16-3-2022

Practica 4 Lista simplemente dinámica

Materia: Seminario de estructura de datos 1

Sección: D13.

Código: 216584703

Carrera: Ingeniería en computación.

Nombre alumno: Padilla Pérez Jorge Daray

Nombre profesor: Julio Esteban Valdes Lopez



**Introducción**

En esta practica se realizó la implementación de una Lista simplemente ligada dinámica sin encabezado utilizando una estructura llamada dirección en la que se almacena la dirección de una persona, en donde se utiliza una clase llamada lista la cual hace los métodos de insertar al inicio en posición anular la lista etc.

Además de que en esta lista se utilizo herencia para las posibles listas nuevas que se dejen en lo que resta del curso, simplemente heredamos la estructura dirección para pasarla por herencia a una posible lista nueva que se pida y tener ahí todas las listas en un solo programa.

Texto

Descripción generada automáticamente

Aquí se aprecia el menú.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Al querer mostrar la lista dice que esta vacia.

Texto

Descripción generada automáticamente

Si queremos insertar en una posición invalida no nos deja, ya que tiene que ser lineal y en este caso debería empezar en la posición 0.

Texto

Descripción generada automáticamente

Aquí insertamos un nodo en la posición 0.

Texto

Descripción generada automáticamente

Aquí insertamos otro nodo en la posición 0, ósea al principio de la lista.

Texto

Descripción generada automáticamente

Aquí mostramos la lista.

Texto

Descripción generada automáticamente

Insertamos un tercer nodo en la ultima posición.

Texto

Descripción generada automáticamente

Insertamos un nodo 4 en la posición 1 por lo que recorre a los demás.

Texto

Descripción generada automáticamente

Aquí usamos la función recupera que en este caso pide el nombre del nodo.

Texto

Descripción generada automáticamente

Aquí usamos la función localiza que en este caso pide la posicion del nodo.

Texto

Descripción generada automáticamente

Aquí eliminamos el pin 2.

Texto

Descripción generada automáticamente

Aquí anulamos la lista.

**Conclusión**

Se pudo completar de manera correcta el programa, utilizando una estructura dirección, en la que se almacenan los datos de la persona, seguido de una herencia a una clase llamada lista simple, en la cual pues se tienen los métodos principales de una lista simple como la que se pidió.

También pues cabe resaltar que me gustan más las listas dinámicas y hasta ahorita se me han hecho más fáciles de programar las dinámicas que con registros, aunque sigo ansioso por si nos toca hacer los métodos de ordenamiento con listas dinámicas, ya que pensándolo siento que será mas difícil de implementar que en una estática.

**Codigo fuente**

1 #include <iostream>

2 #include <string.h>

3 *using namespace std*;

4

5 *class* StructBase

6 {

7 *protected*:

8 *typedef struct* Address

9 {

10 *char* name[50];

11 *char* street[100];

12 *char* city[50];

13 *char* state[20];

14 *int* pin;

15 *struct* Address \*next;

16 }Address;

17 Address \* first;

18 Address \* last;

19

20 *void* inicializa()

21 {

22 first = *nullptr*;

23 last = *nullptr*;

24 }

25

26 *void* show()

27 {

28 Address \*temp = (Address \*) malloc(*sizeof*(Address));

29 temp = first;

30 *if* (first != NULL){

31 *int* i = 0;

32 *while* (temp != NULL){

33 printf("Actual %d", i);

34 printf("\n Name: %s\n", temp->name);

35 printf("street: %s\n", temp->street);

36 printf("City: %s\n", temp->city);

37 printf("State: %s\n", temp->state);

38 printf("pin: %i\n\n", temp->pin);

39 temp = temp ->next;

40 i++;

41 }

42 }

43 *else*{

44 printf("\n The list is empty\n");

45 }

46 }

47

48 *void* seekNode()

49 {

50 Address \*temp = (Address \*) malloc(*sizeof*(Address));

51 temp = first;

52 *int* found = 0 ;

53 *char* cadena[50];

54 printf(" Introduce the name of the node that you want to seek: ");

55 scanf("%s", &cadena);

56 *if* (first != NULL){

57 *int* i=0;

58 *while* (temp != NULL && found != 1){

59 *if* ( strcmp(temp->name, cadena)==0 ){

60 printf("Actual %d", i);

61 printf("\n Name: %s\n", temp->name);

62 printf("street: %s\n", temp->street);

63 printf("City: %s\n", temp->city);

64 printf("State: %s\n", temp->state);

65 printf("pin: %i\n\n", temp->pin);

66 found = 1;

67 }

68 temp = temp ->next;

69 i++;

70 }

71 *if* (found == 0){

72 printf("El nodo no fue encontrado");

73 }

74 }

75 *else*{

76 printf("\n The list is empty\n");

77 }

78 }

79

80 *void* Localiza()

