers less than or equal to n. It's defined as:

* n! = n \* (n-1) \* (n-2) \* ... \* 1
* Recursive case: n! = n \* (n-1)!

**Fibonacci Sequence:**

The Fibonacci sequence is defined as:

* F(0) = 0, F(1) = 1
* F(n) = F(n-1) + F(n-2) for n > 1

**GCD (Greatest Common Divisor)**

* **Euclidean Algorithm**: GCD of two numbers a and b is found using:
  + GCD(a, b) = GCD(b, a % b)

**Tower of Hanoi**

* 1. Move n-1 disks from source to auxiliary peg.
  2. Move the nth disk from source to target peg.
  3. Move the n-1 disks from auxiliary peg to target peg.

**Compiler:** DEV C++

**Language :** C++

**Source Code:**

#include<iostream>

using namespace std;

// Function prototypes

int fact(int);

int fib(int);

void TOH(int n, char source, char helper, char target);

int GCD(int, int);

int counter = 1;

int main() {

cout << "Programmer - Sarfraj Alam" << endl;

int choice;

while (true) {

cout << "\n\nMenu of Recursion\n";

cout << "1. Factorial\n";

cout << "2. Fibonacci Series\n";

cout << "3. Tower of Hanoi\n";

cout << "4. GCD\n";

cout << "Choose any other option to exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

int n;

cout << "Enter the number for factorial calculation: ";

cin >> n;

if (n < 0) {

cout << "Factorial is not defined for negative numbers." << endl;

} else {

cout << "The factorial of " << n << " is " << fact(n) << endl;

}

break;

}

case 2: {

int n;

cout << "Enter the number of terms for the Fibonacci series: ";

cin >> n;

if (n < 1) {

cout << "Please enter a positive integer greater than 0." << endl;

} else {

cout << "The first " << n << " terms of the Fibonacci series are: ";

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

cout << fib(i) << "\t"; // Corrected fib(1) to fib(i)

}

cout << endl;

}

break;

}

case 3: {

int nodisk;

cout << "Enter the number of disks for Tower of Hanoi: ";

cin >> nodisk;

if (nodisk < 1) {

cout << "Number of disks must be at least 1." << endl;

} else {

char a = 'A', b = 'B', c = 'C';

counter = 1; // Reset counter for each run

cout << "Steps to solve the Tower of Hanoi are:\n";

TOH(nodisk, a, b, c);

cout << "The minimum number of steps is: " << (1 << nodisk) - 1 << endl;

}

break;

}

case 4: {

int x, y;

cout << "Enter (x, y) to find GCD\n";

cout << "x = ";

cin >> x;

cout << "y = ";

cin >> y;

cout << "The GCD value of (" << x << ", " << y << ") = " << GCD(x, y) << endl;

break;

}

default:

cout << "Exiting the program. Goodbye!" << endl;

return 0;

}

}

}

// Factorial using recursion

int fact(int n) {

if (n == 0 || n == 1)

return 1;

else

return n \* fact(n - 1);

}

// Fibonacci using recursion

int fib(int n) {

if (n == 0)

return 0;

else if (n == 1)

return 1;

else

return fib(n - 1) + fib(n - 2);

}

// Tower of Hanoi using recursion

void TOH(int n, char source, char helper, char target) {

if (n == 1) {

cout << counter << ") Move disk 1 from rod " << source << " to rod " << target << endl;

counter++;

return;

}

TOH(n - 1, source, target, helper);

cout << counter << ") Move disk " << n << " from rod " << source << " to rod " << target << endl;

counter++;

TOH(n - 1, helper, source, target);

}

// GCD using recursion

int GCD(int a, int b) {

if (b == 0)

return a;

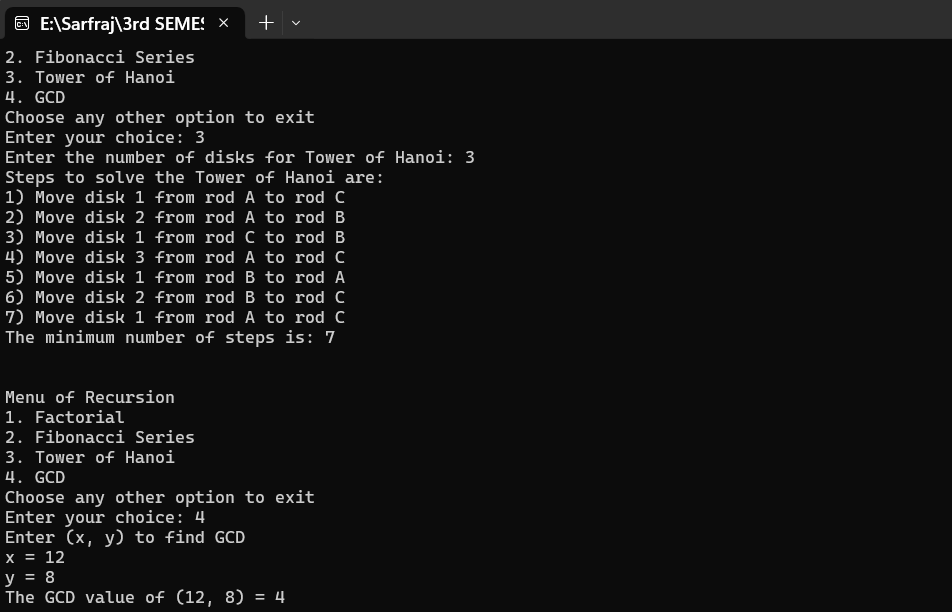
else

return GCD(b, a % b);

}

A screenshot of a computer

Description automatically generated

****