#include<iostream>

using namespace std;

/\*

DLL == ER queue

    new critical patients arrive

    discharged patients leave

-----------------------------

insertAtBeginning(patientID)

insertAtEnd(patientID)

insertAtPosition(patientID, position)

deleteFromBeginning()

-----------------------------

Node == patient

    patientID (integer)

    pointers to previous patient

    pointers to next patient

-----------------------------

-----------------------------

\*/

class Patient {

    public:

        int patientID;

        Patient\* next;

        Patient\* prev;

        Patient(int patientID)

        {

            this->patientID = patientID;

            next = prev = nullptr;

        }

};

class ER\_Queue{

    public:

        Patient \*head;

        Patient \*tail;

        ER\_Queue(){

            cout << "ER Queue is now Open for Emergency's." << endl;

            head = nullptr;

            tail = nullptr;

        }

        ~ER\_Queue(){

            Patient\* temp = head;

            if(head != nullptr){

                cout << "Remaining Patients have been Sent Home." << endl;

            }

            while(head != nullptr){// run loop half forward from head to middle simultaneously from tail to middle delete at 2x speed

                head = head->next;

                cout << "Patient ID: " << temp->patientID << " Sent Home." << endl;

                delete temp;

                temp = head;

            }//understand this sir even though tail+prev's are dangling but after this distructor they wont exist thus it doesnt matter if i equal them to nullptr or not cos either way no one can access them anymore

            cout << "ER Queue Ended." << endl;

        }

        void insertAtBeginning(int patientID){

            cout << "new critical patient arrived, Patient ID: " << patientID << endl;

            Patient\* newPatient = new Patient(patientID);

            if(head != nullptr){

                newPatient->next = head;

                head->prev = newPatient;

            }

            else{

                tail = newPatient;

            }

            head = newPatient;

        }

        void insertAtEnd(int patientID){

            cout << "new normal walk-in patient arrived, Patient ID: " << patientID << endl;

            Patient\* newPatient = new Patient(patientID);

            if(tail == nullptr){//if queue empty

                head = newPatient;

                tail = newPatient;

                return;

            }

            newPatient->prev = tail;

            tail->next = newPatient;

            tail = newPatient;

        }

        void deleteFromBeginning(){

            if(head == nullptr){

                cout << "Underflow, Queue Empty." << endl;

                return;

            }

            cout << "discharged patient left, Patient ID: " << head->patientID << endl;

            Patient\* temp = head;

            if(head->next != nullptr){

                head->next->prev = nullptr;

            }

            head = head->next;

            delete temp;

            if(head == nullptr){

                tail = nullptr;

            }

        }

        void insertAtPosition(int patientID, int position){

            if(position<1){

                cout << "Invalid Position, out of bounds -ve." << endl;

            }

            if(position == 1){

                insertAtBeginning(patientID);

                return;

            }

            Patient\* newPatient = new Patient(patientID);

            Patient\* curr = head,\*temp;

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

                if(curr->next == nullptr){

                    cout << "Invalid Position, out of bounds +ve." << endl;

                    cout << "Patient Inserted At End" << endl;

                    insertAtEnd(patientID);

                    return;

                }

                curr = curr->next;

            }

            if(curr == tail){

                insertAtEnd(patientID);

                return;

            }

            temp = curr->next;

            curr->next = temp->next;

            if(temp->next != nullptr){

                temp->next->prev = curr;

            }

            delete temp;

            cout << "new patient arrived, Patient ID: " << patientID << " At Position " << position << endl;

        }

        void Display(){

            Patient\* curr = head;

            if(head==nullptr){

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

                return;

            }

            while(curr != nullptr){

                cout << curr->patientID;

                if(curr->next != nullptr){

                    cout << " -> ";

                }

                curr = curr->next;

            }

            cout << endl;

        }

};

int main(){

    //---------------------------------------------------------------

    cout << "Welcome To Hospital Emergency Queue System" << endl;

