30-9-2021

Lista estática mejorada

Practica: 3

Materia: Estructura de datos

Sección: D01.

Código: 216584703

Carrera: Ingeniería en computación.

Nombre alumno: Padilla Pérez Jorge Daray

Nombre profesor: Julio Esteban Valdes Lopez

Introducción

Mi práctica consiste en la implementación de una lista estructurada en la cual se implementan lo típico de un TDA Lista lo que viene siendo insertar en orden como si fuera una lista consultar elemento remover etc. En esta practica se logro hacer todo lo pedido para esta, además de poder realizar la función de localizar posición por elemento la cual me costó más, pero si se logró.

También se pudo lograr recorrer los datos una vez ingresados en la lista, aunque no se implementaron las funciones anterior y siguiente creo que es una función fácil de comprender y de hacer por lo que me parece bien que se hayan saltado esas 2 funciones que en si no existe una utilidad buena para ellos.

Además de la practica 2 se agregaron métodos de búsqueda lineal y binario, también de métodos de ordenamiento tanto iterativos en este caso (3), y recursivos en este caso (2).

Pantallazos

```
1   #include <iostream>
2   #include <cstring>
3
4   #define TAMLISTA 10
5
6   using namespace std;
7
8   void menu();
9
10  typedef int tipo_dato,temp;
11
12  struct Lista{
13  tipo_dato datos[TAMLISTA];
14  void inicializa();
15  bool vacia();
16  bool llena();
17  void insertar(int pos, tipo_dato elem);
18  void elimina(int pos);
19  int ultimo;
20  int primero;
21  void localiza(tipo_dato dato);
22  void recupera(int pos);
23  void imprimir();
24  void anular();
25  void burbuja_mejorada();
26  void insercion();
27  void seleccion();
28  void mezcla( int , int );
29  void quicksort ( int , int );
30  bool lista_ordenada ();
31  temp arreglo_copia[TAMLISTA];
```

Estructura de la lista junto con el tamaño de la lista y donde se implementan todas las funciones del programa.

```
33  Lista(){
34  inicializa();
35  }
36  };
37
38  void Lista::inicializa(){
39  ultimo = -1;
40  primero = 0;
41  }
42
43  bool Lista::vacia(){
44  return ultimo == -1;
45  }
46
47  bool Lista::llena(){
48  return ultimo == TAMLISTA - 1;
49  }
50
```

Funciones principales de una lista, mas el constructor de la lista.

```
52  void Lista::insertar(int pos, tipo_dato elem){
53
54      if (llena() || pos < 0 || pos > ultimo + 1){
55
56          cout<<"Ingresa un elemento consecutivo valido"<<endl;
57          system("pause");
58          return;
59      }
60
61      for(int i = ultimo+1 ; i > pos ; i-- ){
62
63          datos[i] = datos[i - 1];
64      }
65
66          datos[pos] = elem;
67
68          ultimo++;
69  }
```

Funcion insertar en la cual se ponen por posicion, cuidando la continuidad de esta, ademas de recorrer los datos de la lista si se inserta en una posicion anterior al ultimo.

```
72  void Lista::elimina(int pos){
73      if (vacia() || pos < 0 || pos > ultimo ){
74
75          cout<<"La lista esta vacia"<<endl;
76          system("pause");
77
78          return;
79      }
80
81      for (int i = pos ; i <= ultimo ; i++){
82
83      datos[i] = datos[i + 1];
84
85      }
86      ultimo--;
87  }
```

Funcion elimina la cual simplemente hace que los saque de la lista y ultimo disminuye al eliminar el elmento de la posicion dada

```
89  void Lista::imprimir(){
90      if (vacia()){
91
92          cout<<"La lista esta vacia"<<endl;
93          system("pause");
94          return;
95      }
96      for(int i = primero ; i <= ultimo ; i++){
97      cout<<"Posicion numero: "<<i<<"\n Dato: "<<datos[i]<< " "<<endl;
98      }
99  }
```

Función imprimir no tiene mayor complejidad mas que un for que vaya imprimiendo los datos conforme las posiciones.

```
101  void Lista::recupera(int pos){
102      if (vacia() || pos < 0 || pos > ultimo ){
103
104          cout<<"La lista esta vacia"<<endl;
105          system("pause");
106
107          return;
108      }
109          cout<<"Posicion numero: "<<pos<<"\n Dato: "<<datos[pos]<<" "<<endl;
110          cout<<endl;
111
112  }
```

Función recupera recupera el dato guardado en la posición que se le solicita, simplemente recibe la posición que se agrega en el menú y esta se imprime.

```
114  bool Lista::lista_ordenada(){
115      int j = primero, i = ultimo;
116          while (j < i){
117              if (datos[j] > datos[j+1]){
118                      return 1;
119              }
120              j++;
121          }
122          i--;
123      return 0
124      ;
125  }
```

