Bài 1: Cài đặt Queue bằng mảng

#include <iostream>  
#include <cstdlib>  
using namespace std;  
  
template <**class** Data>  
**class** Queue{

**private**:

Data \*p;  
 **int** head, tail, size, lengthQueue;  
**public**:  
 Queue(**int** size = 200){  
 **this**->size = size;  
 p = **new** Data[size];  
 lengthQueue = 0;  
 head = 0;  
 tail = -1;  
 }  
 virtual ~Queue(){  
 delete []p;  
 }  
 bool empty(){  
 **return** lengthQueue == 0;  
 }  
 **int** length(){  
 **return** lengthQueue;  
 }  
 **void** enQueue(**const** Data& a){  
 **if**(lengthQueue < size){  
 tail = (tail + 1)%size;  
 p[tail] = a;  
 }**else** {  
 *// trong truong hop mang day ta tao mang moi voi kich thuoc gap 2 lan  
 //sao chep du lieu cu sang va them phan tu moi vao* Data \*newData = **new** Data[size \* 2];  
 **for**(**int** i = 0; i < lengthQueue; i++){  
 newData[i] = p[(head + i)%size];  
 }  
 newData[size] = a;  
 delete []p;  
 p = newData;  
 head = 0;  
 tail = size;  
 size = size\*2;  
 }  
 lengthQueue++;  
 }  
 Data deQueue(){  
 **if**(lengthQueue > 0){  
 Data remove = p[head];  
 **if**(lengthQueue == 1){  
 *// neu chi co 1 phan tu -> khong co phan tu phia sau nen dat lai head = 0, tail = -1* head = 0;  
 tail = -1;  
 }  
 **else**{  
 head = (head + 1)%size;  
 }  
 lengthQueue--;  
 **return** remove;  
 } **else**{  
 cout << **"Queue is empty"**;  
 exit(1);  
 }  
 }  
 Data getHead(){  
 **if**(lengthQueue > 0){  
 **return** p[head];  
 } **else**{  
 cout << **"Queue is empty"**;  
 exit(1);  
 }  
 }  
};  
*// Test***int** main(){  
 Queue<**int**> demo(21);  
 **for**(**int** i = 1; i <= 20; i++){  
 demo.enQueue(i);  
 }  
 cout << **"length = "** << demo.length() << endl;  
 **for**(**int** i = 0; i < 20; i++){  
 cout << demo.deQueue() << **" "**;  
 }  
 cout << endl;  
 Queue<**char**> demo2;  
 **if**(demo2.empty()) cout << **"Demo2 is empty"** << endl;  
 demo2.enQueue(**'h'**);  
 demo2.enQueue(**'e'**);  
 demo2.enQueue(**'l'**);  
 demo2.enQueue(**'l'**);  
 demo2.enQueue(**'o'**);  
 **if**(!demo2.empty()) cout << **"Demo2 is not empty after enQueue"** << endl;  
 **while**(!demo2.empty())  
 cout << demo2.deQueue() << endl;  
 }

Tính độ phức tạp:

1. emtpy

bool empty(){  
 **return** lengthQueue == 0;

}

Độ phức tạp là O(1);

2.length

**int** length(){  
 **return** lengthQueue;  
}

Độ phức tạp là O(1);

3.enQueue

**void** enQueue(**const** Data& a){  
 **if**(lengthQueue < size){  
 tail = (tail + 1)%size;  
 p[tail] = a;  
 }**else** {  
 *// trong truong hop mang day ta tao mang moi voi kich thuoc gap 2 lan  
 //sao chep du lieu cu sang va them phan tu moi vao* Data \*newData = **new** Data[size \* 2];  
 **for**(**int** i = 0; i < lengthQueue; i++){  
 newData[i] = p[(head + i)%size];  
 }  
 newData[size] = a;  
 delete []p;  
 p = newData;  
 head = 0;  
 tail = size;  
 size = size\*2;  
 }  
 lengthQueue++;  
 }  
Trong trường hợp độ dài của Queue vẫn còn nhỏ hơn size thì độ phức tạp là O(1)

Trong trường hợp độ dài của Queue hiện tại không nhỏ hơn size (= size), bây giờ không đủ bộ nhớ để thêm phần tử mới nên ta khởi tạo ra một mảng động với kích thước bằng 2 lần kích thước ban đầu, và sao chép toàn bộ dữ liệu cũ sang, sau đó thêm phần tử mới vào