    ER\_Queue\* Queue = new ER\_Queue;

    Queue->insertAtEnd(101);

    Queue->insertAtEnd(102);

    Queue->insertAtBeginning(200); // critical patient

    Queue->insertAtPosition(150, 2);

    Queue->deleteFromBeginning();

    Queue->insertAtEnd(300);

    //.-------------------------------------------------------------

    return 0;

}

#include<iostream>

using namespace std;

/\*

DLL == ER queue

new critical patients arrive

discharged patients leave

-----------------------------

insertAtBeginning(patientID)

insertAtEnd(patientID)

insertAtPosition(patientID, position)

deleteFromBeginning()

-----------------------------

Node == patient

patientID (integer)

pointers to previous patient

pointers to next patient

-----------------------------

-----------------------------

\*/

class Patient {

public:

int patientID;

Patient\* next;

Patient\* prev;

Patient(int patientID)

{

this->patientID = patientID;

next = prev = nullptr;

}

};

class ER\_Queue{

public:

Patient \*head;

Patient \*tail;

ER\_Queue(){

cout << "ER Queue is now Open for Emergency's." << endl;

head = nullptr;

tail = nullptr;

}

~ER\_Queue(){

Patient\* temp = head;

if(head != nullptr){

cout << "Remaining Patients have been Sent Home." << endl;

}

while(head != nullptr){// run loop half forward from head to middle simultaneously from tail to middle delete at 2x speed

head = head->next;

cout << "Patient ID: " << temp->patientID << " Sent Home." << endl;

delete temp;

temp = head;

}//understand this sir even though tail+prev's are dangling but after this distructor they wont exist thus it doesnt matter if i equal them to nullptr or not cos either way no one can access them anymore

cout << "ER Queue Ended." << endl;

}

void insertAtBeginning(int patientID){

cout << "new critical patient arrived, Patient ID: " << patientID << endl;

Patient\* newPatient = new Patient(patientID);

if(head != nullptr){

newPatient->next = head;

head->prev = newPatient;

}

else{

tail = newPatient;

}

head = newPatient;

}

void insertAtEnd(int patientID){

cout << "new normal walk-in patient arrived, Patient ID: " << patientID << endl;

Patient\* newPatient = new Patient(patientID);

if(tail == nullptr){//if queue empty

head = newPatient;

tail = newPatient;

return;

}

newPatient->prev = tail;

tail->next = newPatient;

tail = newPatient;

}

void deleteFromBeginning(){

if(head == nullptr){

cout << "Underflow, Queue Empty." << endl;

return;

}

cout << "discharged patient left, Patient ID: " << head->patientID << endl;

Patient\* temp = head;

if(head->next != nullptr){

head->next->prev = nullptr;

}

head = head->next;

delete temp;

if(head == nullptr){

tail = nullptr;

}

}

void insertAtPosition(int patientID, int position){

if(position<1){

cout << "Invalid Position, out of bounds -ve." << endl;

}

if(position == 1){

insertAtBeginning(patientID);

return;

}

Patient\* newPatient = new Patient(patientID);

Patient\* curr = head,\*temp;

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

if(curr->next == nullptr){

cout << "Invalid Position, out of bounds +ve." << endl;

cout << "Patient Inserted At End" << endl;

insertAtEnd(patientID);

return;

}

curr = curr->next;

}

if(curr == tail){

insertAtEnd(patientID);

return;

}

temp = curr->next;

curr->next = temp->next;

if(temp->next != nullptr){

temp->next->prev = curr;

}

delete temp;

cout << "new patient arrived, Patient ID: " << patientID << " At Position " << position << endl;

}

void Display(){

Patient\* curr = head;

if(head==nullptr){

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

return;

}

while(curr != nullptr){

cout << curr->patientID;

if(curr->next != nullptr){

cout << " -> ";

}

curr = curr->next;

}

cout << endl;