Función booleana que revisa si la lista esta ordenada empezando por el primer dato de la lista y va comparando con el siguiente si es mayor.

```
127  void Lista::localiza(tipo_dato dato){
128      int opc;
129      if (vacia()){
130
131          cout<<"La lista esta vacia"<<endl;
132          system("pause");
133
134          return;
135      }
136      system("cls");
137      cout<<"Que metodo quieres usar :"<<endl;
138      cout<<"1) Metodo lineal"<<endl;
139      cout<<"2) metodo binario"<<endl;
140      cout<<"2) Seleccione opcion : "<<endl;
141      cin>>opc;
142      switch(opc){
```

Función localiza que recupera la posición conforme al dato, aquí nomas se aprecia el menú para el usuario.

```
case 1:{
    cout<<"\n Metodo lineal\n\n"<<endl;
    int i;
    bool encontrado;
    for ( i = primero ; i <= ultimo ; i++ ){
        if (datos[i] == dato){
            cout<<" Posicion numero: "<<i<<"\n Dato: "<<datos[i]<<" "<<endl;
            cout<<endl;
            encontrado = true;
        }
    }
        if (encontrado == false){
            cout<<" Dato no encontrado: "<<endl;
        }
}break;
```

Case 1 de la función recupera la cual es el método lineal y es tan fácil como buscar el dato a través de toda la lista y si se encuentra el dato que puso el usuario se imprime si no se pone dato no encontrado.

```
160         case 2:{
161     if (lista_ordenada()){
162         cout<<" La lista no esta ordenada"<<endl;
163         return;
164     }
165         int i = 0, j = ultimo, m;
166         bool encontrado;
167         while ( i <= j ){
168             m = ( i + j )/2;
169             if ( datos[m] == dato ){
170                 cout<<" Posicion numero: "<<m<<"\n Dato: "<<datos[m]<<" "<<endl;
171                 cout<<endl;
172                 encontrado = true;
173             }
174             if ( dato < datos[m] ){
175                 j = m - 1;
176             }
177             else{
178                 i = m + 1;
179             }
180         }
181         if (encontrado == false){
182             cout<<" Dato no encontrado: "<<endl;
183         }
184         }break;
185     }
186 }
```

Case 2 en el cual se manda a llamar a la función de lista ordenada para saber si la lista esta ordenada ya que este método binario así lo requiere, una vez se comprobó que si se inicializa la función.

```
191  void Lista::burbuja_mejorada(){
192  if (vacia()){
193
194          cout<<"La lista esta vacia"<<endl;
195          system("pause");
196
197          return;
198      }
199          int i = ultimo, j;
200          int aux;
201          bool cambio;
202          do{
203              cambio = false;
204              j = 0;
205              while (j < i){
206                  if (datos[j] > datos[j+1]){
207                      aux = datos[j];
208                      datos[j] = datos[j+1];
209                      datos[j+1] = aux;
210                      cambio = true;
211                  }
212                  j++;
213              }
214              i--;
215          }while(cambio);
216  }
```

Funcion metodo de ordenamiento burbuja(iterativo), el cual hace parecido como la manera de validar si la lista esta ordenada pero en vez de dar aviso intercambia los valores para que queden acomodados.

```
218  void Lista::insercion(){
219  int i = 1, j;
220  int aux;
221      while(i <= ultimo){
222          aux = datos[i];
223          j = i;
224          while( j > 0 && aux < datos[j-1]){
225              datos[j] = datos[j-1];
226              j--;
227          }
228          if(i != j){
229              datos[j] = aux;
230          }
231          i++;
232      }
233  }
```

Funcion metodo de ordenamiento insercion(iterativo), el cual inicia el iterador en el primer dato y se garega un auxiliar para poder darle condicion al mientras, el cual se encarga de cambiar los datos por su posicion correcta.

```
235  void Lista::seleccion(){
236      int i = 0, j, m;
237      int aux;
238      while( i < ultimo ){
239          m = i;
240          j = i +1;
241          while( j <= ultimo ){
242              if( datos[j] < datos[m] ){
243                  m = j;
244              }
245              j++;
246          }
247              if( m!= i ){
248                  aux = datos[i];
249                  datos[i] = datos[m];
250                  datos[m] = aux;
251              }
252          i++;
253      }
254
255  }
```