Data \*newData = **new** Data[size \* 2];  
 **for**(**int** i = 0; i < lengthQueue; i++){ // (1)  
 newData[i] = p[(head + i)%size]; // (2)  
 }  
 newData[size] = a; // (3)  
 delete []p; // (4)  
 p = newData; // (5)  
 head = 0; // (6)  
 tail = size; // (7)  
 size = size\*2; // (8)

Từ dòng 1 đến dòng 2, độ phức tạp là O(lengthQueue) = O(size) // vi luc này lengthQueue = size; trong đó size là giá trị độ dài ước lượng của Queue sao cho phù hợp với yêu cầu đề bài;

Các dòng còn lại đều có độ phức tạp là O(1)

=> Do đó độ phức tạp trong trường hợp lengthQueue == size là O(size) = O(n)

Vậy độ phức tạp trong trường hợp này là O(n)

4. deQueue

Data deQueue(){  
 **if**(lengthQueue > 0){  
 Data remove = p[head];  
 **if**(lengthQueue == 1){  
 *// neu chi co 1 phan tu -> khong co phan tu phia sau nen dat lai head = 0, tail = -1* head = 0;  
 tail = -1;  
 }  
 **else**{  
 head = (head + 1)%size;  
 }  
 lengthQueue--;  
 **return** remove;  
 } **else**{  
 cout << **"Queue is empty"**;  
 exit(1);  
 }  
 }  
Dễ thấy độ phức tạp là O(1);

5. getHead

Data getHead(){  
 **if**(lengthQueue > 0){  
 **return** p[head];  
 } **else**{  
 cout << **"Queue is empty"**;  
 exit(1);  
 }  
 }

Dễ thấy độ phức tạp là O(1)

Bài 2: Cài đặt Queue bằng danh sách liên kết:

#include <iostream>  
#include <cstdlib>  
  
using namespace std;  
  
template<**class** T>  
**class** Node{  
**private**:  
 T data;  
 Node<T>\* link;

**public**:  
 Node(T data1, Node<T>\* link1) : data(data1), link(link1){}  
 Node<T>\* getLink(){  
 **return** link;  
 }  
 T getData(){  
 **return** data;  
 }  
 **void** setLink(Node<T>\* link1){link = link1;}  
 **void** setData(**const** T& data1){data = data1;}  
};  
  
template<**class** T>  
**class** Queue{  
**private**:  
 Node<T> \*head;  
 Node<T> \*tail;  
 **int** lengthQueue;  
**public**:  
 Queue() : head(NULL), tail(NULL), lengthQueue(0){}  
 virtual ~Queue(){  
 **while**(!empty()){  
 deQueue();  
 }  
 }  
 bool empty(){  
 **return** lengthQueue == 0;  
 }  
 **int** length(){  
 **return** lengthQueue;  
 }  
 **void** enQueue(T a){  
 **if**(empty()){  
 head = tail = **new** Node<T>(a, NULL);  
 }  
 **else**{  
 tail->setLink(**new** Node<T>(a, NULL));  
 tail = tail->getLink();  
 }  
 lengthQueue++;  
 }  
 T deQueue(){  
 **if**(empty()){  
 cout << **"The Queue is empty"**;  
 exit(1);  
 }  
 **else**{  
 T temp = head->getData();  
 Node<T>\* p = head;  
 head = head->getLink();  
 **if**(head == NULL) tail = NULL;  
 delete p;  
 lengthQueue--;  
 **return** temp;  
 }  
 }  
 T getHead(){  
 **if**(empty()){  
 cout << **"The Queue is empty"**;  
 exit(1);  
 }**else**{  
 **return** head->getData();  
 }  
 }  
};  
  
*// Test***class** Integer{  
**private**:  
 **int** value;  
**public**:  
 Integer(**int** value1 = 0){  
 value = value1;  
 }  
 **int** getValue(){  
 **return** value;  
 }  
 **int** setValue(){  
 **return** value;  
 }  
 friend ostream& operator<< (ostream& os, Integer temp){  
 os << temp.value;  
 **return** os;  
 }  
  
};  
  
**int** main(){  
 Queue<Integer> p;  
 **if**(p.empty()) cout << **"p is empty"** << endl;  
  
 cout << **"length = "** << p.length() << endl;  
  
 **for**(**int** i = 0; i < 20; i++){  
 p.enQueue(\*(**new** Integer(i)));  
 }  
 cout << **"head = "** << p.getHead() << endl;  
 cout << **"length = "** << p.length() << endl;  
  
 **for**(**int** i = 0; i < 20; i++){  
 cout << p.deQueue() << **" "**;  
 }  
  
 cout << **"length = "** << p.length() << endl;

