**Question#1**

**Code:**

#include<iostream>

#include<ctime>

#include<cstdlib>

using namespace std;

struct node {

int data;

node\* next;

node() :data(0), next(nullptr) {};

};

class BaseSingly {

protected:

int count;

node\* head;

node\* current;

node\* tail;

public:

BaseSingly() {

count = 0;

head = nullptr;

tail = nullptr;

current = nullptr;

}

virtual void insert\_at\_front(int val) = 0;

virtual void insert\_at\_end(int val) = 0;

virtual void insert\_at\_index(int index, int val) = 0;

virtual void delete\_at\_front() = 0;

virtual void print() = 0;

};

class singlylist : public BaseSingly {

public:

singlylist() :BaseSingly() {}

~singlylist() {

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

//delete\_at\_end();

}

}

void insert\_at\_front(int val) {

if (head == nullptr) {

head = new node;

current = head;

head->data = val;

tail = head;

count++;

}

else {

node\* temp = new node;

temp->data = val;

temp->next = head;

head = temp;

count++;

}

}

void insert\_at\_end(int val) {

if (head == nullptr) {

insert\_at\_front(val);

}

else {

current = head;

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

current = current->next;

}

node\* temp = new node;

temp->data = val;

current->next = temp;

count++;

}

tail = tail->next;

}

void insert\_at\_index(int index, int val) {

if (index == 1) {

insert\_at\_front(val);

}

else if (index == count) {

insert\_at\_end(val);

}

else if (index < count && index > 1) {

current = head->next;

for (int i = 2; i < index - 1; i++) {

current = current->next;

}

node\* temp = new node;

temp->data = val;

temp->next = current->next;

current->next = temp;

count++;

}

else {

cout << "Invalid index" << endl;

}

}

void delete\_at\_front() {

current = head;

head = current->next;

delete current;

current = head;

count--;

}

void print() {

current = head;

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

cout << "[" << current->data << "]";

current = current->next;

}

cout << endl;

}

};

class baseQueue: public singlylist {

private:

//singlylist obj;

public:

virtual void EnQueue(int val) = 0;

virtual int DeQueue() = 0;

virtual int Peek() = 0;

};

class Queue :public baseQueue {

public:

void EnQueue(int val) {

if (head == nullptr) {

insert\_at\_front(val);

}

else if (val <= tail->data && tail != nullptr) {

insert\_at\_end(val);

}

else if (val >= head->data) {

insert\_at\_front(val);

}

else if (val>=tail->data && val<=head->data) {

node \*ptr = head->next;

int index = 2;

if (val <= head->data && val >= ptr->data) {

node\* temp = new node;

temp->next = ptr;

head->next = temp;

temp->data = val;

count++;

return;

}

for (int i = 2; i < count - 1; i++) {

if (ptr->data >= val && ptr->next->data <= val) {

insert\_at\_index(index, val);

}

index++;

ptr = ptr->next;

}

}

}

int DeQueue() {

int temp = head->data;

delete\_at\_front();

return temp;

}

int Peek() {

return head->data;

}

};

int main() {

Queue obj;

srand(time(0));

for (int i = 1; i < 6; i++) {

int val = rand() % 10 + 1;

obj.EnQueue(val);

//obj.print();

}

cout << "Print : ";

obj.print();

cout << "obj.DeQueue() : " << obj.DeQueue() << endl;

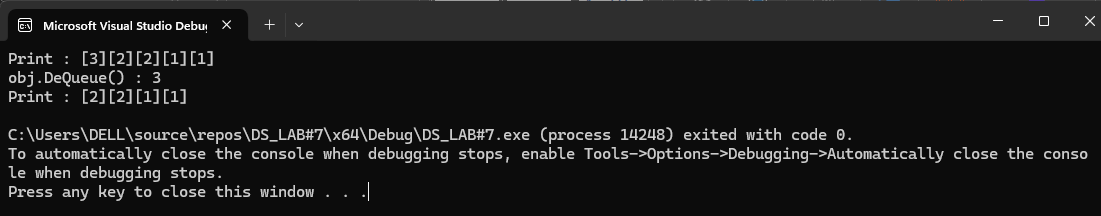
cout << "Print : ";

obj.print();

return 0;

}

**Output:**

****

**Question#3**

**Code:**

#include <iostream>

#include <string>

using namespace std;

struct Node {

string type;

int priority;

Node\* next;

Node(string t, int p) : type(t), priority(p), next(nullptr) {}

};

class PriorityQueue {

public:

Node\* head;

public:

PriorityQueue() : head(nullptr) {}

void push(string type, int priority) {

Node\* newNode = new Node(type, priority);

if (!head || priority < head->priority) {

newNode->next = head;

head = newNode;

return;

