3.1 Пример - пример пользовательского типа для реализации динамического массива

```
1  /**
2  \file   simpleDynamicArray.cpp
3  \brief  пример работы с пользовательским типом данных динамический массив
4  \author Полевой Дмитрий
5  \date   27.02.2013
6  */
7  #include <cstdio>
8  #include <cassert>
9  #include <cstring>
10
11 /**
12 \brief дескриптор динамического массива
13 */
14 struct DynamicArray
15 {
16     int m_size;   // размер массива
17     int* m_pData; // элементы массива
18 };
19
20 /**
21 \brief проверка массива на пустоту
22 \param arr - дескриптор массива
23 \retval bool - true
24 */
25 bool arrayIsEmpty(const DynamicArray& arr)
26 {
27     return (0 == arr.m_size) && (0 == arr.m_pData);
28 }
29
30 /**
31 \brief получение размера массива
32 \param arr - дескриптор массива
33 \retval int - размер массива
34 */
35 int arraySize(const DynamicArray& arr)
36 {
37     return arr.m_size;
38 }
39
40 /**
41 \brief создание массива
42 \param arr - дескриптор пустого массива
43 \param size - размер массива
44 */
45 void createArray(DynamicArray& arr, const int size)
46 {
47     assert(arrayIsEmpty(arr));
48     assert(0 < size);
49     if (arrayIsEmpty(arr))
50     {
51         arr.m_size = size;
52         arr.m_pData = new int[arr.m_size];
53         memset(arr.m_pData, 0, arr.m_size * sizeof(*arr.m_pData));
54     }
55 }
56
57 /**
58 \brief уничтожение массива и очистка дескриптора
59 \param arr - дескриптор массива
```

