TASK 2

#include<iostream>

using namespace std;

struct node

{

int data;

node\* NEXT;

};

class queue

{

int size;

node\* temp;

node\* tail;

node\* heading;

public:

queue()

{

heading = NULL;

tail = NULL;

size = 0;

}

~queue()

{}

bool is\_empty()

{

if (heading == NULL)

{

cout << "Queue is empty \n";

return true;

}

else

{

cout << "Queue is not empty \n";

return 0;

}

};

void enqueue(int data)

{

temp = new node;

temp->data = data;

temp->NEXT = NULL;

if (heading == NULL)//for first time

{

tail = temp;

heading = temp;

}

else

{

tail->NEXT = temp;

tail = temp;

}

size++; //getting the size

};

void dequeue()

{

int del;

if (heading == NULL)

{

cout << "Queue is empty \n";

}

else

{

node\* ptr = heading;

del = ptr->data;

heading = heading->NEXT;

delete (ptr);

cout << "dequeue Successfully \n";

size--;

}

};

void display()

{

if (heading == NULL)

{

cout << "Queue is empty \n";

}

else

{

node\* ptr = heading;

cout << "Queue elements are: ";

while (ptr != NULL)

{

cout << ptr->data << " ";

ptr = ptr->NEXT;

}

cout << endl;

}

};

int return\_middle()

{

if (heading == NULL)

{

cout << "Queue is empty \n";

return 0;

}

else

{

int del;

for (int i = 0; i < (size / 2); i++)

{

node\* ptr = heading;

del = ptr->data;

heading = heading->NEXT;

delete (ptr);

}

return heading->data;

}

}

void sumofqueue()

{

int sum = 0;

node\* ptr=heading;

while (ptr!=NULL)

{

sum += ptr->data;

ptr = ptr->NEXT;

}

cout <<"The sum of the queue is :" <<sum<<endl;

}

int sizeofqueue()

{

int sum = 0,count=0;

node\* ptr = heading;

while (ptr != NULL)

{

ptr = ptr->NEXT;

count++;

}

return count;

cout << endl;

}

void smallest()

{

node\* ptr = heading;

node\* ptr1 = heading;

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

{

if ((ptr1->data) > (ptr->data))

{

ptr1->data = ptr->data;

}

ptr = ptr->NEXT;

}

cout << ptr1->data;

}

int largest()

{

node\* ptr = heading;

node\* ptr1 = heading;

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

{

if ((ptr1->data) < (ptr->data))

{

ptr1->data = ptr->data;

}

ptr = ptr->NEXT;

}

return ptr1->data;

}

void primeitems()

{

node\* ptr = heading;

int f, k = 0;

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

{

f = 0;

for (int j = 2; j < (ptr->data) ; j++)

{

if (ptr->data % j == 0)

{

f = 1;

break;

}

}

if (f == 0)

{

cout << ptr->data << " is a prime number " << endl;

}

ptr = ptr->NEXT;

}

}

};

int main()

{

int n;

queue access;

a: cout << "1.check if queue is empty :";

cout<<"\n2. enter data in queue :";

cout << "\n3. dequeue data :";

cout << "\n4. display the data :";

cout << "\n5. Return the middle data :";

cout << "\n6. sum the queue :";

cout<<"\n7.size of the queue :";

cout << "\n8.smallest number of the queue :";

cout << "\n9.largest number of the queue :";

cout << "\n10.Pime number of the queue :";

cout << "\n11. to exit :" << endl;

cout << "Enter your choice :";

cin >> n;

//Checking queue is empty or not

if (n == 1)

{

access.is\_empty();

goto a;

}

//adding elements in queue

if (n == 2)

{

int data, n;

cout << "Enter the number of element you add in queue" << endl;

cin >> n;

cout << "ENTER THE DATA You WANT TO ADD IN QUEUE \n";

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

{

cin >> data;

access.enqueue(data);

}

goto a;

}

//deleting elements in queue

if (n == 3)

{

cout << endl;

access.dequeue();

goto a;

}

//displaying elements of queue

if (n == 4)

{

cout << endl;

access.display();

goto a;

}

//middle elements in queue

if (n == 5)

{

cout << endl;

cout << access.return\_middle() << " is the middle of the queue \n";

goto a;

}

//sum of elements of queue

if (n == 6)

{

cout << endl;

access.sumofqueue();

}

//size of queue

if (n == 7)

{

cout << endl;

cout << "The size of the queue is :";

access.sizeofqueue();

cout << endl;

}

//smallest elements in queue

if (n == 8)

