

ESTRUCTURAS DE DATOS

listagenerica.hpp

```
lista_generica.hpp
1 //
2 // lista_generica.hpp
3 // lista_generica
4 //
5 // Created by Wilfredo Aleman on 27/4/21.
6 //
7
8 #ifndef lista_generica_hpp
9 #define lista_generica_hpp
10
11 #include <iostream>
12 #include <string>
13 #include <sstream>
14
15 using std::string;
16 using std::ostringstream;
17 using std::cout;
18 using std::endl;
19
20 template <typename TIPODATO>
21 class Lista {
22     private:
23         int cuenta;
24         int capacidad;
25         TIPODATO *items;
26         void agrandar();
27     public:
28         Lista(int capacidad);
29         Lista();
30         ~Lista();
31         Lista(const Lista &otra); // Construir una lista a partir de otra en una dirección de memoria
32         bool estaVacia();
33         int tamano();
34         void insertar(int indice, TIPODATO item);
35         void adjuntar(TIPODATO item);
36         TIPODATO obtener(int indice);
37         bool contiene(TIPODATO item);
38         TIPODATO remover(int indice);
39         string comoCadena();
40 }
```



lista_generica.hpp

```
37     bool contiene(TIPODATO item);
38     TIPODATO remover(int indice);
39     string comoCadena();
40 };
41
42 // Constructor
43 template <typename TIPODATO>
44 Lista<TIPODATO>::Lista(int capacidad) {
45     cout << "Creando una lista vacia..." << endl;
46     this->cuenta = 0;
47     this->capacidad = capacidad;
48     this->items = new TIPODATO[capacidad];
49     cout << " La lista tiene capacidad para " << this->capacidad << " elementos" << endl;
50 }
51
52 // Constructor con tamaño por defecto, en este caso 4
53 template <typename TIPODATO>
54 Lista<TIPODATO>::Lista() : Lista(4) {}
55
56 // Destructor de la lista
57 template <typename TIPODATO>
58 Lista<TIPODATO>::~~Lista() {
59     cout << "Destruyendo la lista..." << endl;
60     delete[] items;
61 }
62
63 // Construir una lista a partir de otra
64 template <typename TIPODATO>
65 Lista<TIPODATO>::Lista(const Lista<TIPODATO> &otra) {
66     cout << "Copiando una lista desde otra..." << endl;
67     this->cuenta = otra.cuenta;
68     this->capacidad = otra.capacidad;
69     this->items = new TIPODATO[capacidad];
70     for (int i = 0; i < this->cuenta; i++)
71         this->items[i] = otra.items[i];
72 }
73
74 // Esta la lista vacia?
75 template <typename TIPODATO>
```



```
lista_generica.hpp
73
74 // Esta la lista vacia?
75 template <typename TIPODATO>
76 bool Lista<TIPODATO>::estaVacia() {
77     cout << "Comprobando si la lista esta vacia..." << endl;
78     return this->cuanta == 0;
79 }
80
81 // Tamano de la lista
82 template <typename TIPODATO>
83 int Lista<TIPODATO>::tamano() {
84     cout << "Obteniendo el tamano de la lista..." << endl;
85     return this->cuanta;
86 }
87
88 template <typename TIPODATO>
89 void Lista<TIPODATO>::agrandar() {
90     cout << "Duplicando el tamano de la lista..." << endl;
91     TIPODATO *temp = this->items;
92     this->capacidad *= 2;
93     this->items = new TIPODATO[capacidad];
94     for (int i = 0; i < cuanta; i++)
95         this->items[i] = temp[i];
96     delete[] temp;
97     cout << " La lista tiene capacidad para " << this->capacidad << " elementos" << endl;
98 }
99
100 // Insertar un elemento en un indice especifico
101 template <typename TIPODATO>
102 void Lista<TIPODATO>::insertar(int indice, TIPODATO item) {
103     // Validar el indice
104     if (indice < 0 || indice > this->cuanta) throw "Indice fuera de rango";
105
106     // Agrandar la lista si el elemento no cabe
107     if (this->cuanta >= this->capacidad) this->agrandar();