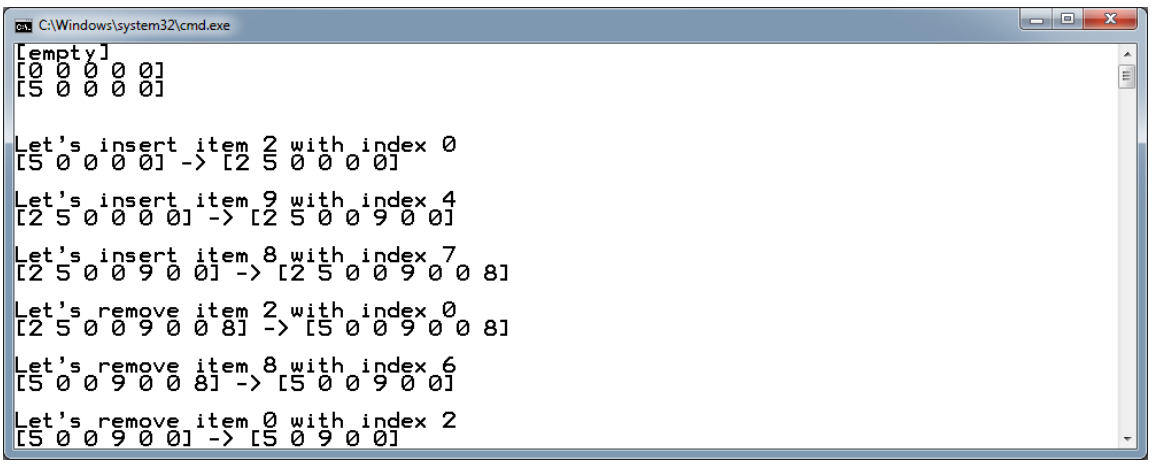
```
60  */
61  void destroyArray(DynamicArray& arr)
62  {
63      arr.m_size = 0;
64      delete[] arr.m_pData;
65      arr.m_pData = 0;
66  }
67
68  /**
69  \brief получение доступа к элементу массива (на чтение и запись)
70  \param arr - дескриптор массива
71  \param i - индекс элемента в диапазоне от 0 до arraySize() - 1
72  \retval int& - элемент массива
73  */
74  int& getArrayItemAt(DynamicArray& arr, const int i)
75  {
76      assert(0 == arrayIsEmpty(arr));
77      assert(0 <= i);
78      assert(i < arr.m_size);
79      static int dummyItem(0);  // страховочный элемент
80      int* pAnswer(&dummyItem); // ответ
81      if ((0 != arr.m_pData) && (0 <= i) && ((i < arr.m_size)))
82      // индекс внутри границ массива
83      {
84          pAnswer = arr.m_pData + i;
85      }
86      return *pAnswer;
87  }
88
89  /**
90  \brief получение доступа к элементу массива (на чтение)
91  \param arr - дескриптор массива
92  \param i - индекс элемента в диапазоне от 0 до arraySize() - 1
93  \retval const int& - элемент массива
94  */
95  const int& getArrayItemAt(const DynamicArray& arr, const int i)
96  {
97      assert(0 == arrayIsEmpty(arr));
98      assert(0 <= i);
99      assert(i < arr.m_size);
100     static int dummyItem(0);        // страховочный элемент
101     const int* pAnswer(&dummyItem); // ответ
102     if ((0 != arr.m_pData) && (0 <= i) && ((i < arr.m_size)))
103     // индекс внутри границ массива
104     {
105         pAnswer = arr.m_pData + i;
106     }
107     return *pAnswer;
108 }
109
110 /**
111 \brief вставка элемент в указанную позицию
112 \param arr - дескриптор массива
113 \param i - индекс позиции вставляемого элемента в диапазоне от 0 до arraySize()
114 \param val - вставляемое значение
115 */
116 void insertArrayItemAt(DynamicArray& arr, const int i, const int val)
117 {
118     assert(0 == arrayIsEmpty(arr));
119     assert(0 <= i);
120     assert(i <= arraySize(arr));
121     int* pNewData(new int[arraySize(arr) + 1]);
122     // скопируем элементы до вставляемого элемента
```

```
123     memcpy(pNewData, arr.m_pData, i * sizeof(*arr.m_pData));
124     // "вставим" элемент
125     pNewData[i] = val;
126     // скопируем элементы после вставляемого элемента
127     memcpy(pNewData + i + 1, arr.m_pData + i, (arraySize(arr) - i) * sizeof(*arr.m_pData));
128     // подменим массив в дескрипторе на новый
129     delete[] arr.m_pData;
130     arr.m_pData = pNewData;
131     arr.m_size += 1;
132 }
133 
134 /**
135 \brief удаление элемента в указанной позиции
136 \param arr - дескриптор массива
137 \param i - индекс позиции удаляемого элемента в диапазоне от 0 до arraySize() - 1
138 */
139 void removeArrayItemAt(DynamicArray& arr, const int i)
140 {
141     assert(0 == arrayIsEmpty(arr));
142     assert(0 <= i);
143     assert(i < arraySize(arr));
144     // скопируем элементы после вставляемого элемента со сдвигом влево на 1
145     memcpy(arr.m_pData + i, arr.m_pData + i + 1, (arraySize(arr) - i - 1) * sizeof(*arr.m_pData));
146     arr.m_size -= 1;
147 }
148 
149 /**
150 \brief конкатенация двух массивов (последовательное объединение)
151 \param arr - дескриптор массива для результат нужного размера
152 \param srcL - дескриптор первого непустого массива
153 \param srcR - дескриптор второго непустого массива
154 */
155 void concatArrays(DynamicArray& dest, const DynamicArray& srcL, const DynamicArray& srcR)
156 {
157     assert(0 == arrayIsEmpty(dest));
158     assert(0 != arrayIsEmpty(srcL));
159     assert(0 != arrayIsEmpty(srcR));