{

cout << endl;

cout << "The smallest number is :";

access.smallest();

cout << endl;

}

//largest elements in queue

if (n == 9)

{

cout << "The largest number is :";

cout<<access.largest();

cout << endl;

}

//prime elements in queue

if (n == 10)

{

cout << endl;

access.primeitems();

cout << endl;

}

if (n == 11)

{

goto z;

}

if (n > 11 && n < 1)

{

cout << "WRONG INPUT \n";

goto a;

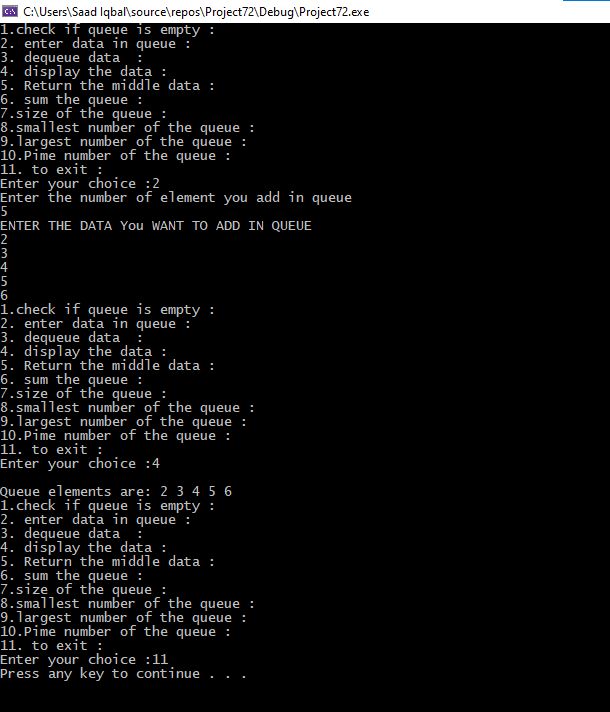
}

z:

system("pause");

return 0;

}



TASK 1:

#include<iostream>

#include<stack>

using namespace std;

struct node

{

string data;

node\* NEXT;

};

class stack1

{

int size;

node\* heading;

public:

stack1()

{

size = 0;

heading = NULL;

}

bool is\_empty()

{

if (heading == NULL)//check empty

{

cout << "Stack is empty \n";

return 1;

}

else

{

cout << "Stack is not empty \n";

return 0;

}

};

void push(string data)

{

node\* temp = new node;

temp->data = data;

temp->NEXT = NULL;

if (heading == NULL)

{

heading = temp;

}

else

{

temp->NEXT = heading;

heading = temp;

}

size++;

};

void pop()

{

if (heading == NULL)

{

cout << "Stack is empty \n";

}

else

{

string p = heading->data;

node\* pop = heading;

heading = heading->NEXT;

delete pop;

}

size--;

};

void display()

{

if (heading == NULL)

{

cout << "stack is empty \n ";

}

else

{

node\* display = heading;

cout << "STACK ELEMENTS ARE : ";

while (display != NULL)

{

cout << display->data << " ";

display = display->NEXT;

}

}

cout << "\n";

}

void checkexp(string st,int n)

{

bool a = false;

stack<int> s;

node\* ptr = heading;

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

{

if (st[i] == '(' || st[i] == '[' || st[i] == '{')

s.push(st[i]);

else

{

if (!s.empty())

{

char temp = s.top();

s.pop();

if (st[i] == ')' && temp != '(')

{

a = true;

break;

}

if (st[i] == ']' && temp != '[')

{

a = true;

break;

}

if (st[i] == '}' && temp != '{')

{

a = true;

break;

}

}

else

{

a = true;

break;

}

}

}

if (!s.empty())

{

a = true;

}

if (a)

{

cout << "Not balanced"<<endl;

}

else

{

cout << " balanced"<<endl;

}

}

};

int main()

{

int n = 0, n1;

stack1 access;

string data;

a: cout << " \n1.check if stack is empty.\n 2. enter data in stack \n 3. pop data in stack \n 4. display the data \n 5. to check whether it is balance\n6.to exit"<<endl;

cout << "Enter your choice :";

cin >> n;

if (n == 1)

{

access.is\_empty();

goto a;

}

if (n == 2)

{

cout << "ENTER THE DATA TO WANT TO ADD IN STACK \n";

cin >> data;

access.push(data);

cout << "DATA ADDED SUCCESSFULLY \n";

goto a;

}

if (n == 3)

{

cout << "Successfully pop ";

access.pop();

goto a;

}

if (n == 4)

{

access.display();

goto a;

}

if (n == 5)