108
109     cout << "Insertando un elemento..." << endl;
110     // Desplazar elementos a partir del indice a la derecha
111     for (int i = cuanta - 1; i >= indice; i--) {
```



```
lista_generica.hpp
109 cout << "Insertando un elemento..." << endl;
110 // Desplazar elementos a partir del indice a la derecha
111 for (int i = cuenta - 1; i >= indice; i--) {
112     cout << " Desplazando elemento " << items[i] << " del indice ";
113     cout << i << " al indice " << (i+1) << endl;
114     this->items[i+1] = this->items[i];
115 }
116
117 // Insertar nuevo elemento
118 cout << " Insertando elemento " << item << " en el indice " << indice << endl;
119 this->items[indice] = item;
120
121 // Incrementar la cuenta de elementos
122 cout << " Incrementando la cuenta de elementos" << endl;
123 this->cuenta++;
124 }
125
126 // Agregar un elemento al final
127 template <typename TIPODATO>
128 void Lista<TIPODATO>::adjuntar(TIPODATO item) {
129     this->insertar(this->cuenta, item);
130 }
131
132 // Valor del elemento en la posicion indice
133 template <typename TIPODATO>
134 TIPODATO Lista<TIPODATO>::obtener(int indice) {
135     // Validar el indice
136     if (indice < 0 || indice >= this->cuenta) throw "Indice fuera de rango";
137     if (this->estaVacia()) throw "No se pued recuperar elementos de una lista vacia";
138
139     cout << "Recuperando elemento en el indice " << indice << endl;
140     return this->items[indice];
141 }
142
143 // Esta el item en la lista?
144 template <typename TIPODATO>
145 bool Lista<TIPODATO>::contiene(TIPODATO item) {
146     for (int i = 0; i < this->cuenta; i++) {
147         cout << "Recorriendo elemento con indice " << i << endl;
```



```
lista_generica.hpp
145 bool Lista<TIPODATO>::contiene(TIPODATO item) {
146     for (int i = 0; i < this->cuenta; i++) {
147         cout << "Recorriendo elemento con indice " << i << endl;
148         if (this->items[i] == item)
149             return true;
150     }
151
152     return false;
153 }
154
155 // Remover elemento en el indice indice'
156 template <typename TIPODATO>
157 TIPODATO Lista<TIPODATO>::remove(int indice) {
158     // Validar el indice
159     if (this->estaVacia()) throw "No se puede remover elementos de una lista vacia";
160     if (indice < 0 || indice >= this->cuenta) throw "Indice fuera de rango";
161
162     cout << "Removiendo el elemento con indice " << indice << "..." << endl;
163     // Lee el valor en el indice y desplaza elementos a la izquierda
164     TIPODATO valor = this->items[indice];
165     for (int i = indice; i < cuenta - 1; i++) {
166         cout << " Desplazando elemento " << items[i] << " del indice ";
167         cout << (i+1) << " al indice " << i << endl;
168         this->items[i] = this->items[i+1];
169     }
170
171     // Reducir la cuenta de elementos
172     cout << " Reduciendo la cuenta de elementos" << endl;
173     this->cuenta--;
174     return valor;
175 }
176
177 template <typename TIPODATO>
178 string Lista<TIPODATO>::comoCadena() {
179     ostringstream s;
180     s << "[";
181     for (int i = 0; i < (this->cuenta); i++) {
182         s << this->items[i];
183         if (i < this->cuenta-1)
```



```
176
177 template <typename TIPODATO>
178 string Lista<TIPODATO>::comoCadena() {
179     ostringstream s;
180     s << "[";
181     for (int i = 0; i < (this->cuanta); i++) {
182         s << this->items[i];
183         if (i < this->cuanta-1)
184             s << ", ";
185     }
186     s << "]";
187     return s.str();
188 }
189
190 #endif /* lista_generica_hpp */
```

