|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **ИУК «Информатика и управление»** |
| **КАФЕДРА** | **ИУК4 «Программное обеспечение ЭВМ,** |
| **информационные технологии»** | |

**ЛАБОРАТОРНАЯ РАБОТА №1**

**«Работа со списочными структурами»**

**ДИСЦИПЛИНА: «Типы и структуры данных»**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил: студент гр. ИУК4-31Б | |  |  | ( | Суриков Н. С. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |
| Проверил: | |  |  | ( | Былинка М. И. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |

|  |  |
| --- | --- |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | |
|  | - Балльная оценка:  - Оценка: |

**Цель:** формирование практических навыков создания алгоритмов обработки списочных структур данных.

**Задачи:**

1. Изучить основные виды списочных структур.
2. Изучить организацию списочных структур.
3. Познакомиться с основными операциями для обработки списков.
4. Изучить типовые алгоритмы решения задач с использованием списков.
5. Реализовать основные алгоритмы обработки списочных структур данных (создание, удаление, поиск, добавление и удаление элемента), а также алгоритм согласно полученному варианту.

**Вариант 25**

**Формулировка задания:**

1. Разработать консольное приложение, написанное с помощью объектно-ориентированной технологии. Индивидуальное задание предусмотрено вариантом, который назначает преподаватель.

2. Приложение необходимо запускать для демонстрации из командной строки с указанием названий приложения и трех файлов:

− все входные данные (например, последовательности чисел, коэффициенты многочленов и т.д.) считать из первого файла;

− все выходные данные записать во второй файл;

− все возникшие ошибки записать в третий файл – файл ошибок.

3. Все основные сущности приложения представить в виде отдельных классов.

4. Необходимо предусмотреть пользовательское меню, содержащее набор команд всех основных операций для работы со списком, а также команду для запуска индивидуального задания.

5. В приложении также должны быть учтены все критические ситуации, обработанные с помощью класса исключений.

**Индивидуальное задание:**

Дана последовательность латинских букв, оканчивающаяся точкой. Среди букв есть специальный символ, появление которого означает отмену предыдущей буквы; k знаков подряд отменяют k предыдущих букв, если такие есть. Учитывая вхождение этого символа преобразовать последовательность (для решения задачи использовать линейный двусвязный динамический список).

**Листинг программы:**

**main.cpp**

1 *#include* *"CMenu.h"*

2 *#include* *"CMenuItem.h"*

3 *#include* *"DoubleList.h"*

4 *#include* *"FileLogging.h"*

5 *#include* *"MyError.h"*

6 *#include* *<climits>*

7 *#include* *<sstream>*

8 *#include* *<string>*

9 *#include* *<vector>*

10

11 enum Points

12 {

13 CREATE\_LIST = 1,

14 PRINT\_LIST,

15 INSERT\_IN\_LIST,

16 DELETE\_FROM\_LIST,

17 CLEAR\_LIST,

18 FIND\_ELEMENT,

19 CHECK\_EMPTY,

20 GET\_LENGTH,

21 INDIVIDUAL\_TASK,

22 DELETE\_LIST,

23 OPEN\_ERROR\_LOG,

24 OPEN\_OUTPUT\_LOG,

25 INPUT\_DATA\_FROM\_FILE,

26 EXIT

27 };

28

29 bool IsDigit(char c);

30 int Input(std::string message, int min, int max, FileLogging &f1);

31 void InputDataFromFile(DoubleList \*dl, const std::string &input);

32 void PrintList(DoubleList \*dl, FileLogging &fl);

33 void InsertInList(DoubleList \*dl, FileLogging &fl);

34 void DeleteFromList(DoubleList \*dl, FileLogging &fl);

35 void ClearList(DoubleList \*dl, FileLogging &fl);

36 void FindElement(DoubleList \*dl, FileLogging &fl);

37 void CheckEmpty(DoubleList \*dl, FileLogging &fl);

38 void GetLength(DoubleList \*dl, FileLogging &fl);

39 void IndividualTask(FileLogging &fl);

40 void WaitForEnter();

41

42 void run(CMenu menu)

43 {

44 try

45 {

46 while (menu.runCommand())