Función método de ordenamiento selección(iterativo).

```
257 void Lista::mezcla( int primero, int ultimo ){
258     if ( primero >= ultimo ){
259         return;
260     }
261 
262     int medio = ((primero + ultimo)/2);
263     mezcla ( primero, medio );
264     mezcla ( medio + 1, ultimo );
265 
266     for ( int c = primero ; c <= ultimo ; c++ ){
267         arreglo_copia[c] = datos[c];
268     }
269 
270     int i = primero, j = medio + 1, x = primero;
271     while ( i <= medio && j <= ultimo ){
272         while ( i <= medio && arreglo_copia[i] <= arreglo_copia[j] ){
273             datos[x++] = arreglo_copia[i++];
274         }
275         if ( i <= medio ){
276             while ( j <= ultimo && arreglo_copia[j] <= arreglo_copia[i] ){
277                 datos[x++] = arreglo_copia[j++];
278             }
279         }
280     }
281     while ( i <= medio ){
282         datos[x++] = arreglo_copia[i++];
283     }
284     while ( j <= ultimo ){
285         datos[x++] = arreglo_copia[j++];
286     }
287 }
```

Función método de ordenamiento mezcla(recursivo), el cual recibe de parámetro el extremo izquierdo y el derecho de la lista en este caso es igual a primero y ultimo de esta, para este método se copia la lista, por lo que esta en la estructura principal de la lista.

```
289  void Lista::quicksort( int primero, int ultimo ){
290      int aux, j, i;
291      if ( primero >= ultimo ){
292          return;
293      }
294
295      aux = datos[(primero + ultimo)/2];
296      datos[(primero + ultimo)/2] = datos[ultimo];
297      datos[ultimo] = aux;
298
299
300      i = primero, j = ultimo;
301      while ( i < j ){
302          while ( i < j && datos[i] <= datos[ultimo] ){
303              i++;
304          }
305          while ( i < j && datos[j] >= datos[ultimo] ){
306              j--;
307          }
308          if ( i != j ){
309              aux = datos[i];
310              datos[i] = datos[j];
311              datos[j] = aux;
312          }
313      }
314          if ( i != ultimo ){
315              aux = datos[i];
316              datos[i] = datos[ultimo];
317              datos[ultimo] = aux;
318          }
319      quicksort( primero ,i-1 );
320      quicksort( i+1 ,ultimo );
321
```

Función recursiva Quicksort(recursiva), la cual al igual que mezcla recibe el extremo derecho e izquierdo.

```
struct Lista mi_lista;
int main(){
int opc=0;
    do{
    system("cls");
     menu();
    cout<<"Continuar 1 salir 12:"<<endl;cin>>opc;
   }while(opc!=12);
    system("pause>>cls");
return 0;
}
```

Se inicializa una variable mi_lista de la estructura principal Lista, ademas de un do-while el cual muestra el menu principal y hace repetir el ciclo hasta que se ponga un 12 para la salida.

```
void menu(){
int opc=0;
int dato,pos;
    cout<<" Practica 2"<<endl;
    cout<<"1) Insertar elemento (Por posicion)"<<endl;
    cout<<"2) Eliminar elemento"<<endl;
    cout<<"3) recupera elemento"<<endl;
    cout<<"4) localiza posicion"<<endl;
    cout<<"5) Imprimir Lista"<<endl;
    cout<<"6) Anular Lista"<<endl;
    cout<<"7) Ordenamiento_Burbuja"<<endl;
    cout<<"8) Ordenamiento_Insercion"<<endl;
    cout<<"9) Ordenamiento_Seleccion"<<endl;
    cout<<"10) Ordenamiento_Mezcla"<<endl;
    cout<<"11) Ordenamiento_Quicksort"<<endl;
    cout<<"12) Salir"<<endl;
    cout<<"Seleccione opcion:"<<endl;
    cin>>opc;
    switch(opc){
```