listagenerica.cpp



```
lista_generica.hpp lista_generica.cpp
1 //
2 // lista_generica.cpp
3 // lista_generica
4 //
5 // Created by Wilfredo Aleman on 27/4/21.
6 //
7 #include "lista_generica.hpp"
8 #include <iostream>
9
10 using std::cin;
11 using std::cout;
12 using std::endl;
13
14 int main() {
15     /**/
16     Lista<int> milista;
17     cout << "La lista es: " << milista.comoCadena() << endl << endl;
18
19     Lista<int> milista2 = milista;
20     cout << endl;
21
22     cout << (milista.estaVacia() ? "true" : "false") << endl << endl;
23
24     milista.adjuntar(41);
25     cout << "La lista es: " << milista.comoCadena() << endl << endl;
26
27     milista.adjuntar(52);
28     cout << "La lista es: " << milista.comoCadena() << endl << endl;
29
30     milista.adjuntar(63);
31     cout << "La lista es: " << milista.comoCadena() << endl << endl;
32
33     milista.insertar(2, 74);
34     cout << "La lista es: " << milista.comoCadena() << endl << endl;
35
36     milista.insertar(0, 30);
37     cout << "La lista es: " << milista.comoCadena() << endl << endl;
38
39     cout << "Tamano de la lista: " << milista.tamano() << endl << endl;
```



```
lista_generica.hpp lista_generica.cpp
37     cout << "La lista es: " << milista.comoCadena() << endl << endl;
38
39     cout << "Tamano de la lista: " << milista.tamano() << endl << endl;
40
41     cout << milista.obtener(4) << endl << endl;
42
43     cout << milista.remove(2) << endl << endl;
44     cout << "La lista es: " << milista.comoCadena() << endl << endl;
45
46     milista.adjuntar(85);
47     cout << "La lista es: " << milista.comoCadena() << endl << endl;
48     /**/
49
50     Lista<double> milistaDouble;
51     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
52
53     Lista<double> milistaDouble2 = milistaDouble;
54     cout << endl;
55
56     cout << (milistaDouble.estaVacía() ? "true" : "false") << endl << endl;
57
58     milistaDouble.adjuntar(100.1);
59     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
60
61     milistaDouble.adjuntar(105.3);
62     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
63
64     milistaDouble.adjuntar(94.5);
65     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
66
67     milistaDouble.insertar(1, 107.7);
68     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
69
70     milistaDouble.insertar(2, 365.50);
71     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
72
73     cout << "Tamano de la lista: " << milistaDouble.tamano() << endl << endl;
74
75     cout << milistaDouble.obtener(4) << endl << endl;
```




```
lista_generica.hpp lista_generica.cpp
46     milista.adjuntar(85);
47     cout << "La lista es: " << milista.comoCadena() << endl << endl;
48     /**/
49
50     Lista<double> milistaDouble;
51     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
52
53     Lista<double> milistaDouble2 = milistaDouble;
54     cout << endl;
55
56     cout << (milistaDouble.estaVacia() ? "true" : "false") << endl << endl;
57
58     milistaDouble.adjuntar(100.1);
59     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
60
61     milistaDouble.adjuntar(105.3);
62     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
63
64     milistaDouble.adjuntar(94.5);
65     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
66
67     milistaDouble.insertar(1, 107.7);
68     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
69
70     milistaDouble.insertar(2, 365.50);
71     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
72
73     cout << "Tamano de la lista: " << milistaDouble.tamano() << endl << endl;
74
75     cout << milistaDouble.obtener(4) << endl << endl;
76
77     cout << milistaDouble.remove(2) << endl << endl;
78     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
79
80     milistaDouble.adjuntar(104.5);
81     cout << "La lista es: " << milistaDouble.comoCadena() << endl << endl;
82
83     return 0;
84 }
```