47 {

48 }

49 }

50 catch (const MyError &exception)

51 {

52 std::cout << "Error: " << exception.getError() << std::endl;

53 WaitForEnter();

54 std::cout << "\033[2J\033[1;1H";

55 run(menu);

56 }

57 }

58

59 std::string MyError::m\_file = std::string();

60

61 int main(int argc, char \*argv[])

62 {

63 DoubleList \*dl = nullptr;

64 std::string error = "error\_log.txt";

65 std::string output = "output\_log.txt";

66 std::string input = "input.txt";

67

68 if (argc >= 3)

69 {

70 input = argv[1];

71 output = argv[2];

72 error = argv[3];

73 }

74

75 FileLogging errorLog(error);

76 FileLogging outputLog(output);

77 outputLog.Logging("Program is launched.\n");

78

79 CMenuItem menuItems[] = {

80 CMenuItem("Создать список", [&]()

81 {

82 std::cout << "\033[2J\033[1;1H";

83 if (!dl)

84 {

85 dl = new DoubleList();

86 std::cout << "Список успешно создан!\n";

87 }

88 else

89 {

90 std::cout << "Список уже создан!\n";

91 errorLog.Logging("List already created: attempt to create a list.\n");

92 }

93 WaitForEnter();

94 return CREATE\_LIST; }),

95 CMenuItem("Вывести список", [&]()

96 {

97 std::cout << "\033[2J\033[1;1H";

98 PrintList(dl, errorLog);

99 WaitForEnter();

100 return PRINT\_LIST; }),

101 CMenuItem("Вставить элемент в список", [&]()

102 {

103 std::cout << "\033[2J\033[1;1H";

104 InsertInList(dl, errorLog);

105 WaitForEnter();

106 return INSERT\_IN\_LIST; }),

107 CMenuItem("Удалить элемент из списка", [&]()

108 {

109 std::cout << "\033[2J\033[1;1H";

110 DeleteFromList(dl, errorLog);

111 WaitForEnter();

112 return DELETE\_FROM\_LIST; }),

113 CMenuItem("Очистить список", [&]()

114 {

115 std::cout << "\033[2J\033[1;1H";

116 ClearList(dl, errorLog);

117 WaitForEnter();

118 return CLEAR\_LIST; }),

119 CMenuItem("Найти элемент в списке", [&]()

120 {

121 std::cout << "\033[2J\033[1;1H";

122 FindElement(dl, errorLog);

123 WaitForEnter();

124 return FIND\_ELEMENT; }),

125 CMenuItem("Проверить список на пустоту", [&]()

126 {

127 std::cout << "\033[2J\033[1;1H";

128 CheckEmpty(dl, errorLog);

129 WaitForEnter();

130 return CHECK\_EMPTY; }),

131 CMenuItem("Узнать длину списка", [&]()

132 {

133 std::cout << "\033[2J\033[1;1H";

134 GetLength(dl, errorLog);

135 WaitForEnter();

136 return GET\_LENGTH; }),

137 CMenuItem("Индивидуальное задание", [&]()

138 {

139 std::cout << "\033[2J\033[1;1H";

140 IndividualTask(errorLog);

141 WaitForEnter();

142 return INDIVIDUAL\_TASK; }),

143 CMenuItem("Удалить список", [&]()

144 {

145 std::cout << "\033[2J\033[1;1H";

146 if (!dl)

147 {

148 std::cout << "Список еще не создан!\n";

149 errorLog.Logging("List does not exist: attempt to delete list.\n");

150 }

151 else

152 {

153 delete dl;

154 dl = nullptr;

155 std::cout << "Список успешно удален!\n";

156 }

157 WaitForEnter();

158 return DELETE\_LIST; }),

159 CMenuItem("Открыть error\_log.txt", [&]()

160 {

161 std::cout << "\033[2J\033[1;1H";

162 errorLog.PrintFile();

163 WaitForEnter();

164 return OPEN\_ERROR\_LOG; }),

165 CMenuItem("Открыть output\_log.txt", [&]()

166 {

167 std::cout << "\033[2J\033[1;1H";

168 outputLog.PrintFile();