}

};

int main(){

//---------------------------------------------------------------

cout << "Welcome To Hospital Emergency Queue System" << endl;

ER\_Queue\* Queue = new ER\_Queue;

Queue->insertAtEnd(101);

Queue->insertAtEnd(102);

Queue->insertAtBeginning(200); // critical patient

Queue->insertAtPosition(150, 2);

Queue->deleteFromBeginning();

Queue->insertAtEnd(300);

//.-------------------------------------------------------------

return 0;

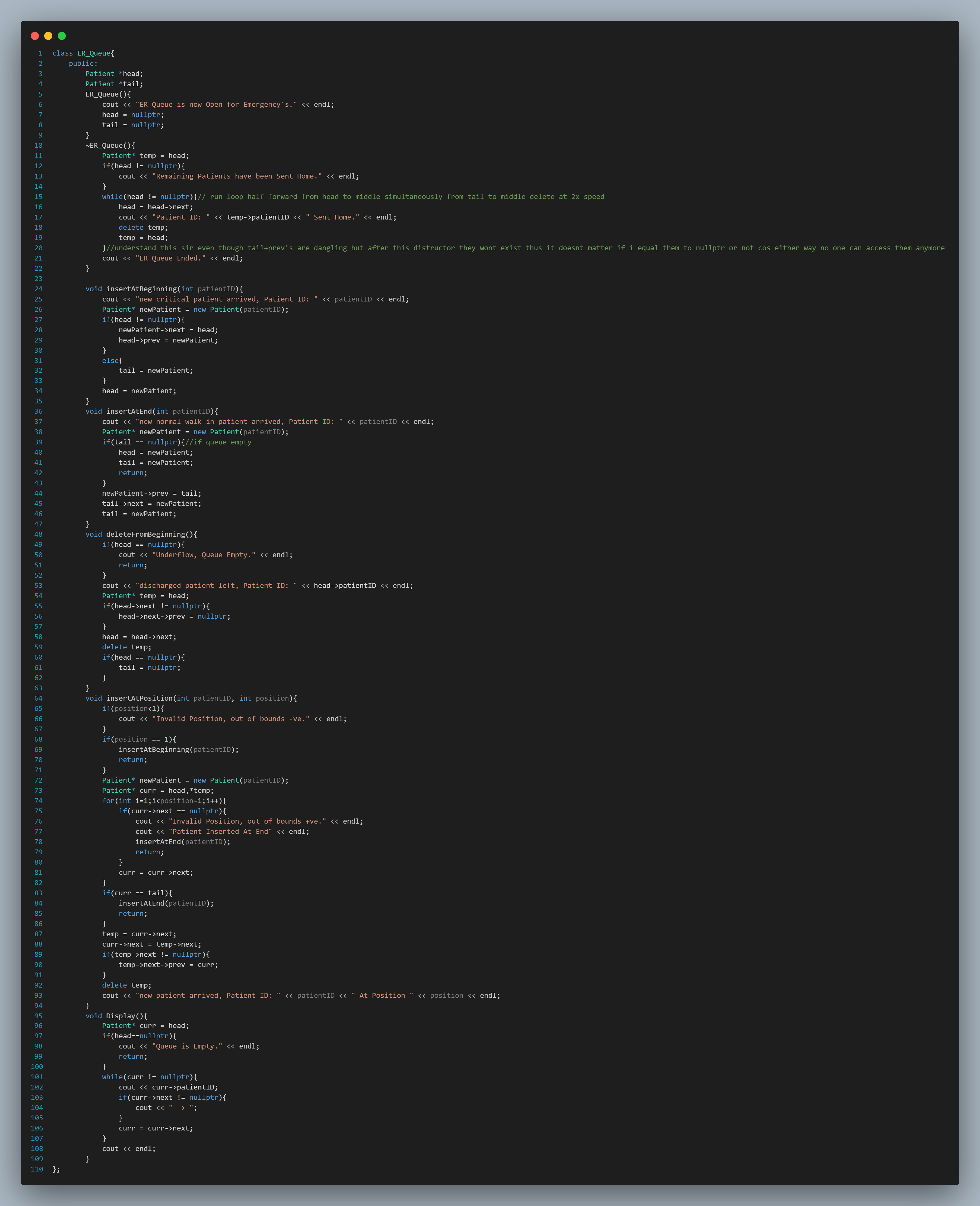
}

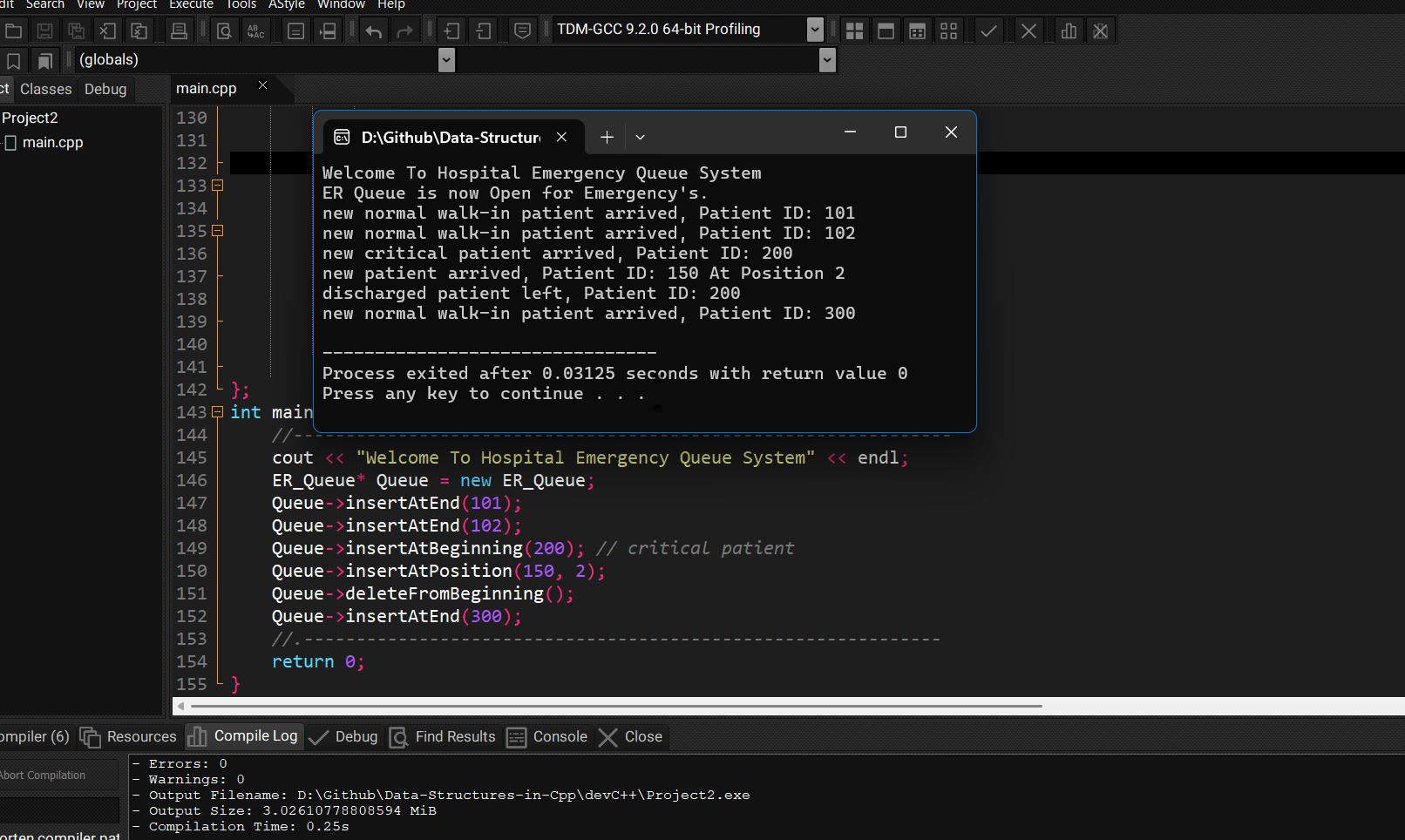
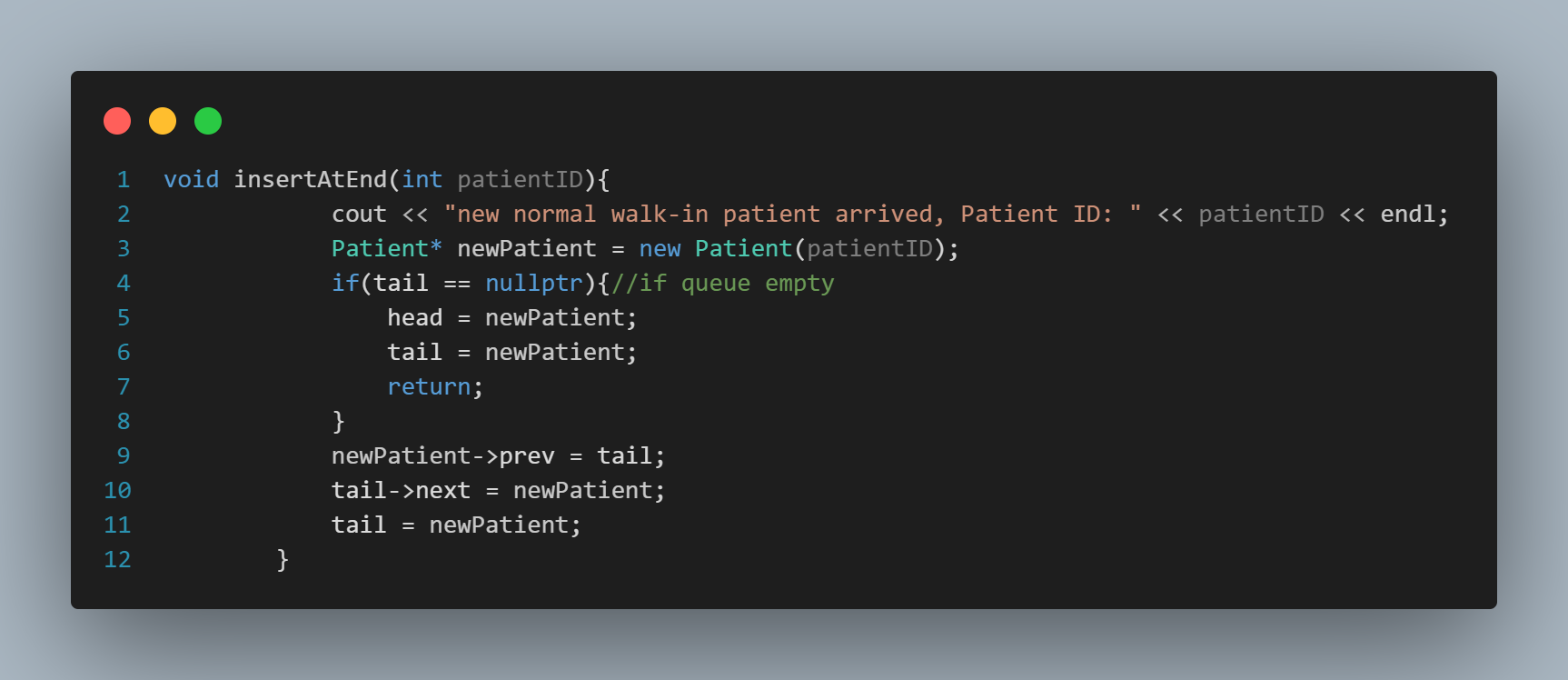
A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.