{

int N = data.length();

access.checkexp(data,N);

goto a;

}

if (n == 5)

{

goto z;

}

if (n > 5 && n < 1)

{

cout << "WRONG INPUT \n";

goto a;

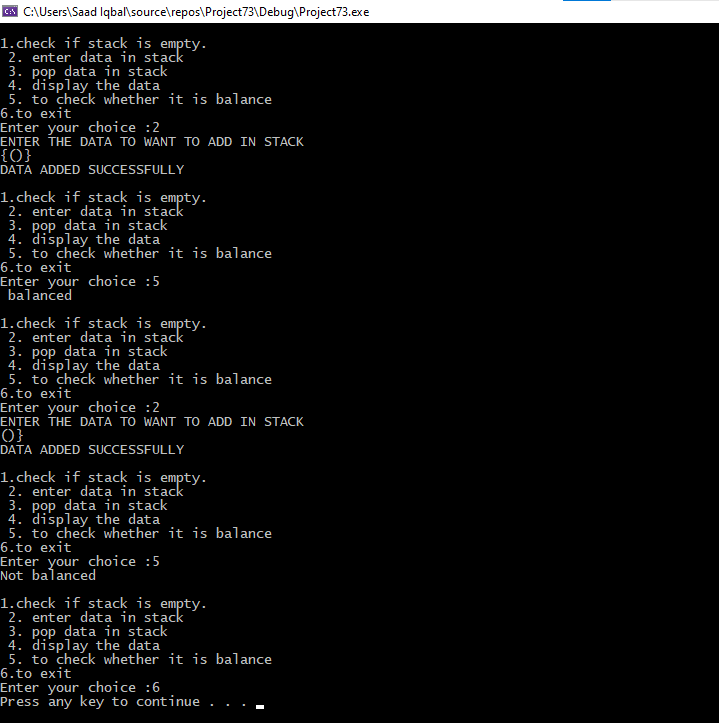
}

z:

system("pause");

return 0;

}



TASK 3:

#include<iostream>

using namespace std;

class queue

{

public:

int data[200], front, rear, size;

queue(){}

queue(int x)

{

front = -1;

rear = -1;

size = x;

}

void enqueue(int x)

{

if ((front == 0 && rear == size - 1) ||

(rear == (front - 1) % (size - 1)))

{

printf("\nQueue is Full");

return;

}

else if (front == -1) /\* Insert First Element \*/

{

front = rear = 0;

data[rear] = x;

}

else if (rear == size - 1 && front != 0)

{

rear = 0;

data[rear] = x;

}

else

{

rear++;

data[rear] = x;

}

}

int dequeue()

{

if (is\_empty())

{

return 0;

}

else if (front == rear)

{

int a = data[front];

front = rear = -1;

return a;

}

else

{

int a = data[front];

front = (front + 1) % size;

return a;

}

}

bool is\_empty()

{

if (front == -1)

return 1;

else

return 0;

}

int is\_full()

{

if ((rear + 1) % size == front)

{

cout << "queue is full" << endl;

return 1;

}

else

{

return 0;

}

}

void print()

{

int a = front;

do

{

cout << data[a] << " ";

a = (a + 1) % size;

} while (a != (rear + 1) % size);

cout << endl;

}

};

int main()

{

int a, num,l;

queue q1;

if (q1.front <= -1)

{

cout << "queue is empty"<<endl;

}

cout << "Please Enter the number of elements : ";

cin >> num;

queue q(num);

cout << "Please enter element in queue " << endl;

while (num--)

{

cin >> a;

q.enqueue(a);

}

cout << "Elements in the queue :" << endl;

q.print();

q.dequeue();

q.print();

if (q.is\_full())

{

cout << "queue is full" << endl;

}

if (q.is\_empty() == 1)

{

cout << "queue is empty"<<endl;

}

else

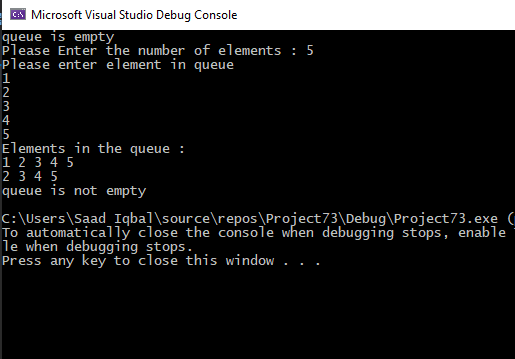
{

cout << "queue is not empty"<<endl;

}

return 0;

}



TASK 4:

#include<iostream>

using namespace std;

struct node

{

int data;

node\* NEXT;

int priority;