169 WaitForEnter();

170 return OPEN\_OUTPUT\_LOG; }),

171 CMenuItem("Считать данные из input.txt", [&]()

172 {

173 std::cout << "\033[2J\033[1;1H";

174 if (!dl)

175 {

176 dl = new DoubleList();

177 }

178 InputDataFromFile(dl, input);

179 WaitForEnter();

180 return INPUT\_DATA\_FROM\_FILE; }),

181 CMenuItem("Выход", [&]()

182 { return 0; })};

183

184 CMenu menu("Меню", menuItems, sizeof(menuItems) / sizeof(CMenuItem));

185 run(menu);

186

187 if (dl)

188 {

189 delete dl;

190 }

191 return 0;

192 }

193

194 bool IsDigit(char c)

195 {

196 return '0' <= c && c <= '9';

197 }

198

199 int Input(std::string message, int min, int max, FileLogging &f1)

200 {

201 int number = 0;

202 bool correct = false;

203 while (!correct)

204 {

205 std::cout << message;

206 std::string input;

207 std::cin >> input;

208 correct = (input[0] == '-' || IsDigit(input[0]));

209

210 for (size\_t i = 1; i < input.size(); i++)

211 {

212 if (!IsDigit(input[i]))

213 {

214 correct = false;

215 break;

216 }

217 }

218

219 if (!correct)

220 {

221 std::cout << "Некорректная запись числа!\n";

222 f1.Logging("Incorrect number entry.\n");

223 continue;

224 }

225

226 if (input.size() > std::to\_string(INT\_MAX).size() - 1)

227 {

228 correct = false;

229 std::cout << "Введенное число выходит из допустимого диапазона!\n";

230 f1.Logging("The entered number out of range.\n");

231 continue;

232 }

233

234 number = stoi(input);

235 if (min > number || max < number)

236 {

237 correct = false;

238 std::cout << "Введенное число выходит из допустимого диапазона!\n";

239 f1.Logging("The entered number out of range.\n");

240 }

241 }

242 return number;

243 }

244

245 void InputDataFromFile(DoubleList \*dl, const std::string &input)

246 {

247 dl->Clear();

248 std::ifstream fin(input);

249 if (fin.is\_open())

250 {

251 int data;

252 while (fin >> data)

253 {

254 dl->PushBack(data);

255 }

256 std::cout << "Данные успешно считались!\n";

257 }

258 else

259 {

260 std::cout << "Не удалось открыть файл: " << input << "\n";

261 }

262 }

263

264 void PrintList(DoubleList \*dl, FileLogging &fl)

265 {

266 if (dl)

267 {

268 dl->PrintDoubleList();

269 }

270 else

271 {

272 std::cout << "Список еще не создан!\n";

273 fl.Logging("List does not exist: attempt to print list.\n");

274 }

275 }

276

277 void WaitForEnter()

278 {

279 std::cout << "Нажмите Enter, чтобы продолжить...";

280 std::cin.clear();

281 std::cin.ignore(1024, '\n');

282 std::cin.get();

283 }

284

285 void InsertInList(DoubleList \*dl, FileLogging &fl)

286 {

287 if (!dl)

288 {

289 std::cout << "Список еще не создан!\n";

290 fl.Logging("List does not exist: attempt to add a new element to the list.\n");

291 return;

292 }

293

294 std::cout << "--------------------------Вставка---------------------------\n"

295 << " 1. Вставить элемент в начало списка\n"

296 << " 2. Вставить элемент в конец списка\n"

297 << " 3. Вставить элемент в список по индексу\n"

298 << " 4. Выйти в главное меню\n"

299 << "-------------------------------------------------------------\n";

300

301 int subchoice = Input("Выбрать: ", 1, 4, fl);

302 int data = 0;

303 int index = 0;

304

305 switch (subchoice)

306 {

307 case 1:

308 data = Input("Введите число для вставки: ", INT\_MIN, INT\_MAX, fl);

309 dl->PushFront(data);

310 std::cout << "Число " << data << " успешно добавлено в начало списка!\n";

311 break;

312

313 case 2:

314 data = Input("Введите число для вставки: ", INT\_MIN, INT\_MAX, fl);

315 dl->PushBack(data);

