Actividad 01 - Repaso

# AVILA CALDERON JOSE FRANCISCO

**SEMINARIO DE SOLUCION DE PROBLEMAS DE ALGORITMIA**

# Lineamientos de evaluación

# Desarrollo

La actividad fue desarrollada con ayuda del libro de programación orientada a objetos donde me apoye para desarrollar el código a partir de ejemplos de listas que tiene este mismo libro, adaptándolo a las clases y objetos que se necesitaba para la actividad

# Conclusiones

# Una de las dificultades al momento de crear el programa fue implementar las clases y objetos, al tratar de adaptar los ejemplos del libro y añadir mas objetos generaba errores así como implementar las funciones de lectura y escritura

# Referencias

Guardati, S. (2022, 17 septiembre). *Estructura de Datos Orientada a Objetos: Algoritmos con C++* (1st.). PEARSON EDUCACION DE MEXICO -75.

Crear y Escribir un archivo es C++ (respaldar) Michel Davalos Boites <https://youtu.be/g3gtAhHgYcA>

Leer un archivo en C++ (recuperar) Michel Davalos Boites <https://www.youtube.com/watch?v=QqU8emKoiIA>

# Código

1 #include <iostream>

2 #include <fstream>

3 #include "ListasSimLig.h"

4 #include "Productos.h"

5

6 **using namespace std**;

7

8 **void** CargarLista(Producto &ObjProd,Lista<Producto> &ListaProds, **fstream** &ArchivoProductoSalida)

9 {

10 ArchivoProductoSalida.seekg(0);

11 ArchivoProductoSalida.read(**reinterpret\_cast** <**char**\*>(&ObjProd),**sizeof**(Producto));

12

13 **while**(!ArchivoProductoSalida.eof())

14 {

15 **if**(ObjProd.RegresaClave() != 0)

16 {

17 ListaProds.InsertaOrdenCrec(ObjProd);

18 }

19 ArchivoProductoSalida.read(**reinterpret\_cast** <**char**\*>(&ObjProd),**sizeof**(Producto));

20 }

21 }

22

23 **void** EliminarElemento(Producto &ObjProd,Lista<Producto> &ListaProds, **fstream** &ArchivoProductoSalida)

24 {

25 **int** Clave=0,Res=0;

26

27 **cout**<<"\n\nIngrese La Clave Del Producto A Eliminar: ";

28 **cin**>>Clave;

29

30 ArchivoProductoSalida.seekp((Clave - 1 ) \* **sizeof**(Producto));

31 ArchivoProductoSalida.read(**reinterpret\_cast** <**char** \*>(&ObjProd),**sizeof**(Producto));

32

33 **if**(ObjProd.RegresaClave() != 0){

34 Producto ProdBlac;

35

36 **cout**<<"\nSe Encontro La Clave: '"<<Clave<<"'"<<**endl**;

37

38 ArchivoProductoSalida.seekp((Clave - 1) \* **sizeof**(Producto));

39 ArchivoProductoSalida.write(**reinterpret\_cast** < **const char** \*>(&ProdBlac),**sizeof**(Producto));

40 }

41 Producto Produc(Clave,"","", 0);

42 Res= ListaProds.EliminaUnNodo(Produc);

43 **switch** (Res)

44 {

45 **case** 1:

46 **cout**<<"\n\nEl Producto Ha Hizo Eliminado.\n";

47 **break**;

48 **case** 0:

49 **cout**<<"\n\nEse Producto No Se Encuentra Registrado.\n";

50 **break**;

51 **case** -1:

52 **cout**<<"\n\nNo Hay Productos Registrados.\n";

53

54 }

55 }

56

57 **void** Escribir(Producto &ObjProd,Lista<Producto> &ListaProds,**fstream** &ArchivoProductoSalida)

58 {

59 **cout**<<"\n\nIngrese Datos Del Producto A Registrar:";

60 **cin**>>ObjProd;

61

62 **if**(ListaProds.BuscaOrdenada(ObjProd)== **nullptr**)

63 {

64 ListaProds.InsertaOrdenCrec(ObjProd);

65 }

66 **else**

67 {

68 **cout** << "\n\nClave Ya Registrada" << **endl**;

69 }

70 ArchivoProductoSalida.seekp((ObjProd.RegresaClave() - 1) \* **sizeof**(Producto));

71 ArchivoProductoSalida.write(**reinterpret\_cast** < **const char**\*>(&ObjProd),**sizeof**(Producto));

72 }

73

74 **int** Menu()