160     assert(arraySize(dest) == arraySize(srcL) + arraySize(srcR));
161     memcpy(dest.m_pData, srcL.m_pData, arraySize(srcL) * sizeof(*srcL.m_pData));
162     memcpy(dest.m_pData + arraySize(srcL), srcR.m_pData, arraySize(srcL) * sizeof(*srcL.m_pData));
163 }
164 
165 /**
166 \brief печать массива
167 \param arr - дескриптор печатаемого массива
168 */
169 void printArray(const DynamicArray& arr)
170 {
171     assert(arrayIsEmpty(arr) || ((0 != arr.m_pData) && (0 != arr.m_size)));
172     printf("[");
173     if (0 == arrayIsEmpty(arr))
174     // непустой массив
175     {
176       for (int i(0); i < arr.m_size; ++i)
177       // по всем элементам массива
178       {
179           printf("%d%s", arr.m_pData[i], (i != arr.m_size - 1) ? " " : "");
180       }
181     }
182     else
183     // пустой массив
184     {
185         printf("empty");
```

```
186     }
187     printf("]");
188 }
189 
190 int main()
191 {
192     DynamicArray arr = {0};
193     printArray(arr);
194     printf("\n");
195 
196     createArray(arr, 5);
197     printArray(arr);
198     printf("\n");
199 
200     getArrayItemAt(arr, 0) = 5;
201     printArray(arr);
202     printf("\n");
203 
204     int i(0);
205     int val(0);
206     val = 2;
207     i = 0;
208     printf("\n\nLet's insert item %d with index %d\n", val, i);
209     printArray(arr);
210     insertArrayItemAt(arr, i, val);
211     printf(" -> ");
212     printArray(arr);
213 
214     val = 9;
215     i = 4;
216     printf("\n\nLet's insert item %d with index %d\n", val, i);
217     printArray(arr);
218     insertArrayItemAt(arr, i, val);
219     printf(" -> ");
220     printArray(arr);
221 
222     val = 8;
223     i = arraySize(arr);
224     printf("\n\nLet's insert item %d with index %d\n", val, i);
225     printArray(arr);
226     insertArrayItemAt(arr, i, val);
227     printf(" -> ");
228     printArray(arr);
229 
230     i = 0;
231     printf("\n\nLet's remove item %d with index %d\n", getArrayItemAt(arr, i), i);
232     printArray(arr);
233     removeArrayItemAt(arr, i);
234     printf(" -> ");
235     printArray(arr);
236 
237     i = arraySize(arr) - 1;
238     printf("\n\nLet's remove item %d with index %d\n", getArrayItemAt(arr, i), i);
239     printArray(arr);
240     removeArrayItemAt(arr, i);
241     printf(" -> ");
242     printArray(arr);
243 
244     i = 2;
245     printf("\n\nLet's remove item %d with index %d\n", getArrayItemAt(arr, i), i);
246     printArray(arr);
247     removeArrayItemAt(arr, i);
248     printf(" -> ");
```

```
249     printArray(arr);
250     printf("\n");
251
252     destroyArray(arr);
253 }
```

Результаты работы программы (simpleDynamicArray.exe)



3.2 Задания (лабораторная работа 3)

3.2.1 Копирование и доработка примера

- в существующее решение добавить проект simpleDynamicArray
- набрать (именно набрать, а не скопировать-вставить) текст программы и произвести отладку (программа должна собираться без ошибок и запускаться)
- убедится, что результат работы программы совпадает с указанным
- дописать тест для непротестированных функций

3.2.2 Переработка примера (бонус трек для 4-5)

- в существующее решение добавить проект smartDynamicArray
- реализовать логику буфера и реалокации массива только при расширении свыше размеров буфера

3.2.3 Решение задач на обработку массивов

Решить не меньше двух задач из задач для самостоятельного решения раздела 3.2 методички 2056 (стр. 84-86, выбрать из диапазона номеров 20-45).

Имена проектов и файлов дожны иметь вид qw03_2_XX, где XX - номер задачи с ведущим нулем.