316 std::cout << "Число " << data << " успешно добавлено в конец списка!\n";

317 break;

318

319 case 3:

320 data = Input("Введите число для вставки: ", INT\_MIN, INT\_MAX, fl);

321 index = Input("Введите индекс элемента для вставки: ", 0, dl->getSize(), fl);

322 dl->Insert(index, data);

323 std::cout << "Число " << data << " успешно добавлено в позицию с номером " << index << " списка!\n";

324 break;

325

326 case 4:

327 break;

328 }

329 }

330

331 void DeleteFromList(DoubleList \*dl, FileLogging &fl)

332 {

333 if (!dl)

334 {

335 std::cout << "Список еще не создан!\n";

336 fl.Logging("List does not exist: attempt to remove an element from a list.\n");

337 return;

338 }

339

340 if (dl->IsEmpty())

341 {

342 std::cout << "Нельзя ничего удалить из пустого списка!\n";

343 fl.Logging("List is empty: attempt to remove an element from a list.\n");

344 return;

345 }

346

347 std::cout << "--------------------------Удаление---------------------------\n"

348 << " 1. Удалить первый элемент списка\n"

349 << " 2. Удалить последний элемент списка\n"

350 << " 3. Удалить элемент из списка по индексу\n"

351 << " 4. Выйти в главное меню\n"

352 << "-------------------------------------------------------------\n";

353

354 int subchoice = Input("Выбрать: ", 1, 4, fl);

355

356 switch (subchoice)

357 {

358 case 1:

359 dl->PopFront();

360 std::cout << "Первый элемент успешно удалён!\n";

361 break;

362

363 case 2:

364 dl->PopBack();

365 std::cout << "Последний элемент успешно удалён!\n";

366 break;

367

368 case 3:

369 int index = Input("Введите индекс элемента для удаления: ", 0, dl->getSize() - 1, fl);

370 dl->Remove(index);

371 std::cout << "Элемент с индексом " << index << " успешно удалён из списка!\n";

372 break;

373 }

374 }

375

376 void ClearList(DoubleList \*dl, FileLogging &fl)

377 {

378 if (dl)

379 {

380 dl->Clear();

381 std::cout << "Список успешно очищен!\n";

382 }

383 else

384 {

385 std::cout << "Список еще не создан!\n";

386 fl.Logging("List does not exist: trying to clear the list.\n");

387 }

388 }

389

390 void FindElement(DoubleList \*dl, FileLogging &fl)

391 {

392 if (!dl)

393 {

394 std::cout << "Список еще не создан!\n";

395 fl.Logging("List does not exist: attempt to find an element in a list.\n");

396 return;

397 }

398

399 std::cout << "--------------------------Поиск------------------------------\n"

400 << " 1. Первое вхождение элемента слева\n"

401 << " 2. Первое вхождение элемента справа\n"

402 << " 3. Выйти в главное меню\n"

403 << "-------------------------------------------------------------\n";

404

405 int subchoice = Input("Выбрать: ", 1, 3, fl);

406 int data = 0;

407

408 switch (subchoice)

409 {

410 case 1:

411 data = Input("Введите число для поиска: ", INT\_MIN, INT\_MAX, fl);

412 std::cout << "Индекс этого элемента: " << dl->FindElement(data) << "\n";

413 break;

414

415 case 2:

416 data = Input("Введите число для поиска: ", INT\_MIN, INT\_MAX, fl);

417 std::cout << "Индекс этого элемента: " << dl->RFindElement(data) << "\n";

418 break;

419

420 case 3:

421 break;

422 }

423 }

424

425 void CheckEmpty(DoubleList \*dl, FileLogging &fl)

426 {

427 if (dl)

428 {

429 std::cout << "Список пуст: " << (dl->IsEmpty() ? "да" : "нет") << "\n";

430 }

431 else

432 {

433 std::cout << "Список еще не создан!\n";

434 fl.Logging("List does not exist: attempt to check if the list is empty.\n");

435 }

436 }

437

438 void GetLength(DoubleList \*dl, FileLogging &fl)

439 {

440 if (dl)

441 {

442 std::cout << "Длина списка: " << dl->getSize() << "\n";

443 }

444 else

445 {

446 std::cout << "Список еще не создан!\n";

447 fl.Logging("List does not exist: attempt to get the length of the list.\n");

448 }

449 }

450

451 void IndividualTask(FileLogging &fl)

452 {

453 std::string input;

454 DoubleList \*list = new DoubleList();

455

456 while (true)

457 {

458 std::cout << "Введите последовательность латинских букв, оканчивающуюся точкой: ";

459 std::cin >> input;

460

461 bool validInput = true;

462 for (char c : input)

463 {

464 if (!((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || c == '#' || c == '.'))