75 {

76 **int** Opc;

77 **cout**<<"\n\nBienvenido Al Sistema De Registro De Productos.";

78 **cout**<<"\n(1) Insertar al inicio.";

79 **cout**<<"\n(2) Eliminar al inicio.";

80 **cout**<<"\n(3) Guardar (archivo)";

81 **cout**<<"\n(4) Recuperar (archivo)";

82 **cout**<<"\n(5) Salir.";

83

84 **cout**<<"\n\nIngrese Opcion De Trabajo: ";

85 **cin**>>Opc;

86 **return** Opc;

87 }

88

89 **int** main()

90 {

91 **fstream** ArchivoProductoSalida;

92 ArchivoProductoSalida.open("prod\_electronicos.dat",ios::in | ios::out | ios::binary);

93

94 Lista<Producto> ListaProds;

95 Producto ObjProd;

96 NodoLista<Producto> \*Apunt;

97 **int** Opc, Res, Clave;

98

99

100 Opc= Menu();

101

102 CargarLista(ObjProd,ListaProds,ArchivoProductoSalida);

103

104 **while** (Opc >= 1 && Opc <= 4)

105 {

106 ArchivoProductoSalida.**clear**();

107 **switch** (Opc)

108 {

109 **case** 1:

110 {

111 Escribir(ObjProd,ListaProds,ArchivoProductoSalida);

112 **break**;

113 }

114 **case** 2:

115 {

116 EliminarElemento(ObjProd,ListaProds,ArchivoProductoSalida);

117 }

118 **break**;

119 **case** 3:

120

121 ListaProds.ImprimeIterativo();

122

123

124 **break**;

125 **case** 4:

126 ListaProds.ImprimeIterativo();

127 **break**;

128 }

129 Opc= Menu();

130 }

131 ArchivoProductoSalida.close();

132 **return** 0;

133 }

1 #ifndef PRODUCTOS\_H\_INCLUDED

2 #define PRODUCTOS\_H\_INCLUDED

3 #include <iostream>

4 #include <string.h>

5 **using namespace std**;

6

7 **class** Producto

8 {

9 **private**:

10 **int** Clave;

11 **char** origen[64];

12 **char** destino[64];

13 **double** peso;

14 **public**:

15 Producto();

16 Producto(**int**, **char**[], **char**[], **double**);

17 **int** RegresaClave();

18 **double** RegresaPrecio();

19 **int operator** == (Producto);

20 **int operator** != (Producto);

21 **int operator** > (Producto);

22 **int operator** < (Producto);

23 **friend istream** &**operator**>> (**istream** &, Producto &);

24 **friend** ostream &**operator**<< (ostream &, Producto &);

25 };

26

27 Producto::Producto():Clave(0), origen(""), destino(""), peso(0.0)

28 {}

29

30 Producto::Producto(**int** Cla, **char** NomP[], **char** NomD[], **double** Pre)

31 {

32 Clave= Cla;

33 strcpy(origen, NomP);

34 strcpy(destino, NomD);

35 peso= Pre;

36 }

37

38 **double** Producto::RegresaPrecio()

39 {

40 **return** peso;

41 }

42

43 **int** Producto::RegresaClave()

44 {

45 **return** Clave;

46 }

47

48 **int** Producto::**operator** == (Producto Prod)

49 {

50 **int** Resp=0;

51 **if** (Clave == Prod.Clave)

52 Resp= 1;

53 **return** Resp;

54 }

55

56 **int** Producto::**operator** != (Producto Prod)

57 {

58 **int** Resp=0;

59 **if** (Clave != Prod.Clave)

60 Resp= 1;

61 **return** Resp;

62 }

63

64 **int** Producto::**operator** > (Producto Prod)

65 {

66 **int** Resp=0;

67 **if** (Clave > Prod.Clave)

68 Resp= 1;

69 **return** Resp;

70 }

71

72 **int** Producto::**operator** < (Producto Prod)

73 {

74 **int** Resp=0;

75 **if** (Clave < Prod.Clave)

76 Resp= 1;

77 **return** Resp;

78 }

79

80 **istream** &**operator**>>(**istream** &Lee, Producto &ObjProd)

81 {

82 **cout**<<"\n\nIngrese Clave Del Producto: ";

83 Lee>>ObjProd.Clave;

84 fflush(stdin);