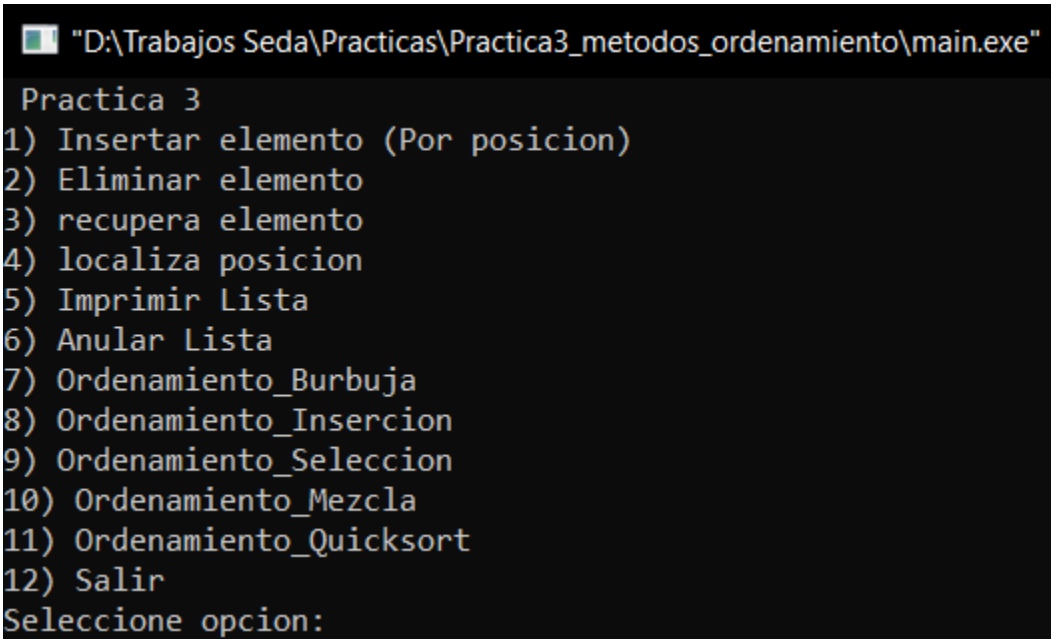
El menú no tiene mayor complicación.

```
356         case 1: {
357         cout<<"En que posicion desea insertar el elemento :"<<endl;cin>>pos;
358         cout<<"Inserte elemento: "<<endl;cin>>dato;
359         mi_lista.insertar(pos,dato);
360 
361         }break;
362         case 2: {
363         cout<<"Que posicion deseas eliminar :"<<endl;cin>>pos;
364         mi_lista.elimina(pos);
365         }break;
366         case 3: {
367         cout<<"Que posicion deseas consultar :"<<endl;cin>>pos;
368         mi_lista.recupera(pos);
369         }break;
370 
371         case 4: {
372         cout<<"Que dato deseas buscar :"<<endl;cin>>dato;
373         mi_lista.localiza(dato);
374         }break;
375 
376         case 5: {
377         mi_lista.imprimir();
378         }break;
379 
380         case 6: {
381         mi_lista.anular();
382         }break;
383 
384         case 7: {
385         mi_lista.burbuja_mejorada();
386         }break;
```

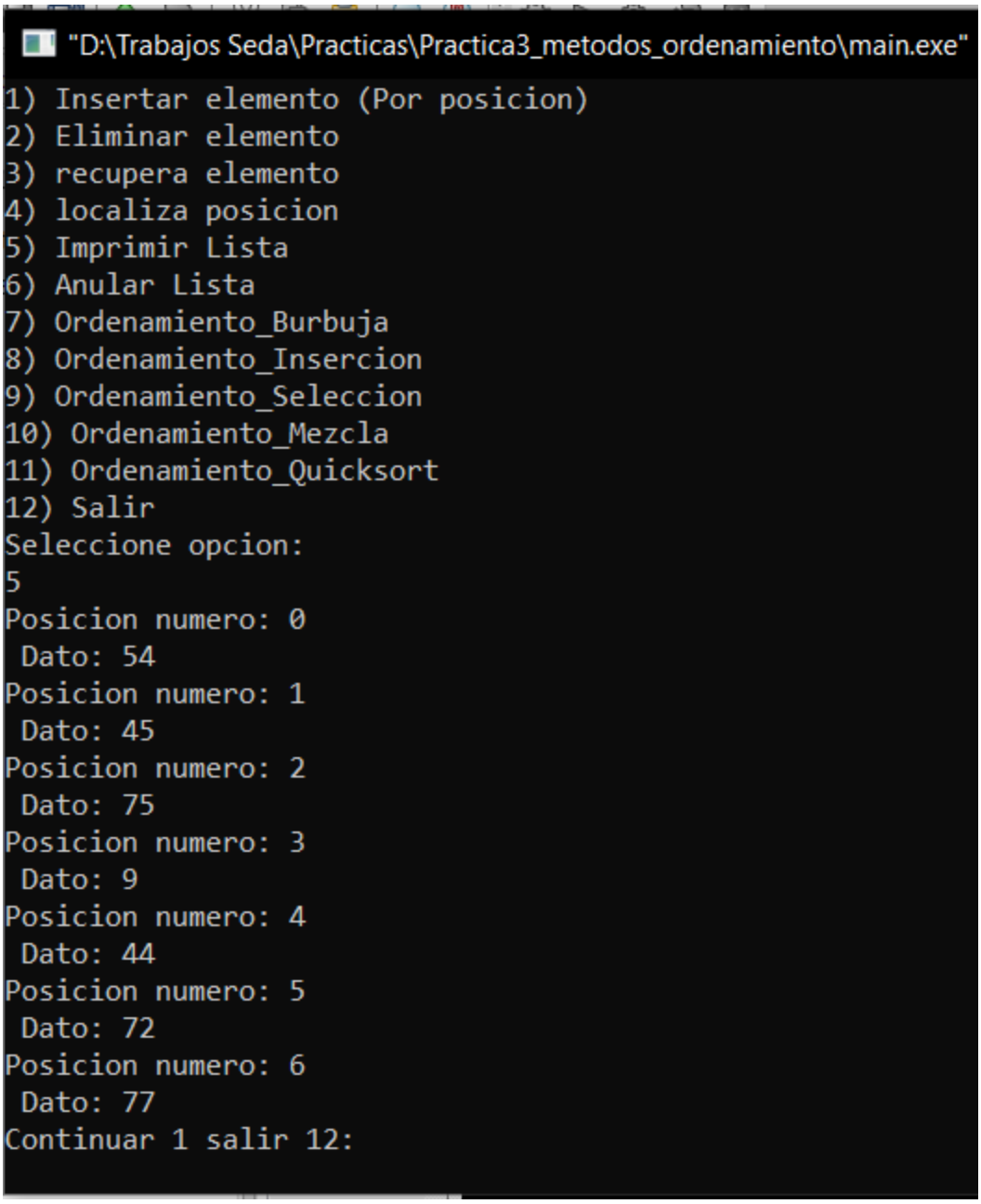
Las opciones del menu.

```
388         case 8: {
389         mi_lista.insercion();
390         }break;
391
392         case 9: {
393         mi_lista.seleccion();
394         }break;
395
396         case 10: {
397         mi_lista.mezcla(mi_lista.primero, mi_lista.ultimo);
398         }break;
399
400         case 11: {
401         mi_lista.quicksort(mi_lista.primero, mi_lista.ultimo);
402         }break;
403
404         case 12:break;
405
406         default:
407             cout<<"La opcion: "<<opc<<"No existe"<<endl;
408     }
409 }
```