465 {

466 validInput = false;

467 break;

468 }

469 }

470

471 if (!validInput)

472 {

473 std::cout << "Ошибка: ввод должен содержать только латинские буквы, '#' и '.'\n";

474 fl.Logging("The input must contain only Latin letters, '#' and '.'\n");

475 continue;

476 }

477

478 for (char c : input)

479 {

480 if (c == '.')

481 {

482 break;

483 }

484 else if (c == '#')

485 {

486 list->PopBack();

487 }

488 else

489 {

490 list->PushBack(c);

491 }

492 }

493 DoubleList::Node \*curr = list->getHead();

494 while (curr)

495 {

496 std::cout << char(curr->data);

497 curr = curr->next;

498 }

499 std::cout << std::endl;

500 break;

501 }

502 }

**Cmenu.cpp**

1 *#include* *"CMenu.h"*

2 *#include* *"MyError.h"*

3 *#include* *<iostream>*

4

5 CMenu::CMenu() {}

6 CMenu::CMenu(std::string title, CMenuItem \*items, size\_t count) : m\_title(title), m\_items(items), m\_count(count) {}

7

8 int CMenu::getSelect() const

9 {

10 return m\_select;

11 }

12

13 bool CMenu::getRunning()

14 {

15 return m\_running;

16 }

17 void CMenu::setRunning(bool \_running)

18 {

19 m\_running = \_running;

20 }

21

22 size\_t CMenu::getCount() const

23 {

24 return m\_count;

25 }

26

27 std::string CMenu::getTitle()

28 {

29 return m\_title;

30 }

31

32 CMenuItem \*CMenu::getItems()

33 {

34 return m\_items;

35 }

36

37 void CMenu::print()

38 {

39

40 for (size\_t i{}; i < m\_count - 1; ++i)

41 {

42 std::cout << i + 1 << ". ";

43 m\_items[i].print();

44 std::cout << std::endl;

45 }

46 std::cout << "0. ";

47 m\_items[m\_count - 1].print();

48 std::cout << std::endl;

49 }

50

51 void CMenu::printTitle()

52 {

53 std::cout << "\033[2J\033[1;1H";

54 std::cout << "\t" << m\_title << std::endl;

55 }

56

57 int CMenu::runCommand()

58 {

59 printTitle();

60 print();

61 std::cout << "\n Select >> ";

62 std::string SelectInput;

63 bool flag = true;

64 std::cin >> SelectInput;

65 for (int i{0}; i < SelectInput.size(); i++)

66 {

67 if (!(SelectInput[i] >= '0' && SelectInput[i] <= '9'))

68 {

69 flag = false;

70 }

71 }

72 if (flag)

73 {

74 m\_select = std::stoi(SelectInput);

75 }

76 else

77 {

78 std::cout << "\033[2J\033[1;1H";

79 throw MyError{"Wrong input. Enter only number."};

80

81 std::cout << "Нажмите Enter, чтобы продолжить...";

82 std::cin.clear();

83 std::cin.ignore(1024, '\n');

84 std::cin.get();

85 std::cin.clear();

86 std::cin.ignore(1024, '\n');

87

88 return 1;

89 }

90 if (m\_select == 0)

91 {

92 return m\_items[m\_count - 1].run();

93 }

94 else

95 {

96 if ((m\_select > m\_count - 1) || (m\_select < 0))

97 {

98 std::cout << "\033[2J\033[1;1H";

99 throw MyError{"Wrong input. Enter correct number of menu."};

100

101 std::cout << "Нажмите Enter, чтобы продолжить...";

102 std::cin.clear();