81 {

82 Address \*temp = (Address \*) malloc(*sizeof*(Address));

83 temp = first;

84 *int* found = 0 ;

85 *int* pos;

86 printf(" Introduce the pos of the node that you want to locate: ");

87 scanf("%d", &pos);

88 *if* (first != NULL){

89 *int* i=0;

90 *while* (temp != NULL && found != 1){

91 *if* ( i == pos ){

92 printf("Actual %d", i++);

93 printf("\n Name: %s\n", temp->name);

94 printf("street: %s\n", temp->street);

95 printf("City: %s\n", temp->city);

96 printf("State: %s\n", temp->state);

97 printf("pin: %i\n\n", temp->pin);

98 found = 1;

99 }

100 temp = temp ->next;

101 i++;

102 }

103 *if* (found == 0){

104 printf("El nodo no fue encontrado");

105 }

106 }

107 *else*{

108 printf("\n The list is empty\n");

109 }

110 }

111 *int* ultimo = 0;

112 *int* tam()

113 {

114 *return* ultimo;

115 }

116

117 *void* eliminateNode()

118 {

119 Address \*actual = (Address \*) malloc(*sizeof*(Address));

120 actual = first;

121

122 Address\* before = (Address \*) malloc(*sizeof*(Address));

123 before = NULL;

124

125 *int* soughtnode = 0, found = 0;

126

127 printf(" Introduce the pin of the node that you want to eliminate: ");

128 scanf("%d", &soughtnode);

129

130 *if*(first != NULL){

131 *while*(actual != NULL && found != 1){

132

133 *if*(actual -> pin == soughtnode){

134

135 *if*(actual == first){

136 first = first ->next;

137 }

138 *else*{

139 before -> next = actual -> next;

140 }

141 printf("\nThe link of the node have been eliminated");

142 found = 1;

143 }

144 before = actual;

145 actual = actual ->next;

146 }

147 *if*(found == 0){

148 printf("\nthe node was not found\n\n");

149 }

150 *else*{

151 free(before);

152 ultimo--;

153 printf("\n\nNODE ELIMINATED SUCCESSFUL");

154 }

155 }

156 *else*{

157 printf("\nTHE LIST IS EMPTY\n\n");

158 }

159 printf("\n");

160 }

161

162 };

163

164 *class* List: StructBase

165 {

166 *protected*:

167 *void* inserfirst()

168 {

169 Address \*in\_first = (Address \*) malloc(*sizeof*(Address));

170 *if*(!in\_first){*// new\_==NULL*

171 printf("Memory allocation error, new node could not be created");

172 *return*;

173 }

174

175 printf("INTRODUCE THE VALUE OF THE NEW NODE: \n");

176 printf("Give the name ");

177 scanf("%s", &in\_first->name);

178 printf("Give the street ");

179 scanf("%s", &in\_first->street);

180 printf("Give the city ");

181 scanf("%s", &in\_first->city);

182 printf("Estado: ");

183 scanf("%s", &in\_first->state);

184 printf("pin: ");

185 scanf("%i",&in\_first->pin);

186 *if*(first == NULL){

187 first = in\_first;

188 first ->next = NULL;

189 last = in\_first;

190 }

191 *else*{

192 in\_first->next = first;

193 first=in\_first;

194 }

195 ultimo++;

196 printf("\nTHE NODE HAVE BEEN INTRODUCE CORRECTLY\n\n");

197 system("pause");

198 }

199

200 *void* insertlast()

201 {

202 Address \*in\_last = (Address \*) malloc(*sizeof*(Address));

203 *if*(!in\_last){*// new\_==NULL*

204 printf("Memory allocation error, new node could not be created");

205 *return*;

206 }

207

208 printf("INTRODUCE THE VALUE OF THE NEW NODE: \n");

209 printf("Give the name ");

210 scanf("%s", &in\_last->name);

211 printf("Give the street ");

212 scanf("%s", &in\_last->street);

213 printf("Give the city ");

214 scanf("%s", &in\_last->city);

215 printf("Estado: ");

216 scanf("%s", &in\_last->state);

217 printf("pin: ");

218 scanf("%i",&in\_last->pin);

219 *if*(last == NULL){

220 first = in\_last;

221 first ->next = NULL;

222 last = in\_last;

223 }

224 *else*{

225 last -> next = in\_last;

226 in\_last -> next = NULL;

227 last = in\_last;

228 }

229 ultimo++;

230 printf("\nTHE NODE HAVE BEEN INTRODUCE CORRECTLY\n\n");

231 system("pause");

232 }

233

234 *void* Anula()

235 {

236 Address \*actual = (Address \*) malloc(*sizeof*(Address));

237 actual = first;

238

239 Address\* before = (Address \*) malloc(*sizeof*(Address));

240 before = NULL;