pila.hpp

```
lista_generica.hpp lista_generica.cpp pila.hpp
1  #include "lista_generica.hpp"
2  #include <string>
3
4  template <typename TIPODATO>
5  class Pila : private Lista<TIPODATO>{
6      public:
7          // Constructor
8          Pila();
9          // Agregar un elemento en la parte superior
10         push(TIPODATO item);
11         // Remover un elemento de la parte superior
12         TIPODATO pop();
13         bool estaPilaVacia();
14         string pilaComoCadena();
15     };
16
17     template<typename TIPODATO>
18     Pila<TIPODATO>::Pila(){
19         Lista<TIPODATO> Lista;
20     }
21
22     template<typename TIPODATO>
23     Pila<TIPODATO>::push(TIPODATO item){
24         this->adjuntar(item);
25     }
26
27     template<typename TIPODATO>
28     TIPODATO Pila<TIPODATO>::pop(){
29         return this->remover(this->tamano() - 1);
30     }
31
32     template<typename TIPODATO>
33     bool Pila<TIPODATO>::estaPilaVacia(){
34         return this->estaVacia();
35     }
36
37     template<typename TIPODATO>
38     string Pila<TIPODATO>::pilaComoCadena(){
39         return this->comoCadena();
40     }
41 }
```



llaves.cpp

```
lista_generica.hpp lista_generica.cpp pila.hpp llaves.cpp
1  #include <iostream>
2  #include <string>
3  #include "pila.hpp"
4
5  using std::string;
6  using std::cin;
7  using std::cout;
8  using std::endl;
9
10 bool estanLlavesBalanceadas(string expresion){
11     Pila<char> unaPila;
12     for(size_t i = 0; i < expresion.length(); i++){
13         char caracter = expresion[i];
14         cout << caracter << endl; // esto se puede borrar despues
15         if(caracter == '{'){
16             unaPila.push(caracter);
17             cout << unaPila.pilaComoCadena() << endl;
18         } else if( caracter == '}' ) {
19             if(unaPila.estaPilaVacia()){
20                 return false;
21             }
22             unaPila.pop();
23             cout << unaPila.pilaComoCadena() << endl;
24         }
25     }
26     return unaPila.estaPilaVacia();
27 }
28
29 int main() {
30     string cadena;
31     cout << "Ingrese una expresion: " << endl;
32     cin >> cadena;
33     if (estanLlavesBalanceadas(cadena)){
34         cout << "Las llaves estan balanceadas.";
35     } else {
36         cout << "Las llaves no estan balanceadas.";
37     }
38     return 0;
39 }
```



cola.hpp

```
lista_generica.hpp lista_generica.cpp pila.hpp llaves.cpp cola.hpp
1  #include "lista_generica.hpp"
2  #include <string>
3
4  template <typename TIPODATO>
5  class Cola : private Lista<TIPODATO>{
6      public:
7          // Constructor
8          Cola();
9          // Agregar un elemento al final de la cola
10         enqueue(TIPODATO item);
11         // Remover un elemento al inicio de la cola
12         TIPODATO dequeue();
13         bool estaColaVacia();
14         string colaComoCadena();
15     };
16
17     template<typename TIPODATO>
18     Cola<TIPODATO>::Cola(){
19         Lista<TIPODATO> Lista;
20     }
21
22     template<typename TIPODATO>
23     Cola<TIPODATO>::enqueue(TIPODATO item){
24         this->adjuntar(item);
25     }
26
27     template<typename TIPODATO>
28     TIPODATO Cola<TIPODATO>::dequeue(){
29         return this->remover(0);
30     }
31
32     template<typename TIPODATO>
33     bool Cola<TIPODATO>::estaColaVacia(){
34         return this->estaVacia();
35     }
36
37     template<typename TIPODATO>
38     string Cola<TIPODATO>::colaComoCadena(){
39         return this->comoCadena();
40     }
41
```