}

Tính độ phức tạp:

bool empty(){ // Độ phức tạp là O(1)  
 **return** lengthQueue == 0;  
 }  
 **int** length(){ // Độ phức tạp là O(1)  
 **return** lengthQueue;  
 }  
 **void** enQueue(T a){ // Độ phức tạp là O(1)  
 **if**(empty()){  
 head = tail = **new** Node<T>(a, NULL); // Độ phức tạp là O(1)  
 }  
 **else**{  
 tail->setLink(**new** Node<T>(a, NULL)); // Độ phức tạp là O(1)  
 tail = tail->getLink(); // O(1)  
 }  
 lengthQueue++;  
 }  
 T deQueue(){ // Độ phức tạp là O(1)  
 **if**(empty()){  
 cout << **"The Queue is empty"**; // O(1)  
 exit(1);  
 }  
 **else**{  
 T temp = head->getData(); // trong block này độ phức tạp là O(1)  
 Node<T>\* p = head;  
 head = head->getLink();  
 **if**(head == NULL) tail = NULL;  
 delete p;  
 lengthQueue--;  
 **return** temp;  
 }  
 }  
 T getHead(){ // O(1)  
 **if**(empty()){  
 cout << **"The Queue is empty"**; // O(1)  
 exit(1);  
 }**else**{  
 **return** head->getData(); // O(1)  
 }  
 }

Ta dễ thấy empty, length, enQueue, deQueue và getHead đều có độ phức tạp là O(1)

khi cài đặt bằng danh sách liên kết như trên

Bài 3:

#include <iostream>  
#include <cstdlib>  
  
using namespace std;  
  
template <**class** T>  
**class** Stack{  
**private**:  
 T\* data;  
 **int** top, size;  
**public**:  
 Stack(**int** size1 = 100){  
 size = size1;  
 top = -1;  
 data = **new** T[size];  
 }  
 virtual ~Stack(){  
 delete []data;  
 }  
 bool empty(){  
 **return** top == -1;  
 }  
 **int** length(){  
 **return** top + 1;  
 }  
 **void** push(**const** T &x){  
 **if**(top == size - 1){  
 T \*temp = **new** T[2\*size];  
 **for**(**int** i = 0; i < size; i++){  
 temp[i] = data[i];  
 }  
 size = 2\*size;  
 delete[] data;  
 data = temp;  
 }  
 data[++top] = x;  
 }  
 T pop(){  
 **if**(top >= 0){  
 **return** data[top--];  
 } **else**{  
 cout << **"The stack is empty"**;  
 exit(1);  
 }  
 }  
 T getTop(){  
 **if**(top >= 0){  
 **return** data[top];  
 } **else**{  
 cout << **"The stack is empty"**;  
 exit(1);  
 }  
 }  
};  
  
*// TEST***class** Integer{  
**private**:  
 **int** value;  
**public**:  
 Integer(**int** value1 = 0) : value(value1){}  
 friend ostream& operator << (ostream& os, Integer a){  
 os << a.value;  
 **return** os;  
 }  
};  
  
**int** main(){  
 Stack<Integer> st(50);  
 **if**(st.empty())  
 cout << **"The stack is empty"** << endl;  
 cout << **"length = "** << st.length() << endl;  
 **for**(**int** i = 0; i < 20; i++){  
 st.push(\*(**new** Integer(i)));  
 }  
  
  
 cout << **"After push some element, length = :"** << st.length() << endl;  
 **for**(**int** i = 0; i < 5; i++){  
 st.pop();  
 }  
 cout << **"After pop first 5 elements, the length is: "** << st.length() << endl;  
 cout << **"Stack contains: "**;  
 **while**(!st.empty()){  
 cout << st.pop() << **" "**;  
 }  
  
 cout << endl << **"After pop all elements, length = :"** << st.length() << endl;

}

Tính độ phức tạp:

bool empty(){  
 **return** top == -1; // O(1)  
 }

Dễ thấy empty có độ phức tạp là O(1)

**int** length(){  
 **return** top + 1; //O(1)  
 }

Dễ thấy length có độ phức tạp là O(1)

**void** push(**const** T &x){  
 **if**(top == size - 1){  
 T \*temp = **new** T[2\*size];   
 **for**(**int** i = 0; i < size; i++){ // (1)  
 temp[i] = data[i]; // (2)  
 }  
 size = 2\*size;  
 delete[] data;  
 data = temp;  
 }  
 data[++top] = x;  
 }

Ta thấy trong vòng for tại vị trí (2) có độ phức tạp là O(1)

* Từ (1) -> (2) độ phức tạp là O(size) do vòng for được lặp lại size lần.