Final del código explicado.



Menu de opciones para el usuario.



Se insertan elementos en la lista y se imprimen.

```
Seleccione opcion:
2
Que posicion deseas eliminar :
2
Continuar 1 salir 12:
```

Se elimina la posicion 2 de la lista en este caso 75

```
Seleccione opcion:
5
Posicion numero: 0
 Dato: 54
Posicion numero: 1
 Dato: 45
Posicion numero: 2
 Dato: 9
Posicion numero: 3
 Dato: 44
Posicion numero: 4
 Dato: 72
Posicion numero: 5
 Dato: 77
Continuar 1 salir 12:
```

Al mostrarlos nuevamente se aprecia que ya fue eliminado.

```
Seleccione opcion:
3
Que posicion deseas consultar :
3
Posicion numero: 3
 Dato: 44

Continuar 1 salir 12:
```

Al recuperar el elemento de la posicion 3 arroja el dato junto la psocion buscada siguiendo la lista que tenemos en este caso el dato 44.

```
Seleccione opcion:
4
Que dato deseas buscar :
44
```

En este caso al escoger la opcion 4 pide el dato que queremos localizar.

```
Que metodo quieres usar :
1) Metodo lineal
2) metodo binario
2) Seleccione opcion :
1

 Metodo lineal


 Posicion numero: 3
 Dato: 44

Continuar 1 salir 12:
```

Una vez hecho eso nos da la opcion de escoger de que metodo encontrar el dato, en este caso se escoge la primera opcion, la cual arroja la posicion en la que se ecuentra y refairma el dato que contiene abajo.

```
Que metodo quieres usar :
1) Metodo lineal
2) metodo binario
2) Seleccione opcion :
2
 La lista no esta ordenada
Continuar 1 salir 12:
```

Si ahorita mismo queremos buscarlo por metodo binario no se puede ya que necesita estar ordenada la lista para esto y manda el mensaje.

```
Seleccione opcion:
5
Posicion numero: 0
 Dato: 9
Posicion numero: 1
 Dato: 44
Posicion numero: 2
 Dato: 45
Posicion numero: 3
 Dato: 54
Posicion numero: 4
 Dato: 72
Posicion numero: 5
 Dato: 77
Continuar 1 salir 12:
```

Hacemos un ordenamiento de los datos que tenemos con cualquier metodo de los 5, los 5 funcionan.

```
Que metodo quieres usar :
1) Metodo lineal
2) metodo binario
2) Seleccione opcion :
2
 Posicion numero: 1
 Dato: 44

Continuar 1 salir 12:
```

