Министерство науки и высшего образования Российской Федерации
Федеральное государственное
бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ	«Фундаментальные Науки»
КАФЕДРА	ФН-12 «Математическое моделирование»

ОТЧЕТ
ПО РУБЕЖНОМУ КОНТРОЛЮ №1
по дисциплине «Типы и структуры данных»
Тема: «Стек и очередь»

Выполнил студент гр. ФН12-31Б: _____ Лямин И.С.
дата, подпись Ф.И.О.

Проверил преподаватель: _____ Волкова Л. Л.
дата, подпись Ф.И.О.

Москва, 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В рубежном контроле будет реализовано две циклические структуры данных, циклическая очередь и стек.

Цель работы: Реализовать структуры данных циклический стек, циклическую очередь и интерфейс взаимодействия с ними. Для достижения поставленной цели требуется выполнить следующие задачи.

1. Описать необходимые структуры данных.

2. Реализовать структуры данных.

3. Реализовать методы интерфейса для взаимодействия с СД.

1 Аналитическая часть

Стек — структура данных которая позволяет извлечь только последний помещённый в неё элемент (FILO - первый зашёл, последний вышел).

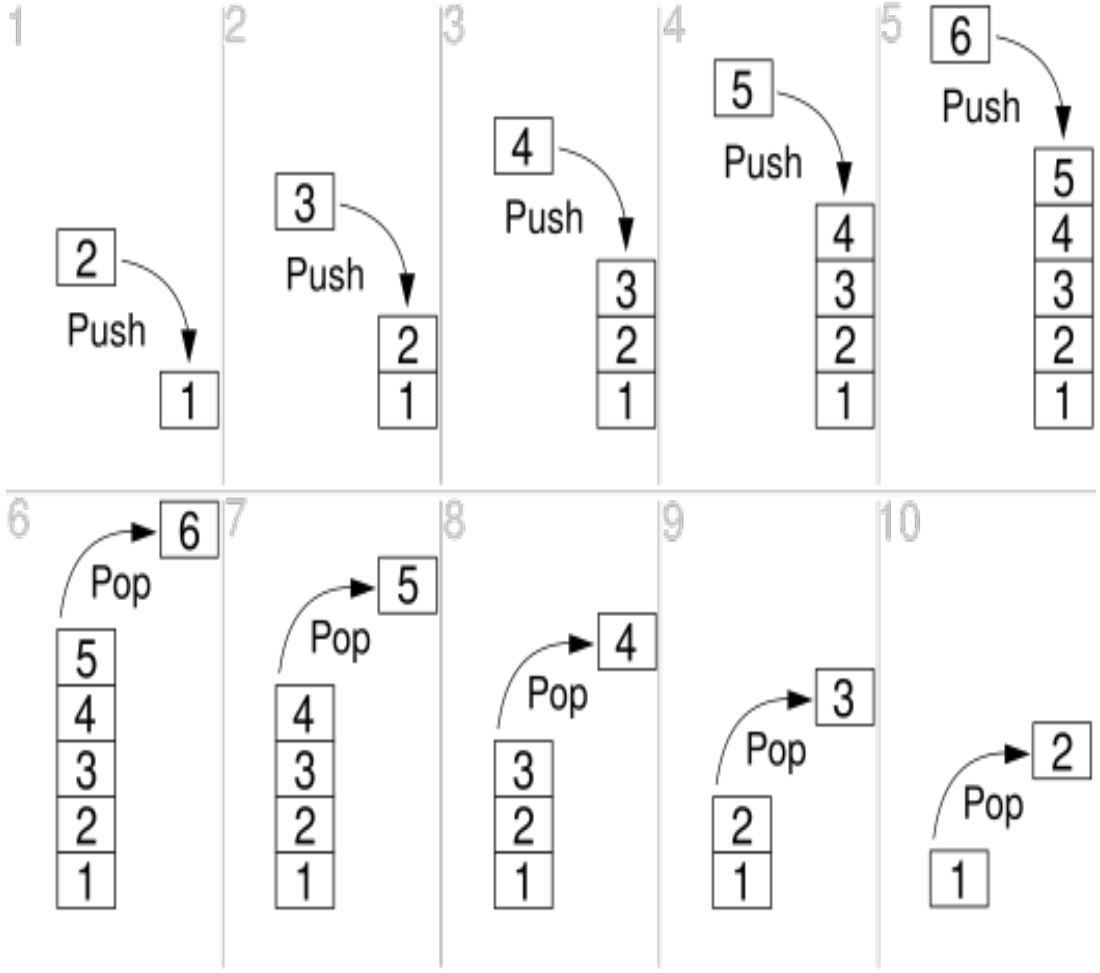

Рисунок 1 — Схема работы стека

Интерфейс стека:

1. Push - добавление элемента в конец массива.
2. Pop - удаление последнего элемента.
3. IsFull - проверка на полноту.
4. IsEmpty - проверка на пустоту.
5. Leek - вывод в консоль массива изначального с индесами и пользовательского массива.