}

Node\* current = head;

while (current->next && current->next->priority <= priority) {

current = current->next;

}

newNode->next = current->next;

current->next = newNode;

}

void pop() {

if (!head) {

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

return;

}

Node\* temp = head;

head = head->next;

delete temp;

}

Node\* top() {

return head;

}

bool isEmpty() {

return head == nullptr;

}

~PriorityQueue() {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

};

class MultiLevelQueue {

private:

PriorityQueue SQ, IQ, IEQ, BQ, UQ;

public:

void addProcess(string type, int priority) {

if (type == "System")

SQ.push(type, priority);

else if (type == "Interactive")

IQ.push(type, priority);

else if (type == "Interactive\_Edit")

IEQ.push(type, priority);

else if (type == "Batch")

BQ.push(type, priority);

else if (type == "User")

UQ.push(type, priority);

else

cout << "Invalid Process Type" << endl;

}

void removeTopProcess(string type) {

if (type == "System")

SQ.pop();

else if (type == "Interactive")

IQ.pop();

else if (type == "Interactive\_Edit")

IEQ.pop();

else if (type == "Batch")

BQ.pop();

else if (type == "User")

UQ.pop();

else

cout << "Invalid Process Type" << endl;

}

void displayTopProcess(string type) {

Node\* top = nullptr;

if (type == "System")

top = SQ.top();

else if (type == "Interactive")

top = IQ.top();

else if (type == "Interactive\_Edit")

top = IEQ.top();

else if (type == "Batch")

top = BQ.top();

else if (type == "User")

top = UQ.top();

else {

cout << "Invalid Process Type" << endl;

return;

}

if (top) {

cout << "Top " << type << " Process : Priority = " << top->priority << endl;

}

else

cout << "No " << type << " Processes in Queue" << endl;

}

bool isQueueEmpty(string type) {

if (type == "System")

return SQ.isEmpty();

else if (type == "Interactive")

return IQ.isEmpty();

else if (type == "Interactive\_Edit")

return IEQ.isEmpty();

else if (type == "Batch")

return BQ.isEmpty();

else if (type == "User")

return UQ.isEmpty();

else {

cout << "Invalid Process Type" << endl;

return true;

}

}

};

int main() {

MultiLevelQueue obj;

obj.addProcess("System", 1);

obj.addProcess("Interactive", 2);

obj.addProcess("Interactive\_Edit", 3);

obj.addProcess("Batch", 4);

obj.addProcess("User", 5);

obj.addProcess("System", 0);

obj.addProcess("Interactive", 1);

cout << "Initial State :";

obj.displayTopProcess("System");

obj.displayTopProcess("Interactive");

obj.displayTopProcess("Interactive\_Edit");

obj.displayTopProcess("Batch");

obj.displayTopProcess("User");

obj.removeTopProcess("System");

obj.removeTopProcess("Interactive");

cout << "After Removing Top System And Interactive Processes : ";

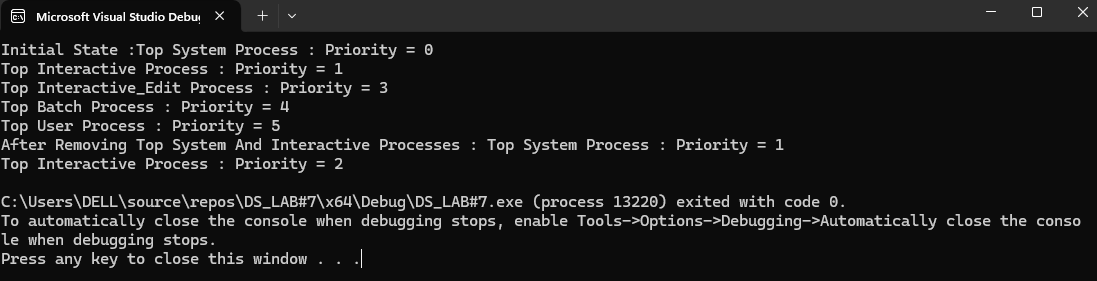
obj.displayTopProcess("System");

obj.displayTopProcess("Interactive");

return 0;

}

**Output:**

****

**Question#4**

**Code:**

**(a):**

#include<iostream>

using namespace std;

bool isPrime(int n, int i ,bool &flag) {

if (n <= 1) {

return false;

}

if (n == 2 || n == 3) {

return true;

}

if (n % i == 0) {

flag = false;

return false;

}

if (i == n / 2 && flag == true) {

return true;

}

return isPrime(n, i + 1,flag);

}

int main() {

int i = 3;

bool flag = true;

cout << "Prime: " << isPrime(12, i, flag) << endl;

return 0;

}

**Output:**

**(a):**

**A screenshot of a computer

Description automatically generated**