103 std::cin.ignore(1024, '\n');

104 std::cin.get();

105 std::cin.clear();

106 std::cin.ignore(1024, '\n');

107

108 return 1;

109 }

110 else

111 {

112 std::cout << "\033[2J\033[1;1H";

113 return m\_items[m\_select - 1].run();

114 }

115 }

116 }

**Cmenu.h**

1 *#ifndef MYMENU\_CMENU\_H*

2 *#define MYMENU\_CMENU\_H*

3

4 *#include* *"CMenuItem.h"*

5 *#include* *<cstddef>*

6

7 class CMenu

8 {

9 public:

10 CMenu();

11 CMenu(std::string, CMenuItem \*, size\_t);

12 int getSelect() const;

13 bool getRunning();

14 void setRunning(bool);

15 std::string getTitle();

16 size\_t getCount() const;

17 CMenuItem \*getItems();

18 void print();

19 void printTitle();

20 int runCommand();

21

22 private:

23 int m\_select{-1};

24 size\_t m\_count{};

25 bool m\_running{true};

26 std::string m\_title{};

27 CMenuItem \*m\_items{};

28 };

29

30 *#endif // MYMENU\_CMENU\_H*

**CmenuItem.h**

1 *#ifndef MYMENU\_CPP\_CMENUITEM\_H*

2 *#define MYMENU\_CPP\_CMENUITEM\_H*

3 *#include* *<string>*

4 *#include* *<functional>*

5

6 using Func = std::function<int()>;

7

8 class CMenuItem

9 {

10 public:

11 CMenuItem(std::string, Func);

12 Func m\_func{};

13 std::string m\_item\_name{};

14 std::string getName();

15 void print();

16 int run();

17 };

18

19 *#endif // MYMENU\_CPP\_CMENUITEM\_H*

**CmenuItem.cpp**

1 *#include* *"CMenuItem.h"*

2 *#include* *<iostream>*

3

4 CMenuItem::CMenuItem(std::string item\_name, Func func) : m\_item\_name(item\_name), m\_func(func) {}

5

6 std::string CMenuItem::getName()

7 {

8 return m\_item\_name;

9 }

10

11 void CMenuItem::print()

12 {

13 std::cout << m\_item\_name;

14 }

15

16 int CMenuItem::run()

17 {

18 return m\_func();

19 }

**DoubleList.cpp**

1 *#include* *"DoubleList.h"*

2

3 DoubleList::DoubleList()

4 {

5 head = nullptr;

6 tail = nullptr;

7 size = 0;

8 }

9

10 DoubleList::~DoubleList()

11 {

12 if (IsEmpty())

13 {

14 return;

15 }

16 Node\* curr = head->next;

17 while (curr)

18 {

19 delete curr->prev;

20 curr = curr->next;

21 }

22 delete tail;

23 }

24

25 bool DoubleList::IsEmpty() const

26 {

27 return !size;

28 }

29

30 void DoubleList::PrintDoubleList() const

31 {

32 std::cout << \*this;

33 }

34

35 void DoubleList::PushBack(int data)

36 {

37 Node\* temp = new Node;

38 temp->data = data;

39 if (IsEmpty())

40 {

41 temp->prev = nullptr;

42 head = temp;

43 }

44 else

45 {

46 tail->next = temp;

47 temp->prev = tail;

48 }

49 temp->next = nullptr;

50 tail = temp;

51 size++;

52 }

53

54 void DoubleList::PushFront(int data)

55 {

56 Node\* temp = new Node;

57 temp->data = data;

58 if (IsEmpty())

59 {

60 temp->next = nullptr;

61 tail = temp;

62 }

63 else

64 {

65 head->prev = temp;

66 temp->next = head;

67 }

68 temp->prev = nullptr;

69 head = temp;

70 size++;

71 }

72

73 void DoubleList::Insert(size\_t index, int data)

74 {

75 if (index >= 0 && index <= size)

76 {

77 if (index == 0)

78 {

79 PushFront(data);

80 return;

81 }

82 if (index == size)

83 {

84 PushBack(data);

85 return;

86 }

87 Node\* currLeft = head;

88 for (int i = 0; i < index - 1; i++)