85

86 **cout**<<"\nIngrese Nombre De origen: ";

87 Lee>>ObjProd.origen;

88 fflush(stdin);

89

90 **cout**<<"\nIngrese Nombre De destino: ";

91 Lee>>ObjProd.destino;

92 fflush(stdin);

93

94

95 **cout**<<"\nIngrese peso: ";

96 Lee>>ObjProd.peso;

97 fflush(stdin);

98

99 **return** Lee;

100 }

101

102 ostream &**operator**<< (ostream &Escribe, Producto &ObjProd)

103 {

104 Escribe<<"\n\nDatos Del Producto: \n";

105 Escribe<<"\nClave: "<<ObjProd.Clave;

106 Escribe<<"\nOrigen: "<<ObjProd.origen;

107 Escribe<<"\nDestino: "<<ObjProd.destino;

108 Escribe<<"\nPrecio: "<<ObjProd.peso<<"\n";

109 **return** Escribe;

110 }

111 #endif // PRODUCTOS\_H\_INCLUDED

1 #ifndef LISTASSIMLIG\_H\_INCLUDED

2 #define LISTASSIMLIG\_H\_INCLUDED

3 #include <iostream>

4

5 **using namespace std**;

6

7 **template** <**class** T>

8 **class** Lista;

9

10 **template** <**class** T>

11 **class** NodoLista

12 {

13 **private**:

14 NodoLista<T> \* Liga;

15 T Info;

16 **public**:

17 NodoLista();

18 T RegresaInfo();

19 **friend class** Lista<T>;

20 };

21

22 **template** <**class** T>

23 NodoLista<T>::NodoLista()

24 {

25 Liga= NULL;

26 }

27

28 **template** <**class** T>

29 T NodoLista<T>::RegresaInfo()

30 {

31 **return** Info;

32 }

33

34

35 **template** <**class** T>

36 **class** Lista

37 {

38 **private**:

39 NodoLista<T> \* Primero;

40 **public**:

41 Lista ();

42 NodoLista<T> \* RegresaPrimero();

43 **void** CreaInicio();

44 **void** CreaFinal();

45 **void** ImprimeIterativo();

46 **void** ImprimeRecursivo(NodoLista<T> \* );

47 **void** ImprimeUnNodo(NodoLista<T> \* );

48 **void** InsertaInicio(T);

49 **void** InsertaFinal(T);

50 **void** InsertaOrdenCrec(T);

51 **int** InsertaAntes(T, T);

52 **int** InsertaDespues(T, T);

53 **int** EliminaPrimero();

54 **int** EliminaUltimo();

55 **int** EliminaUnNodo(T);

56 **int** EliminaAnterior(T);

57 **int** EliminaDespues(T);

58 NodoLista<T> \* BuscaDesordenada(T);

59 NodoLista<T> \* BuscaOrdenada(T);

60 NodoLista<T> \* BuscaRecursivo(T, NodoLista<T> \*);

61 };

62

63 **template** <**class** T>

64 Lista<T>::Lista()

65 {

66 Primero= NULL;

67 }

68

69 **template** <**class** T>

70 NodoLista<T> \* Lista<T>::RegresaPrimero()

71 {

72 **return** Primero;

73 }

74

75 **template** <**class** T>

76 **void** Lista<T>::CreaInicio()

77 {

78 NodoLista<T> \* P;

79 T Dato;

80 **char** Resp;

81 Primero= **new** NodoLista<T>();

82 **cout**<< "Ingrese La Informacion A Almacenar: \n";

83 **cin**>>Dato;

84 Primero->Info= Dato;

85 **cout**<< "\nDesea Ingresar Otro Elemento (S/N)?";

86 **cin**>>Resp;

87 **while** (Resp == 'S' || Resp == 's')

88 {

89 **cout**<< "Ingrese La Informacion: \n";

90 **cin**>> Dato;

91 P = **new** NodoLista<T>();

92 P->Info= Dato;

93 P->Liga= Primero;

94 Primero= P;

95 **cout**<< "\nDesea Ingresar Otro Elemento (S/N)? ";

96 **cin**>>Resp;

97 }

98 }

99

100 **template** <**class** T>

101 **void** Lista<T>::CreaFinal()

102 {

103 NodoLista<T> \* P, \*Ultimo;

104 T Dato;

105 **char** Resp;