};

class queue

{

int size;

node\* heading;

public:

queue()

{

size = 0;

}

~queue()

{}

bool is\_empty()

{

if (heading == NULL)

{

cout << "Queue is empty \n";

return true;

}

else

{

cout << "Queue is not empty \n";

return 0;

}

};

void enqueue(int data,int p)

{

{

node \* q;

node \*temp = new node;

temp->data = data;

temp->priority = p;

if (heading== NULL || p < heading->priority)

{

temp->NEXT = heading;

heading = temp;

}

else

{

q = heading;

while (q->NEXT != NULL && q->NEXT->priority <= p)

q = q->NEXT;

temp->NEXT = q->NEXT;

q->NEXT = temp;

}

}

};

void dequeue()

{

int del;

node\* ptr = heading;

if (heading == NULL)

{

cout << "Queue is empty \n";

}

else

{

cout << "dequeued successfully"<<endl;

heading = heading->NEXT;

delete ptr;

}

};

//displaying elements of queue

void display()

{

node\* ptr;

ptr = heading;

if (heading == NULL)

{

cout << "Queue is empty\n";

}

else

{

cout << "Priority Item\t";

cout << "Queue is \n";

while (ptr != NULL)

{

cout << ptr->priority << "\t\t" << ptr->data << endl;

ptr = ptr->NEXT;

}

}

}

//Searching elements of queue

void search(int x)

{

node\* temp = heading;

while (temp->NEXT != NULL)

{

if (temp->data == x)

{

cout << "\nNumber has been found" << endl;

break;

}

else

{

cout << "\nNumber has not been found" << endl;

break;

}

temp = temp->NEXT;

}

}

//Updating elements of queue

void update(int x)

{

node\* temp = heading;

int update;

while (temp->NEXT != NULL)

{

if (temp->data == x)

{

cout << "\nNumber has been found" << endl;

cout << "Enter the number " << endl;

cin >> update;

temp->data = update;

break;

}

else

{

cout << "\nNumber has not been found" << endl;

break;

}

temp = temp->NEXT;

}

}

};

int main()

{

int n;

queue access;

a: cout << "1.check if queue is empty :";

cout << "\n2. enter data in queue :";

cout << "\n3. dequeue data :";

cout << "\n4. display the data :";

cout << "\n5. Searching the data :";

cout << "\n6. updating the data :";

cout << "\n7. to exit :" << endl;

cout << "Enter your choice :";

cin >> n;

//Checking queue is empty or not

if (n == 1)

{

access.is\_empty();

goto a;

}

//adding elements in queue

if (n == 2)

{

int data, n,p;

cout << "Enter the number of element you add in queue" << endl;

cin >> n;

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

{

cout << "ENTER THE DATA You WANT TO ADD IN QUEUE "<<i+1<<" :";

cin >> data;

cout << "ENTER THE priority number IN QUEUE " << i + 1 << " :";

cin >> p;

access.enqueue(data,p);

}

goto a;

}

//deleting elements in queue

if (n == 3)

{

cout << endl;

access.dequeue();

goto a;

}

//displaying elements of queue

if (n == 4)

{

cout << endl;

access.display();

goto a;

}

//Searching elements of queue

if (n == 5)

{

cout << endl;

int a;

cout << "Enter number you want to search :";

cin >> a;

access.search(a);

goto a;

}

//Searching and updating elements of queue

if (n == 6)

{

int a;

cout << "Enter number you want to search and update :";

cin >> a;

access.update(a);

goto a;

}

if (n == 7)

{

goto z;

}

if (n > 7 && n < 1)

{

cout << "WRONG INPUT \n";

goto a;