241

242 *if*(first != NULL)

243 {

244 *while*(actual != NULL)

245 {

246

247 *if*(actual == first)

248 {

249 first = first ->next;

250 }

251 *else*

252 {

253 before -> next = actual -> next;

254 }

255 before = actual;

256 actual = actual ->next;

257 }

258 free(before);

259 ultimo--;

260 printf("\n\nLIST ELIMINATED SUCCESSFUL");

261 }

262 *else*

263 {

264 printf("\nTHE LIST IS EMPTY\n\n");

265 }

266 printf("\n");

267 inicializa();

268 }

269

270 *public*:

271 *void* insertNode(*int* pos)

272 {

273 Address \*aux = *nullptr*;

274 aux = *new* Address;

275 *if* (pos == 0)

276 {

277 inserfirst();

278 }

279 *else if* (pos == tam())

280 {

281 insertlast();

282 }

283 *else if*(pos >= 1 && pos < tam())

284 {

285 printf("Give the name ");

286 scanf("%s", &aux->name);

287 printf("Give the street ");

288 scanf("%s", &aux->street);

289 printf("Give the city ");

290 scanf("%s", &aux->city);

291 printf("Estado: ");

292 scanf("%s", &aux->state);

293 printf("pin: ");

294 scanf("%i",&aux->pin);

295 Address \*aux2, \*aux3;

296 aux2 = first;

297 *for* (*int* i=0; i<pos; i++)

298 {

299 aux3 = aux2;

300 aux2 = aux2->next;

301 }

302 aux3->next = aux;

303 aux->next = aux2;

304 ultimo++;

305 printf("\nTHE NODE HAVE BEEN INTRODUCE CORRECTLY\n\n");

306 system("pause");

307 }

308 }

309 *void* getshow(){

310 *return* show();

311 }

312 *void* getseeknode(){

313 *return* seekNode();

314 }

315 *void* geteliminatenode(){

316 *return* eliminateNode();

317 }

318 *void* getinsertfirst(){

319 *return* inserfirst();

320 }

321 *void* getinsertlast(){

322 *return* insertlast();

323 }

324 *void* getLocaliza(){

325 *return* Localiza();

326 }

327 *void* getAnula(){

328 *return* Anula();

329 }

330 *void* getinicializa(){

331 *return* inicializa();

332 }

333

334 };

335

336 *void* menu()

337 {

338 *int* opc;

339 List mi\_lista;

340 *int* pos;

341 mi\_lista.getinicializa();

342

343 *do*{

344 printf("\tINTRODUCE AN OPCION:\n");

345 printf("0.-Exit\n1.-Single list\n");

346 scanf("%d", &opc);

347

348

349 *switch*(opc){

350 *case* 0: printf("\tGoodbye");

351 system("pause");

352 *break*;

353 *case* 1: printf("\tSingle list\n\n");

354 system("cls");

355 *int* opcion;

356 *do*{

357 printf("\tINTRODUCE AN OPCION:\n");

358 printf("0.-Exit\n1.-Show\n2.-Insert\n3.-Seek\n4.-Delete\n5.-Insert first\n6.-Insert last\n7.-Localiza\n8.-Anula\n");

359 scanf("%d", &opcion);

360 *switch*(opcion){

361 *case* 0: printf("\t Adios\n");

362 system("pause");

363 *break*;

364 *case* 1: printf("\tShow the list\n\n");

365 system("cls");

366 mi\_lista.getshow();

367 system("pause");

368 *break*;

369 *case* 2: system("cls");

370 printf("\tInsert a node in the list\n\n");

371 printf("Posicion a insertar:\n");

372 scanf("%d", &pos);

373 mi\_lista.insertNode(pos);

374 *break*;

375 *case* 3: system("cls");

376 printf("\tSeek a node in the list\n\n");

377 mi\_lista.getseeknode();

378 *break*;

379 *case* 4: system("cls");

380 printf("\tDelete node\n\n");

381 mi\_lista.geteliminatenode();

382 *break*;

383 *case* 5: system("cls");

384 printf("\tInsert first\n\n");

385 mi\_lista.getinsertfirst();

386 *break*;

387 *case* 6: system("cls");

388 printf("\tInsert first\n\n");

389 mi\_lista.getinsertlast();

390 *break*;

391 *case* 7: system("cls");

392 printf("\tInsert first\n\n");

393 mi\_lista.getLocaliza();

394 *break*;

395 *case* 8: system("cls");

396 printf("\tInsert first\n\n");

397 mi\_lista.getAnula();

398 *break*;

399 }

400 }*while*(opcion!=0);

401 *break*;

402 *default*: printf("\tChoose a correct value");

403 *break*;

404 }

405 system("cls");

406 }*while*(opc!=0);

407

408 }

409

410 *int* main()

411 {

412 menu();

413 *return* 0;

414 }