Циклическая очередь — это структура данных позволяющая извлеч только первый помешённый в неё элемент (FIFO - первый зашёл, первый вышел). Циклическая очередь от обычной отличается тем, что если при полном заполнении памяти удалить один элемент и попытаться поместить на его место новый помещённый на его место элемент будет первым только в изначальном массиве, отображаться он будет как последний.

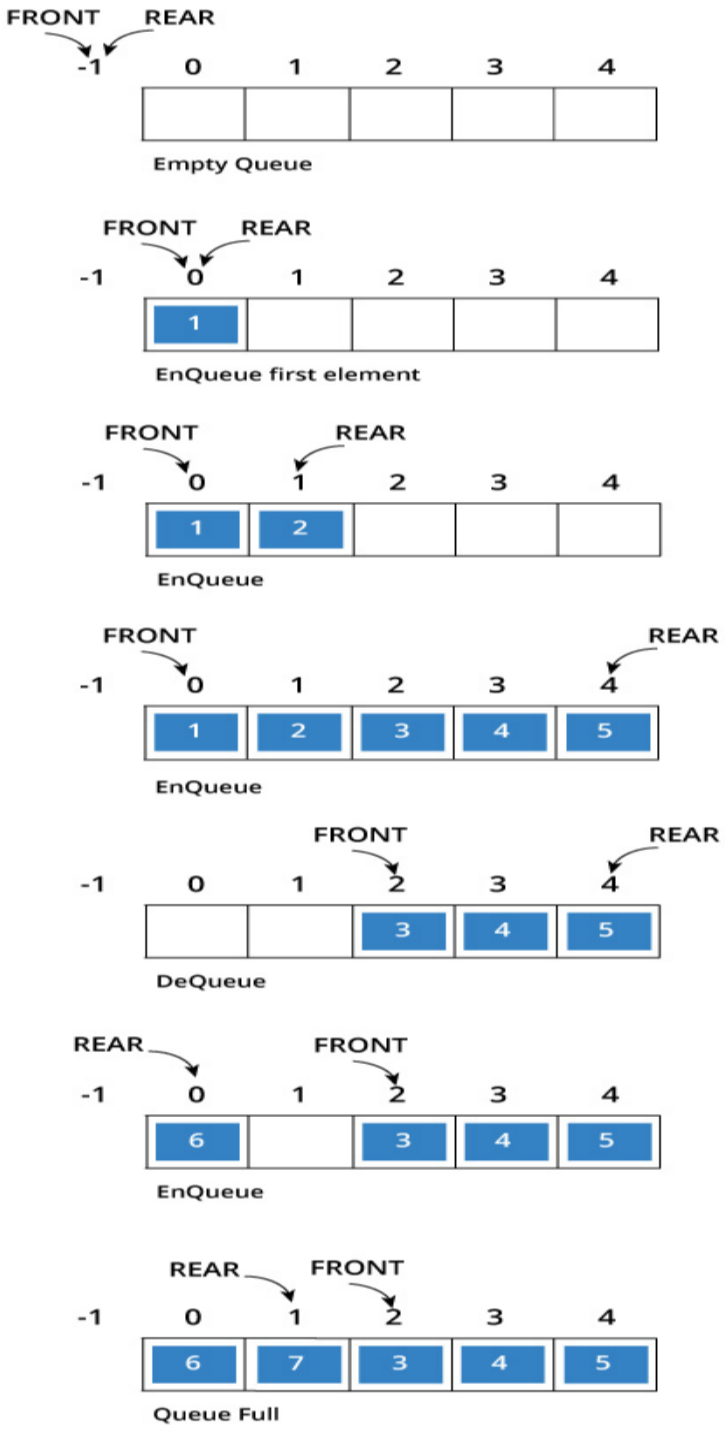

Рисунок 2 — Схема работы циклической очереди

Интерфейс очереди:

1. Push - добавление элемента в конец массива.
2. Delete - удаление первого элемента.
3. IsFull - проверка на полноту.
4. IsEmpty - проверка на пустоту.
5. Leek - вывод в консоль массива изначального с индесами и пользовательского массива.

2 Конструкторская часть

На рисунках 3–8 представлены функциональнальные схемы функций, осуществляющих работу со стеком и очередью. Функциональные схемы даны в нотации IDEF0.