Y ahora si encuntra el valor ya que si esta ordenada la lista.

```
Seleccione opcion:
1
En que posicion desea insertar el elemento :
0
Inserte elemento:
20
```

Si ahora insertamos un elemnto en la posicion 0 se recorren los valores despues de este, y ademas volvemos a desordenar la lista.

```
Seleccione opcion:
5
Posicion numero: 0
 Dato: 20
Posicion numero: 1
 Dato: 9
Posicion numero: 2
 Dato: 44
Posicion numero: 3
 Dato: 45
Posicion numero: 4
 Dato: 54
Posicion numero: 5
 Dato: 72
Posicion numero: 6
 Dato: 77
Continuar 1 salir 12:
```

aquí se muestra lo que se explico.

```
Que metodo quieres usar :
1) Metodo lineal
2) metodo binario
2) Seleccione opcion :
2
 La lista no esta ordenada
Continuar 1 salir 12:
```

Entonces si queremos buscar el mismo dato “44”, por metodo binario ya no esta ordenada nuevamente y manda el mensaje.

```
Seleccione opcion:
5
Posicion numero: 0
 Dato: 9
Posicion numero: 1
 Dato: 20
Posicion numero: 2
 Dato: 44
Posicion numero: 3
 Dato: 45
Posicion numero: 4
 Dato: 54
Posicion numero: 5
 Dato: 72
Posicion numero: 6
 Dato: 77
Continuar 1 salir 12:
```

Volvemos a ordenar la lista con cualquier metodo.

```
Que metodo quieres usar :
1) Metodo lineal
2) metodo binario
2) Seleccione opcion :
2
 Posicion numero: 2
 Dato: 44

Continuar 1 salir 12:
```

y ahora si la encuentra pero ahora en la posicion 2, no como al inicio en la posicion 1, ya que el dato 20 ahora acupa ese lugar.

Conclusión

Respecto a la realización del código concluyo que se logró bien la realización de este programa ya que a lo que se puede apreciar funciona de manera correcta como debería hacerlo una lista además de implementar bien sus funciones recalco mis conocimientos, y estoy abierto a posibles errores que pueda tener el programa al momento de que el profe la evalúe.

También se concluye que, aunque no parezca que tiene una utilidad buena al momento de referirse a trabajo si lo piensas un rato le hayas muchas utilidades importantes tanto que pensándolo muchas aplicaciones y juegos realizan una lista implementada en estos mismos por lo cual espero poder aprender mas sobre las listas y los tipos de datos abstractos.

Al momento de realizar el uso de métodos de ordenamientos y de búsquedas mejoro muchísimo el código base de la practica 2 y se le agregan más utilidades además de lograr un nuevo reto que fue su implementación.

