Instituto Tecnológico y de Estudios Superiores de Monterrey

Programación de estructuras de datos y algoritmos fundamentales

Conceptos Básicos y Algoritmos Fundamentales.

Estructura: “Fila priorizada”

Presenta:
Miguel Ángel Pérez Ávila A01369908

Profesor:
Dr. Mauricio Paletta Nannarone

Toluca, Estado de México a 23 de noviembre del 2023

.HPP

```
#include <vector>

using namespace std;

class Input{        //Clase para los elementos de la fila priorizada
        private:
                int data;
                int priority;

        public:
                Input(int n, int p){
                        data= n;
                        priority= p;
                }

        int getdata(){
                return data;
        }
        int getprio(){
                return priority;
        }
        void changeD(int newD){
                data= newD;
        }
        void changePrior(int newP){
                priority= newP;
        }
};

class PriorityQueue{

        private:
                vector<Input> fila;

        public:

                //Constructor de clase
                PriorityQueue(){
                }

        // Métodos:
        bool push( Input data );

        bool pop();

        int top();

        bool empty();
```

```
        int size();

        void show();

        vector<Input> getrow();
};
```

priorityQueue.cpp

```
//  Definición de métodos  -> outline

#include "ADT.hpp"

// Public ============================================================================

bool PriorityQueue::push( Input data ){
        if(data.getprio()<0 || data.getprio()>9){   //Solo se admite un Input con prioridad entre 0-9
                return false;
        }else{
                if(!empty()){
                        for(int i=0; i<fila.size(); i++){ //Recorrido de la fila para ubicar la nueva data
                                // Verificar la prioridad de los datos existentes en la fila
                                if(fila.at(i).getprio() > data.getprio()){
                                        auto it = fila.begin()+i;
                                        fila.insert(it, data);
                                        return true;
                                }
                                if(fila.at(i).getprio() == data.getprio()){
                                        for(int j=i; j<fila.size(); j++){
                                                if(fila.at(j).getprio() != data.getprio() || j == fila.size()){
                                                        auto it = fila.begin()+j;
                                                        fila.insert(it, data);
                                                        return true;
                                                }else{
                                                        auto it = fila.begin()+j;
                                                        fila.insert(it, data);
                                                        return true;
                                                }
                                        }
                                }
                                if(fila.at(i).getprio() < data.getprio() && i==fila.size()-1){
                                        auto it = fila.begin()+i+1;
                                        fila.insert(it, data);
                                        return true;
                                }
                                if(i== fila.size()){
                                        fila.push_back(data);
                                        return true;
                                }
                        }
                }else{
```

```
                        fila.push_back(data);
                        return true;
                }
        }
}

bool PriorityQueue::pop(){  /*Saca el último elemento de la fila que fue el primer dato en entrar de la mayor prioridad existente en la fila*/
        if(!empty()){
                fila.pop_back();
                return true;
        }else{
                return false;
        }
}

int PriorityQueue::top(){  /* Retorna el el último elemento de la fila que fue el primer dato en entrar de la mayor prioridad existente en la fila*/
        if(!empty()){
                return fila.back().getdata();
        }else{
                return 0;
        }
}

bool PriorityQueue::empty(){  // Retorna el estado de la fila donde True=Vacía y False= No vacía
        return fila.empty();
}

int PriorityQueue::size(){ // Retorna el tamaño de la fila (número de elementos)
        return(fila.size());
}



vector<Input> PriorityQueue::getrow(){
        return fila;
}
```