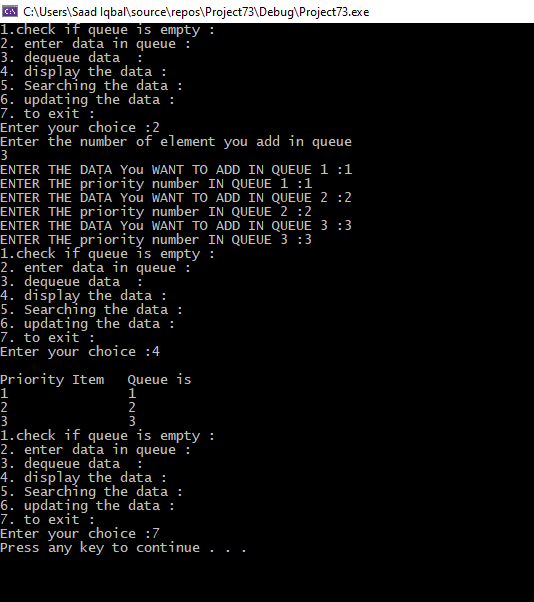
}

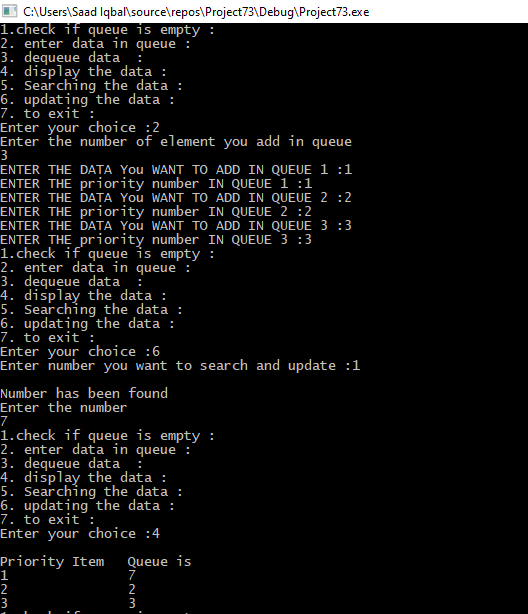
z:

system("pause");

return 0;

}





TASK 5:

#include<iostream>

#include<queue>

using namespace std;

struct node

{

int data;

node\* NEXT;

int data1;

};

class queue1

{

int size;

node\* heading;

public:

queue1()

{

size = 0;

}

~queue1()

{}

bool is\_empty()

{

if (heading == NULL)

{

cout << "Queue is empty \n";

return true;

}

else

{

cout << "Queue is not empty \n";

return 0;

}

};

void enqueue(int callingorder,int idealorder)

{

{

node\* q;

node\* temp = new node;

temp->data = callingorder;

temp->data1 = idealorder;

if (heading == NULL || idealorder < heading->data1)

{

temp->NEXT = heading;

heading = temp;

}

else

{

q = heading;

while (q->NEXT != NULL)

q = q->NEXT;

temp->NEXT = q->NEXT;

q->NEXT = temp;

}

}

};

void dequeue()

{

int del;

node\* ptr = heading;

if (heading == NULL)

{

cout << "Queue is empty \n";

}

else

{

cout << "dequeued successfully" << endl;

heading = heading->NEXT;

delete ptr;

}

};

//displaying elements of queue

void display()

{

node\* ptr;

ptr = heading;

if (heading == NULL)

{

cout << "Queue is empty\n";

}

else

{

cout << "ideal Order\t";

cout << "calling order \n";

while (ptr != NULL)

{

cout << ptr->data1 << "\t\t" << ptr->data << endl;

ptr = ptr->NEXT;

}

}

}

int monkPowerTime(int n, int callingorder[], int idealorder[])

{

queue <int> q;

int count = 0;

int i = 0;

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

{

q.push(callingorder[i]);

}

while (!q.empty())

{

int x = q.front();

q.pop();

if (x == idealorder[i])

{

i++;

count++;

}

else

{

q.push(x);

count++;

}

}

return count;

}

};

int main()

{

int n;

queue1 access;

a: cout << "1.check if queue is empty :";

cout << "\n2. enter data in queue :";

cout << "\n3. dequeue data :";

cout << "\n4. display the data :";

cout << "\n5. display the Totol time taken :";

cout << "\n5. to exit :" << endl;

cout << "Enter your choice :";

cin >> n;

//Checking queue is empty or not

if (n == 1)

{

access.is\_empty();

goto a;

}

//adding elements in queue

if (n == 2)

{

int data, data1;

cout << "Enter the number of process" << endl;

cin >> n;

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

{

cout << "Enter data for calling order :";

cin >> data;

cout << "Enter data for ideal order :";

cin >> data1;

access.enqueue(data, data1);

}

goto a;

}

//deleting elements in queue

if (n == 3)

{

cout << endl;

access.dequeue();

goto a;

}

//displaying elements of queue

if (n == 4)

{

cout << endl;

access.display();

goto a;

}

if (n == 5)

{

const int n = 3;

int calling[n] = { 3, 2, 1 };

int ideal[n] = { 1, 3, 2 };

cout << endl;

cout << "The total time taken is :";

cout<<access.monkPowerTime(n, calling, ideal);

cout << endl;

goto a;

}

if (n == 6)

{

goto z;

}

if (n > 6 && n < 1)

{

cout << "WRONG INPUT \n";

goto a;

}

z:

system("pause");

return 0;

}