106 Primero= **new** NodoLista<T>();

107 **cout**<<"Ingrese La Informacion A Almacenar: ";

108 **cin**>>Dato;

109 Primero->Info= Dato;

110

111 Ultimo= Primero;

112 **cout**<<"\nDesea Ingresar Otro Elemento (S/N)? ";

113 **cin**>>Resp;

114 **while** (Resp == 'S' || Resp == 's')

115 {

116 **cout**<< "\nIngrese La Informacion \n";

117 **cin**>>Dato;

118 P= **new** NodoLista<T>();

119 P->Info= Dato;

120 Ultimo->Liga= P;

121 Ultimo= P;

122 **cout**<< "\nDesea Ingresar Otro Elemento (S/N)? ";

123 **cin**>>Resp;

124 }

125 }

126

127

128 **template** <**class** T>

129 **void** Lista<T>::ImprimeIterativo()

130 {

131 NodoLista<T> \* P;

132 P= Primero;

133 **while** (P)

134 {

135 **cout**<< "\nInformacion: "<< P->Info;

136 P= P->Liga;

137 }

138 **cout**<< '\n';

139 }

140

141 **template** <**class** T>

142 **void** Lista<T>::ImprimeRecursivo(NodoLista<T> \* P)

143 {

144 **if** (P)

145 {

146 **cout**<<"\nInformacion: "<< P->Info;

147 ImprimeRecursivo(P->Liga);

148 }

149 **cout**<<'\n';

150 }

151

152 **template** <**class** T>

153 **void** Lista<T>::ImprimeUnNodo(NodoLista<T> \* P)

154 {

155 **if** (P)

156 **cout**<< P->Info;

157 }

158

159 **template** <**class** T>

160 **void** Lista<T>::InsertaInicio(T Dato)

161 {

162 NodoLista<T> \* P;

163 P= **new** NodoLista<T>();

164 P->Info= Dato;

165 P->Liga= Primero;

166 Primero= P;

167 }

168

169 **template** <**class** T>

170 **void** Lista<T>::InsertaFinal(T Dato)

171 {

172 NodoLista<T> \* P, \*Ultimo;

173 P= **new** NodoLista<T>();

174 P->Info= Dato;

175 **if** (Primero)

176 {

177

178 Ultimo= Primero;

179 **while** (Ultimo->Liga)

180 Ultimo= Ultimo->Liga;

181

182 Ultimo->Liga= P;

183 }

184 **else**

185 Primero= P;

186 }

187

188

189 **template** <**class** T>

190 **void** Lista<T>::InsertaOrdenCrec(T Dato)

191 {

192 NodoLista<T> \* P, \*Q, \*Ant;

193 **if**(!Primero || Primero->Info > Dato)

194 InsertaInicio(Dato);

195 **else**

196 {

197 Q= Primero;

198 **while** (Q && Q->Info < Dato)

199 {

200 Ant= Q;

201 Q= Q->Liga;

202 }

203 P= **new** NodoLista<T>();

204 P->Info= Dato;

205 Ant->Liga= P;

206 P->Liga= Q;

207 }

208 }

209

210 **template** <**class** T>

211 **int** Lista<T>::InsertaAntes(T Dato, T Ref)

212 {

213 NodoLista<T> \* P, \*Ant, \*Q;

214 **int** Resp= 1;

215 **if** (Primero)

216 {

217 Q= Primero;

218 **while** ((Q != NULL ) && (Q->Info != Ref))

219 {

220 Ant= Q;

221 Q= Q->Liga;

222 }

223 **if** (Q != NULL )

224 {

225 P= **new** NodoLista<T>();

226 P->Info= Dato;

227

228 **if** (Primero == Q)

229 {

230 P->Liga= Primero;

231 Primero= P;

232 }

233 **else**

234 {

235 Ant->Liga= P;

236 P->Liga= Q;

237 }

238 }

239 **else**

240

241 Resp= 0;

242 }

243 **else**

244 Resp= -1;

245 **return** Resp;

246 }

247

248 **template** <**class** T>

249 **int** Lista<T>::InsertaDespues(T Dato, T Ref)

250 {

251 NodoLista<T> \* Q, \*P;

252 **int** Resp= 1;

253 **if** (Primero)

254 {

255 Q = Primero;

256 **while** ((Q != NULL ) && (Q->Info != Ref))

257 Q= Q->Liga;

258 **if** ( Q != NULL )