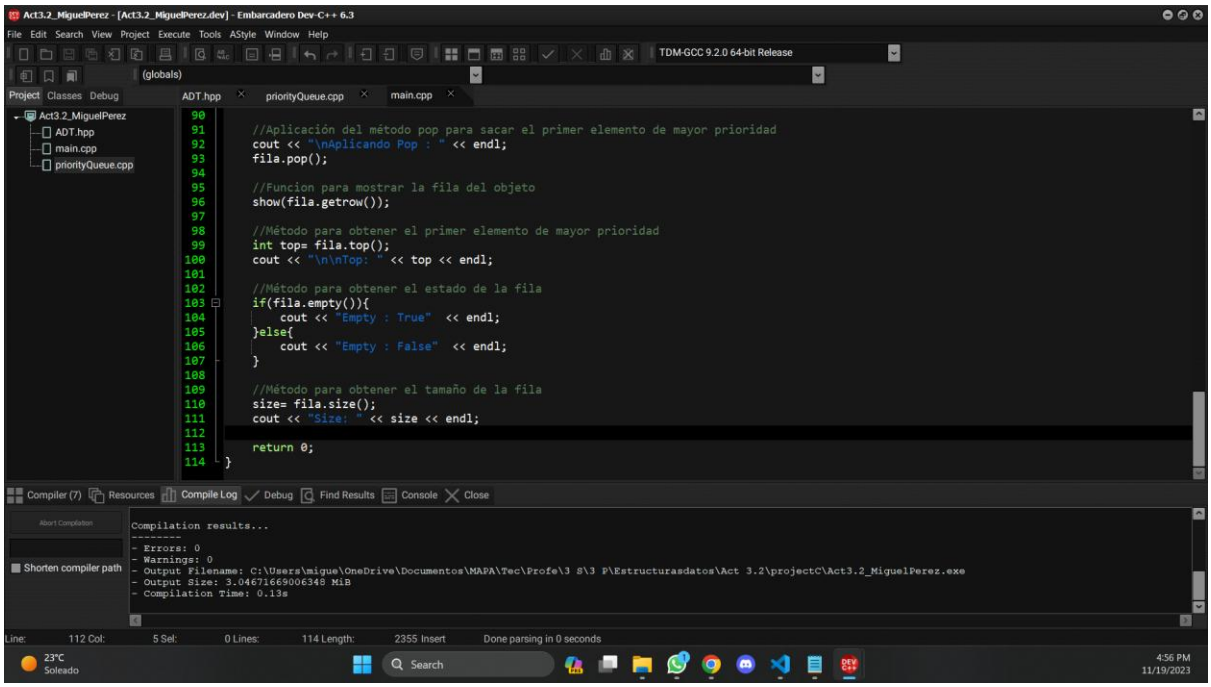
Main.cpp

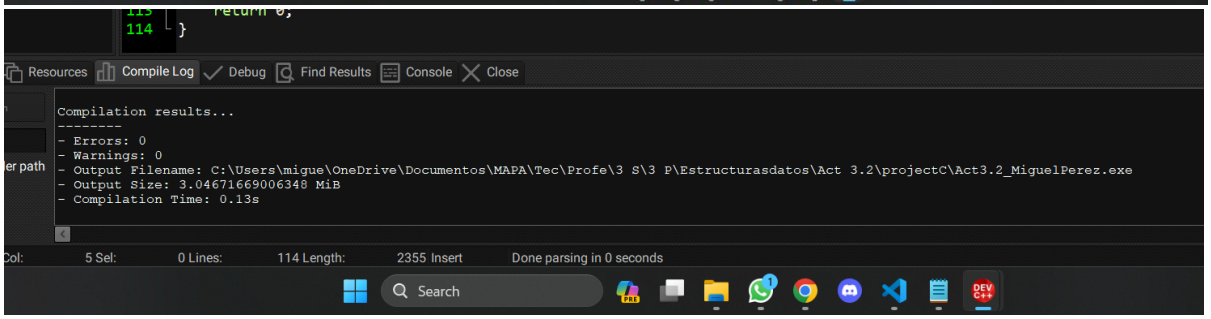
```
#include "ADT.hpp"

// Función para mostrar la fila del objeto
void show(vector<Input> fila){
        cout << "Data - Priority" << endl;
        if(!fila.empty()){
                for(int i=0; i< fila.size(); i++){
                        Input aux= fila.at(i);
                        cout << aux.getdata() << "   - ";
                        cout << aux.getprio()<< endl;
```

```
                }
        }
}

//MAIN
int main(){

        PriorityQueue fila; // Instancia de clase

        //Muestra del estado inicial de la fila
        cout << "INICIAL STATE ----------" << endl;
        cout << "SIZE : " << fila.size() << "\n\n\n" << endl;

        //Instancia de clase Input con valores iniciales
        Input aux(23,0);
        //Aplicación del método Push para todos los Inputs a continuación:
        cout << "\nAplicando Push : "  << endl;
        fila.push( aux );

        aux.changeD(24);
        aux.changePrior(1);
        fila.push( aux );

        aux.changeD(25);
        aux.changePrior(1);
        fila.push( aux );

        aux.changeD(26);
        aux.changePrior(1);
        fila.push( aux );

        aux.changeD(27);
        aux.changePrior(0);
        fila.push( aux );

        aux.changeD(28);
        aux.changePrior(0);
        fila.push( aux );

        aux.changeD(29);
        aux.changePrior(3);
        fila.push( aux );

        aux.changeD(30);
        aux.changePrior(8);
        fila.push( aux );

        aux.changeD(31);
        aux.changePrior(2);
        fila.push( aux );

        aux.changeD(32);
```

```
        aux.changePrior(1);
        fila.push( aux );

        aux.changeD(33);
        aux.changePrior(8);
        fila.push( aux );

        aux.changeD(34);
        aux.changePrior(2);
        fila.push( aux );

        aux.changeD(35);
        aux.changePrior(9);
        fila.push( aux );

        aux.changeD(36);
        aux.changePrior(3);
        fila.push( aux );

        //Funcion para mostrar la fila del objeto
        show(fila.getrow());

        //Método para obtener el tamaño de la fila
        int size= fila.size();
        cout << "\nSize: " << size << endl;

        //Aplicación del método pop para sacar el primer elemento de mayor prioridad
        cout << "\nAplicando Pop : " << endl;
        fila.pop();

        //Funcion para mostrar la fila del objeto
        show(fila.getrow());

        //Método para obtener el primer elemento de mayor prioridad
        int top= fila.top();
        cout << "\n\nTop: " << top << endl;

        //Método para obtener el estado de la fila
        if(fila.empty()){
                cout << "Empty : True"  << endl;
        }else{
                cout << "Empty : False"  << endl;
        }

        //Método para obtener el tamaño de la fila
        size= fila.size();
        cout << "Size: " << size << endl;

        return 0;
}
```

EVIDENCIA DE COMPILACIÓN





EVIDENCIA DE CASO DE PRUEBA / EJECUCIÓN DE MAIN.CPP