89 {

90 currLeft = currLeft->next;

91 }

92 Node\* currRight = currLeft->next;

93 Node\* temp = new Node;

94 temp->data = data;

95 temp->prev = currLeft;

96 temp->next = currRight;

97 currLeft->next = temp;

98 currRight->prev = temp;

99 size++;

100 }

101 }

102

103 void DoubleList::PopBack()

104 {

105 if (IsEmpty())

106 {

107 return;

108 }

109 if (size == 1)

110 {

111 delete head;

112 head = nullptr;

113 tail = nullptr;

114 size--;

115 return;

116 }

117 tail = tail->prev;

118 delete tail->next;

119 tail->next = nullptr;

120 size--;

121 }

122

123 void DoubleList::PopFront()

124 {

125 if (IsEmpty())

126 {

127 return;

128 }

129 if (size == 1)

130 {

131 delete head;

132 head = nullptr;

133 tail = nullptr;

134 size--;

135 return;

136 }

137 head = head->next;

138 delete head->prev;

139 head->prev = nullptr;

140 size--;

141 }

142

143 void DoubleList::Remove(size\_t index)

144 {

145 if (index >= 0 && index <= size)

146 {

147 if (IsEmpty())

148 {

149 return;

150 }

151 if (index == 0)

152 {

153 PopFront();

154 return;

155 }

156 if (index == size - 1)

157 {

158 PopBack();

159 return;

160 }

161 Node\* currLeft = head;

162 for (int i = 0; i < index - 1; i++)

163 {

164 currLeft = currLeft->next;

165 }

166 Node\* currRight = currLeft->next->next;

167 currLeft->next = currRight;

168 delete currRight->prev;

169 currRight->prev = currLeft;

170 size--;

171 }

172 }

173

174 void DoubleList::Clear()

175 {

176 this->~DoubleList();

177 head = nullptr;

178 tail = nullptr;

179 size = 0;

180 }

181

182 int DoubleList::FindElement(int data) const

183 {

184 size\_t index = 0;

185 Node\* curr = head;

186 bool find = false;

187 while (curr)

188 {

189 if (curr->data == data)

190 {

191 find = true;

192 break;

193 }

194 curr = curr->next;

195 index++;

196 }

197 return (find ? index : -1);

198 }

199

200 int DoubleList::RFindElement(int data) const

201 {

202 size\_t index = size - 1;

203 Node\* curr = tail;

204 bool find = false;

205 while (curr)

206 {

207 if (curr->data == data)

208 {

209 find = true;

210 break;

211 }

212 curr = curr->prev;

213 index--;

214 }

215 return (find ? index : -1);

216 }

217

218 size\_t DoubleList::getSize() const

219 {

220 return size;

221 }

222

223 DoubleList::Node\* DoubleList::getHead() const

224 {

225 return head;

226 }

227

228 DoubleList::Node\* DoubleList::getTail() const

229 {

230 return tail;

231 }

232

233 std::ostream& operator<<(std::ostream& out, const DoubleList& dl)

234 {

235 DoubleList::Node\* curr = dl.getHead();

236 out << "[";

237 while (curr)

238 {

239 out << curr->data << (curr->next ? ", " : "");

240 curr = curr->next;

241 }

242 out << "] \nsize: " << dl.getSize() << "\n";

243 return out;

244 }

**DoubleList.h**

1 *#pragma once*

2 *#include* *<iostream>*

3

4 class DoubleList

5 {

6 public:

7 struct Node

8 {

9 int data;

10 Node\* next;

11 Node\* prev;

12 };

13

14 DoubleList();

15 ~DoubleList();

16 bool IsEmpty() const;

17 void PrintDoubleList() const;

18 void PushBack(int data);

19 void PushFront(int data);

20 void Insert(size\_t index, int data);

21 void PopBack();

22 void PopFront();

23 void Remove(size\_t index);

24 void Clear();

25 int FindElement(int data) const;

26 int RFindElement(int data) const;

27 size\_t getSize() const;

28 Node\* getHead() const;

29 Node\* getTail() const;

30 friend std::ostream& operator<<(std::ostream& out, const DoubleList& dl);

31

32 private:

33 Node\* head;

34 Node\* tail;

35 size\_t size;

36 };

**FileLogging.h**

1 *#pragma once*

2 *#ifdef \_MSC\_VER*

3 *#define \_CRT\_SECURE\_NO\_WARNINGS*

4 *#endif*

5 *#include* *<string>*

6 *#include* *<fstream>*

7 *#include* *<ctime>*

8 *#include* *<iostream>*

9

10 class FileLogging

11 {

12 public:

13 FileLogging(std::string fileName);

14 void Logging(std::string message);

15 void PrintFile();

16

17 private:

18 std::string getTime();

19 std::string fileName;

20 };

**FileLogging.cpp**

1 *#include* *"FileLogging.h"*

2

3 FileLogging::FileLogging(std::string fileName)

4 {

5 this->fileName = fileName;

6 }

7

8 void FileLogging::Logging(std::string message)

9 {

10 std::ofstream fout(fileName, std::ios::out | std::ios::app);

11 if (fout.is\_open())

12 {

13 fout << "[" << getTime() << "]" << message;

14 }

15 fout.close();

16 }

17

18 std::string FileLogging::getTime()

19 {

20 time\_t seconds = time(NULL);

21 tm\* timeinfo = localtime(&seconds);

22 std::string currTime = asctime(timeinfo);

23 currTime.pop\_back();

24 return currTime;

25 }

26

27 void FileLogging::PrintFile()

28 {

29 std::ifstream fin(fileName, std::ios::in);

30 std::string temp;

31 std::cout << fileName << ":\n";

32 if (fin.is\_open())

33 {

34 while (std::getline(fin, temp))

35 {

36 std::cout << temp << std::endl;

37 }

38 }

39 }

**MyError.cpp**

1 *#define \_CRT\_SECURE\_NO\_WARNINGS*

2 *#include* *"MyError.h"*

3 *#include* *<chrono>*

4 *#include* *<fstream>*

5 *#include* *<iostream>*

6

7 const char \*MyError::getError() const

8 {

9 return m\_error.c\_str();

10 }

11

12 MyError::MyError(std::string error)

13 {

14 m\_error = error;

15 logging();

16 }

17

18 void MyError::logging()

19 {

20 std::fstream file;

21 auto now = std::chrono::system\_clock::now();

22 std::time\_t end\_time = std::chrono::system\_clock::to\_time\_t(now);

23 file.open(m\_file, std::ios::app);

24 file << "WARNING: " << m\_error.c\_str() << "|" << std::ctime(&end\_time);

25 file.close();

26 }

**MyError.h**

1 *#ifndef MYERROR\_H*

2 *#define MYERROR\_H*

3 *#include* *<string>*

4 *#include* *<string\_view>*

5

6 class MyError

7 {

8 public:

9 MyError(std::string error);

10 static std::string m\_file;

11 const char \*getError() const;

12

13 private:

14 void logging();

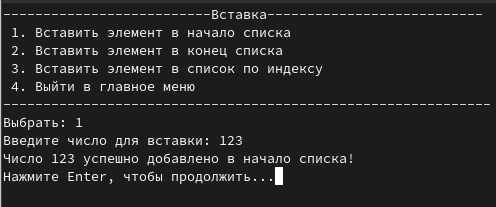
15 std::string m\_error;

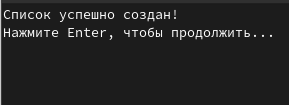
16 };

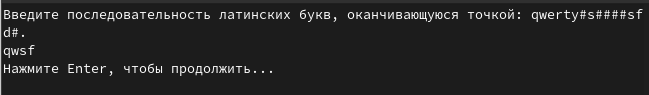
17

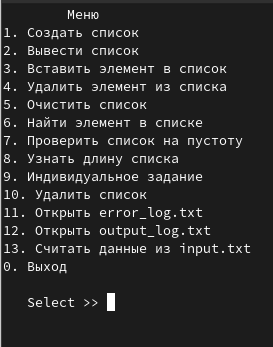
18 *#endif // MYERROR\_H*

**Результат работы:**

****

****

****

****

**Вывод:** в ходе работы были сформированы практические навыки создания алгоритмов обработки списочных структур данных.