Código fuente

```
1   #include <iostream>
2   #include <cstring>
3
4   #define TAMLISTA 10
5
6   using namespace std;
7
8   void menu();
9
10  typedef int tipo_dato,temp;
11
12  struct Lista{
13  tipo_dato datos[TAMLISTA];
14  void inicializa();
15  bool vacia();
16  bool llena();
17  void insertar(int pos, tipo_dato elem);
18  void elimina(int pos);
19  int ultimo;
20  int primero;
21  void localiza(tipo_dato dato);
22  void recupera(int pos);
23  void imprimir();
24  void anular();
25  void burbuja_mejorada();
26  void insercion();
27  void seleccion();
28  void mezcla( int , int );
```

```
29  void quicksort ( int , int );
30  bool lista_ordenada ();
31  temp arreglo_copia[TAMLISTA];
32
33  Lista(){
34  inicializa();
35  }
36  };
37
38  void Lista::inicializa(){
39  ultimo = -1;
40  primero = 0;
41  }
42
43  bool Lista::vacia(){
44  return ultimo == -1;
45  }
46
47  bool Lista::llena(){
48  return ultimo == TAMLISTA - 1;
49  }
50
51
52  void Lista::insertar(int pos, tipo_dato elem){
53
54      if (llena() || pos < 0 || pos > ultimo + 1){
55
56          cout<<"Ingresa un elemento consecutivo valido"<<endl;
57          system("pause");
58          return;
```

```
59      }
60
61      for(int i = ultimo+1 ; i > pos ; i-- ){
62
63          datos[i] = datos[i - 1];
64      }
65
66          datos[pos] = elem;
67
68          ultimo++;
69  }
70
71
72  void Lista::elimina(int pos){
73      if (vacia() || pos < 0 || pos > ultimo ){
74
75          cout<<"La lista esta vacia"<<endl;
76          system("pause");
77
78          return;
79      }
80
81      for (int i = pos ; i <= ultimo ; i++){
82
83      datos[i] = datos[i + 1];
84
85      }
86      ultimo--;
87  }
88
```

```
89  void Lista::imprimir(){
90      if (vacia()){
91
92          cout<<"La lista esta vacia"<<endl;
93          system("pause");
94          return;
95      }
96      for(int i = primero ; i <= ultimo ; i++){
97      cout<<"Posicion numero: "<<i<<"\n Dato: "<<datos[i]<< " "<<endl;
98      }
99  }
100
101 void Lista::recupera(int pos){
102     if (vacia() || pos < 0 || pos > ultimo ){
103
104         cout<<"La lista esta vacia"<<endl;
105         system("pause");
106
107         return;
108     }
109         cout<<"Posicion numero: "<<pos<<"\n Dato: "<<datos[pos]<<" "<<endl;
110         cout<<endl;
111
112 }
113
114 bool Lista::lista_ordenada(){
115     int j = primero, i = ultimo;
116         while (j < i){
117             if (datos[j] > datos[j+1]){
118                     return 1;
```

```
119             }
120             j++;
121         }
122         i--;
123     return 0
124     ;
125 }
126
127 void Lista::localiza(tipo_dato dato){
128     int opc;
129     if (vacia()){
130
131         cout<<"La lista esta vacia"<<endl;
132         system("pause");
133
134         return;
135     }
136       system("cls");
137     cout<<"Que metodo quieres usar :"<<endl;
138     cout<<"1) Metodo lineal"<<endl;
139       cout<<"2) metodo binario"<<endl;
140       cout<<"2) Seleccione opcion : "<<endl;
141       cin>>opc;
142       switch(opc){
143
144         case 1:{
145             cout<<"\n Metodo lineal\n\n"<<endl;
146             int i;
147             bool encontrado;
148             for ( i = primero ; i <= ultimo ; i++ ){
```

```
149                 if (datos[i] == dato){
150                     cout<<" Posicion numero: "<<i<<"\n Dato: "<<datos[i]<<"
151 "<<endl;
152                     cout<<endl;
153                     encontrado = true;
154                 }
155             }
156                 if (encontrado == false){
157                     cout<<" Dato no encontrado: "<<endl;
158                 }
159         }break;
160 
161         case 2:{
162     if (lista_ordenada()){
163         cout<<" La lista no esta ordenada"<<endl;
164         return;
165     }
166         int i = 0, j = ultimo, m;
167         bool encontrado;
168         while ( i <= j ){
169             m = ( i + j )/2;
170             if ( datos[m] == dato ){
171                 cout<<" Posicion numero: "<<m<<"\n Dato: "<<datos[m]<<"
172 "<<endl;
173                 cout<<endl;
174                 encontrado = true;
175             }
176             if ( dato < datos[m] ){
177                 j = m - 1;
178             }
```

```
179             else{
180                 i = m + 1;
181             }
182         }
183         if (encontrado == false){
184             cout<<" Dato no encontrado: "<<endl;
185         }
186         }break;
187       }
188 }
189 
190 void Lista::anular(){
191 ultimo = -1;
192 }
193 
194 void Lista::burbuja_mejorada(){
195 if (vacia()){
196 
197         cout<<"La lista esta vacia"<<endl;
198         system("pause");
199 
200         return;
201     }
202         int i = ultimo, j;
203         int aux;
204         bool cambio;
205         do{
206             cambio = false;
207             j = 0;
208             while (j < i){
```

```
209                 if (datos[j] > datos[j+1]){
210                     aux = datos[j];
211                     datos[j] = datos[j+1];
212                     datos[j+1] = aux;
213                     cambio = true;
214                 }
215                 j++;
216             }
217             i--;
218         }while(cambio);
219 }
220
221 void Lista::insercion(){
222 int i = 1, j;
223 int aux;
224     while(i <= ultimo){
225         aux = datos[i];
226         j = i;
227         while( j > 0 && aux < datos[j-1]){
228             datos[j] = datos[j-1];
229             j--;
230         }
231         if(i != j){
232             datos[j] = aux;
233         }
234         i++;
235     }
236 }
237
238 void Lista::seleccion(){
```

```
239       int i = 0, j, m;
240       int aux;
241       while( i < ultimo ){
242           m = i;
243           j = i +1;
244           while( j <= ultimo ){
245               if( datos[j] < datos[m] ){
246                   m = j;
247               }
248               j++;
249           }
250               if( m!= i ){
251                   aux = datos[i];
252                   datos[i] = datos[m];
253                   datos[m] = aux;
254               }
255           i++;
256       }
257
258   }
259
260   void Lista::mezcla( int primero, int ultimo ){
261       if ( primero >= ultimo ){
262           return;
263       }
264
265       int medio = ((primero + ultimo)/2);
266       mezcla ( primero, medio );
267       mezcla ( medio + 1, ultimo );
268
```

```
269       for ( int c = primero ; c <= ultimo ; c++ ){
270           arreglo_copia[c] = datos[c];
271       }
272
273       int i = primero, j = medio + 1, x = primero;
274       while ( i <= medio && j <= ultimo ){
275           while ( i <= medio && arreglo_copia[i] <= arreglo_copia[j] ){
276               datos[x++] = arreglo_copia[i++];
277           }
278           if ( i <= medio ){
279               while ( j <= ultimo && arreglo_copia[j] <= arreglo_copia[i] ){
280                   datos[x++] = arreglo_copia[j++];
281               }
282           }
283       }
284       while ( i <= medio ){
285           datos[x++] = arreglo_copia[i++];
286       }
287       while ( j <= ultimo ){
288           datos[x++] = arreglo_copia[j++];
289       }
290   }
291
292   void Lista::quicksort( int primero, int ultimo ){
293       int aux, j, i;
294       if ( primero >= ultimo ){
295           return;
296       }
297
298       aux = datos[(primero + ultimo)/2];
```

```
299       datos[(primero + ultimo)/2] = datos[ultimo];
300       datos[ultimo] = aux;
301
302
303       i = primero, j = ultimo;
304       while ( i < j ){
305           while ( i < j && datos[i] <= datos[ultimo] ){
306               i++;
307           }
308           while ( i < j && datos[j] >= datos[ultimo] ){
309               j--;
310           }
311           if ( i != j ){
312               aux = datos[i];
313               datos[i] = datos[j];
314               datos[j] = aux;
315           }
316       }
317           if ( i != ultimo ){
318               aux = datos[i];
319               datos[i] = datos[ultimo];
320               datos[ultimo] = aux;
321           }
322       quicksort( primero ,i-1 );
323       quicksort( i+1 ,ultimo );
324
325   }
326
327   struct Lista mi_lista;
328   int main(){
```

```
int opc=0;
    do{
      system("cls");
       menu();
    cout<<"Continuar 1 salir 12:"<<endl;cin>>opc;
   }while(opc!=12);
    system("pause>>cls");
return 0;
}

void menu(){
int opc=0;
int dato,pos;
      cout<<" Practica 3"<<endl;
      cout<<"1) Insertar elemento (Por posicion)"<<endl;
      cout<<"2) Eliminar elemento"<<endl;
      cout<<"3) recupera elemento"<<endl;
      cout<<"4) localiza posicion"<<endl;
      cout<<"5) Imprimir Lista"<<endl;
    cout<<"6) Anular Lista"<<endl;
    cout<<"7) Ordenamiento_Burbuja"<<endl;
    cout<<"8) Ordenamiento_Insercion"<<endl;
    cout<<"9) Ordenamiento_Seleccion"<<endl;
    cout<<"10) Ordenamiento_Mezcla"<<endl;
    cout<<"11) Ordenamiento_Quicksort"<<endl;
    cout<<"12) Salir"<<endl;
    cout<<"Seleccione opcion:"<<endl;
    cin>>opc;
      switch(opc){