259 {

260 P= **new** NodoLista<T>();

261 P->Info= Dato;

262 P->Liga= Q->Liga;

263 Q->Liga= P;

264 }

265 **else**

266 Resp= 0;

267 }

268 **else**

269 Resp= -1;

270 **return** Resp;

271 }

272

273 **template** <**class** T>

274 **int** Lista<T>::EliminaPrimero()

275

276 {

277 NodoLista<T> \* P;

278 **int** Resp= 1;

279 **if** (Primero)

280 {

281 P= Primero;

282 Primero= P->Liga;

283 **delete** (P);

284 }

285 **else**

286 Resp= 0;

287 **return** Resp;

288 }

289

290 **template** <**class** T>

291 **int** Lista<T>::EliminaUltimo()

292 {

293 NodoLista<T> \* Ant, \*P;

294 **int** Resp= 1;

295 **if** (Primero)

296 {

297 **if** (!Primero->Liga)

298 {

299 **delete** (Primero);

300 Primero= NULL;

301 }

302 **else**

303 {

304 P= Primero;

305 **while** (P->Liga)

306 {

307 Ant= P;

308 P= P->Liga;

309 }

310 Ant->Liga= NULL;

311 **delete** (P);

312 }

313 }

314 **else**

315

316 Resp= 0;

317 **return** Resp;

318 }

319

320 **template** <**class** T>

321 **int** Lista<T>::EliminaUnNodo(T Ref)

322 {

323 NodoLista<T> \* P, \*Ant;

324 **int** Resp= 1;

325 **if** (Primero)

326 {

327 P= Primero;

328 **while** ((P->Liga) && (P->Info != Ref))

329 {

330 Ant= P;

331 P= P->Liga;

332 }

333 **if** (P->Info != Ref)

334

335 Resp= 0;

336 **else**

337 {

338 **if** (Primero == P)

339 Primero= P->Liga;

340 **else**

341 Ant->Liga= P->Liga;

342 **delete** (P);

343 }

344 }

345 **else**

346 Resp= -1;

347 **return** Resp;

348 }

349

350 **template** <**class** T>

351 **int** Lista<T>::EliminaAnterior(T Ref)

352 {

353 NodoLista<T> \* Q, \*Ant, \*P;

354 **int** Resp= 1;

355 **if** (Primero)

356 {

357 **if** (Primero->Info == Ref)

358 Resp= 2;

359 **else**

360 {

361 Q= Primero;

362 Ant= Primero;

363 **while** ((Q->Info != Ref) && (Q->Liga))

364 {

365 P= Ant;

366 Ant= Q;

367 Q= Q->Liga;

368 }

369

370 **if** (Q->Info != Ref)

371 Resp= 3;

372 **else if** (Primero->Liga == Q)

373 {

374 **delete** (Primero);

375 Primero= Q;

376 }

377 **else**

378 {

379 P->Liga= Q;

380 **delete** (Ant);

381 }

382 }

383 }

384 **else**

385

386 Resp= 4;

387 **return** Resp;

388 }

389

390 **template** <**class** T>

391 NodoLista<T> \* Lista<T>::BuscaDesordenada(T Ref)

392 {

393 NodoLista<T> \* Q, \*Resp= NULL;

394 **if** (Primero)

395 {

396 Q= Primero;

397 **while** ((Q->Info != Ref) && (Q->Liga))

398 Q= Q->Liga;

399

400 **if** (Q->Info == Ref)

401 Resp= Q;

402 }

403 **return** Resp;

404 }

405

406 **template** <**class** T>

407 NodoLista<T> \* Lista<T>::BuscaOrdenada(T Ref)

408 {

409 NodoLista<T> \* Q, \*Resp= NULL;

410 **if** (Primero)

411 {

412 Q= Primero;

413 **while** ((Q->Info < Ref) && (Q->Liga))

414 Q= Q->Liga;

415

416 **if** (Q->Info == Ref)

417 Resp= Q;

418 }

419 **return** Resp;

420 }

421

422 **template** <**class** T>

423 NodoLista<T> \* Lista<T>::BuscaRecursivo(T Dato, NodoLista<T> \* Q)

424 {

425 **if** (Q)

426 **if** (Q->Info == Dato)

427 **return** Q;

428 **else**

429 **return** BuscaRecursivo(Dato, Q->Liga);

430 **else**

431 **return** NULL;

432 }

433 #endif // LISTASSIMLIG\_H\_INCLUDED