papaCaliente.cpp

```
lista_generica.hpp lista_generica.cpp pila.hpp llaves.cpp cola.hpp papaCaliente.cpp
1 #include <chrono>
2 #include <thread>
3 #include <iostream>
4 #include <vector>
5 #include <cstdlib>
6 #include <ctime>
7 #include "cola.hpp"
8
9 using namespace std;
10
11 int numeroAleatorio(int max);
12 void papaCaliente(const vector<string> &listaNombres);
13
14 int main(){
15     vector<string> jugadores = {"1.Erick", "2.Emmanuel", "3.Briam", "4.Gabriel", "5.Maria", "6.Oscar", "7.Krista",
16                                "8.Eduardo", "9.Alejandra", "10.Carlos", "11.Diego", "12.Manuel", "13.Jose",
17                                "14.Mateo", "15.Fernando", "16.Martinez", "17.Ernesto", "18.Gustavo", "19.Graham"};
18     papaCaliente(jugadores);
19     return 0;
20 }
21
22 int numeroAleatorio(int max){
23     static bool semillaCreada = false;
24     if(!semillaCreada){
25         srand(time(0));
26         semillaCreada = true;
27     }
28     return rand() % max;
29 }
30
31 void papaCaliente(const vector<string> &listaNombres){
32     int cantidadJugadores = listaNombres.size();
33     Cola<string> simulacion;
34     for(int i = 0; i < cantidadJugadores; i++){
35         simulacion.enqueue(listaNombres[i]);
36     }
37
38     for(int i = cantidadJugadores; i > 1; i--){
39         cout << "Quien tiene la papa caliente?" << endl;
40         for(int pases = numeroAleatorio(2*cantidadJugadores); pases > 0; pases--){
41             string jugador = simulacion.dequeue();
42             cout << jugador << endl;
43             simulacion.enqueue(jugador);
44         }
45     }
46 }
```



```

lista_generica.hpp lista_generica.cpp pila.hpp llaves.cpp cola.hpp papaCaliente.cpp
12 void papaCaliente(const vector<string> &listaNombres);
13
14 int main(){
15     vector<string> jugadores = {"1.Erick", "2.Emmanuel", "3.Briam", "4.Gabriel", "5.Maria", "6.Oscar", "7.Krista",
16                               "8.Eduardo", "9.Alejandra", "10.Carlos", "11.Diego", "12.Manuel", "13.Jose",
17                               "14.Mateo", "15.Fernando", "16.Martinez", "17.Ernesto", "18.Gustavo", "19.Graham"};
18     papaCaliente(jugadores);
19     return 0;
20 }
21
22 int numeroAleatorio(int max){
23     static bool semillaCreada = false;
24     if(!semillaCreada){
25         srand(time(0));
26         semillaCreada = true;
27     }
28     return rand() % max;
29 }
30
31 void papaCaliente(const vector<string> &listaNombres){
32     int cantidadJugadores = listaNombres.size();
33     Cola<string> simulacion;
34     for(int i = 0; i < cantidadJugadores; i++){
35         simulacion.enqueue(listaNombres[i]);
36     }
37
38     for(int i = cantidadJugadores; i > 1; i--){
39         cout << "Quien tiene la papa caliente?" << endl;
40         for(int pases = numeroAleatorio(2*cantidadJugadores); pases > 0; pases--){
41             string jugador = simulacion.dequeue();
42             cout << jugador << endl;
43             simulacion.enqueue(jugador);
44             this_thread::sleep_for(chrono::milliseconds(1000));
45         }
46         string funado = simulacion.dequeue();
47         cout << funado << endl;
48         cout << "Alto!!! " << funado << " sale del juego..." << endl << endl;
49     }
50     string ganador = simulacion.dequeue();
51     cout << ganador << " gana el juego." << endl;
52 }

```