```

```
359             case 1: {
360         cout<<"En que posicion desea insertar el elemento :"<<endl;cin>>pos;
361         cout<<"Inserte elemento: "<<endl;cin>>dato;
362         mi_lista.insertar(pos,dato);
363 
364         }break;
365         case 2: {
366         cout<<"Que posicion deseas eliminar :"<<endl;cin>>pos;
367         mi_lista.elimina(pos);
368         }break;
369         case 3: {
370         cout<<"Que posicion deseas consultar :"<<endl;cin>>pos;
371         mi_lista.recupera(pos);
372         }break;
373 
374         case 4: {
375         cout<<"Que dato deseas buscar :"<<endl;cin>>dato;
376         mi_lista.localiza(dato);
377         }break;
378 
379         case 5: {
380         mi_lista.imprimir();
381         }break;
382 
383         case 6: {
384         mi_lista.anular();
385         }break;
386 
387         case 7: {
388         mi_lista.burbuja_mejorada();
```

```
389             }break;
390 
391             case 8: {
392             mi_lista.insercion();
393             }break;
394 
395             case 9: {
396             mi_lista.seleccion();
397             }break;
398 
399             case 10: {
400             mi_lista.mezcla(mi_lista.primero, mi_lista.ultimo);
401             }break;
402 
403             case 11: {
404             mi_lista.quicksort(mi_lista.primero, mi_lista.ultimo);
405             }break;
406 
407             case 12:break;
408 
409             default:
410                 cout<<"La opcion: "<<opc<<"No existe"<<endl;
411         }
412 }
```