Các dòng còn lại đều có độ phức tạp là O(1).

Vậy độ phức tạp trong trường hợp này là O(size) = O(n).

T pop(){  
 **if**(top >= 0){  
 **return** data[top--]; // O(1)  
 } **else**{  
 cout << **"The stack is empty"**; // O(1)  
 exit(1);  
 }  
 }

Dễ thấy pop có độ phức tạp là O(1)

T getTop(){  
 **if**(top >= 0){  
 **return** data[top]; // O(1)  
 } **else**{  
 cout << **"The stack is empty"**; // O(1)  
 exit(1);  
 }  
 }

Dễ thấy getTop có độ phức tạp là O(1)

Bài 4: Cài đặt Stack bằng danh sách liên kết:

#include <iostream>  
#include <cstdlib>  
  
  
using namespace std;  
  
template <**class** T>  
**class** Node{  
**private**:  
 T data;  
 Node\* link;  
**public**:  
 Node(T data1, Node\* link1 = NULL) : data(data1), link(link1){}  
 **void** setData(T data1){  
 data = data1;  
 }  
 **void** setLink(T\* link1){  
 link = link1;  
 }  
 T getData(){  
 **return** data;  
 }  
 Node\* getLink(){  
 **return** link;  
 }  
};  
  
template <**class** T>  
**class** Stack{  
**private**:  
 Node<T>\* top;  
 **int** lengthStack;  
**public**:  
 Stack() : top(NULL), lengthStack(0){}  
 virtual ~Stack(){  
 **while**(!empty()){  
 pop();  
 }  
 }  
 bool empty(){  
 **return** top == NULL;  
 }  
 **int** length(){  
 **return** lengthStack;  
 }  
 **void** push(**const** T &x){  
 top = **new** Node<T>(x, top);  
 lengthStack++;  
 }  
 T pop(){  
 **if**(!empty()){  
 T temp = top->getData();  
 Node<T>\* remove = top;  
 top = top->getLink();  
 delete remove;  
 lengthStack--;  
 **return** temp;  
 } **else**{  
 cout << **"The stack is empty"**;  
 exit(1);  
 }  
 }  
 T getTop(){  
 **if**(empty()){  
 cout << **"The stack is empty"**;  
 exit(1);  
 }  
 **return** top->getData();  
 }  
};  
  
  
*// TEST***class** Integer{  
**private**:  
 **int** value;  
**public**:  
 Integer(**int** value1 = 0) : value(value1){}  
 friend ostream& operator << (ostream& os, Integer a){  
 os << a.value;  
 **return** os;  
 }  
};  
**int** main(){  
 Stack<Integer> st;  
 **if**(st.empty()) cout << **"The stack is empty"** << endl;  
 cout << **"length = "** << st.length() << endl;  
 **for**(**int** i = 0; i < 20; i++){  
 st.push(\*(**new** Integer(i)));  
 }  
  
  
 cout << **"After push some element, length = :"** << st.length() << endl;  
 **for**(**int** i = 0; i < 5; i++){  
 st.pop();  
 }  
 cout << **"After pop first 5 elements,length = : "** << st.length() << endl;  
 cout << **"Stack contains: "**;  
 **while**(!st.empty()){  
 cout << st.pop() << **" "**;  
 }

}

Tính độ phức tạp:

bool empty(){ // Độ phức tạp là O(1)  
 **return** top == NULL; // O(1)  
 }  
 **int** length(){ // Độ phức tạp là O(1)  
 **return** lengthStack; // O(1)  
 }  
 **void** push(**const** T &x){ // Độ phức tạp là O(1)  
 top = **new** Node<T>(x, top); // O(1)  
 lengthStack++; // O(1)  
 }  
 T pop(){ // Độ phức tạp là O(1)  
 **if**(!empty()){  
 T temp = top->getData(); // Phần trong block có độ phức tạp là O(1)  
 Node<T>\* remove = top;  
 top = top->getLink();  
 delete remove;  
 lengthStack--;  
 **return** temp;  
 } **else**{  
 cout << **"The stack is empty"**; // Phần trong block có độ phức tạp là O(1)  
 exit(1);  
 }  
 }  
 T getTop(){ // Độ phức tạp là O(1)  
 **if**(empty()){  
 cout << **"The stack is empty"**; // Phần trong block có độ phức tạp là O(1)  
 exit(1);  
 }  
 **return** top->getData(); // O(1)  
 }

Dễ thấy empty, length, push, pop, getTop đều có độ phức tạp là O(1)