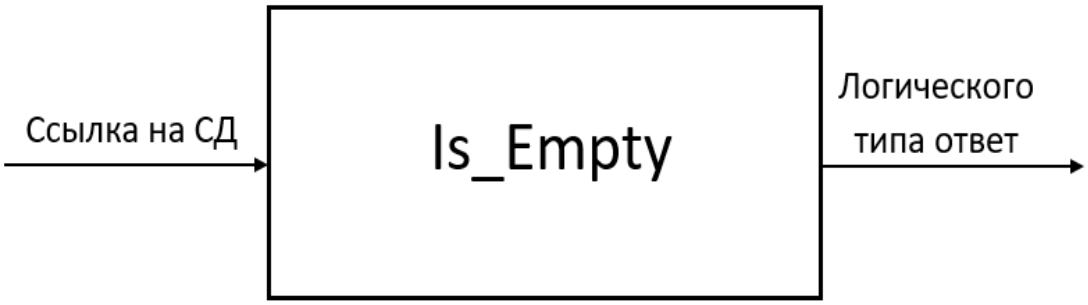

Рисунок 3 — Функция проверки на пустоту СД

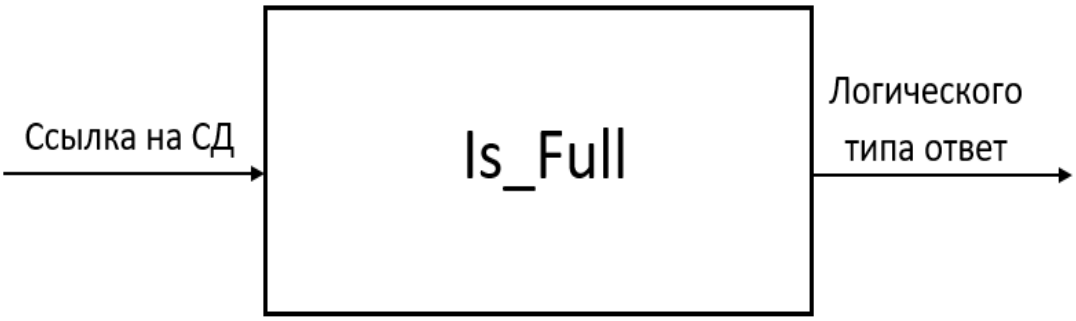

Рисунок 4 — Функция проверки на полноту СД

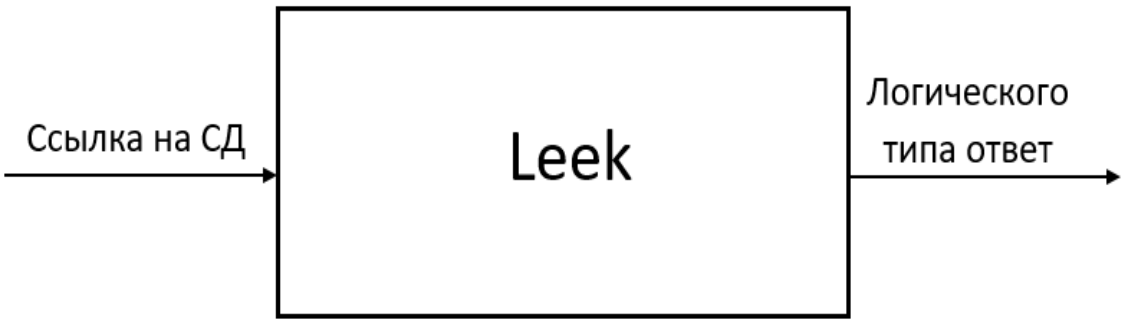

Рисунок 5 — Функция вывода информации о СД

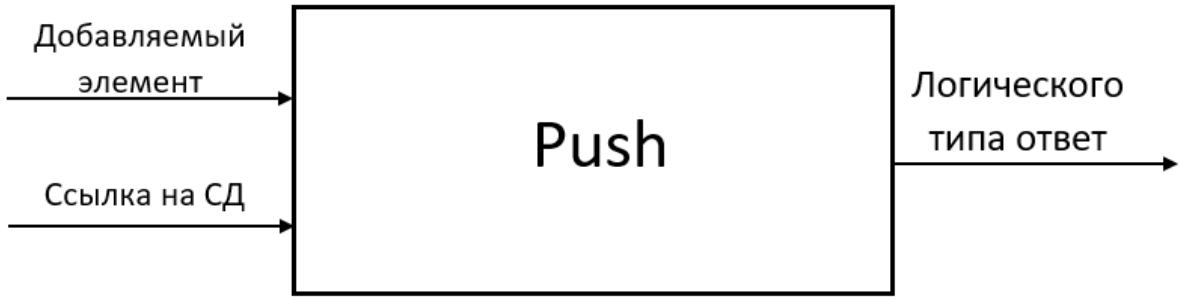

Рисунок 6 — функция добавления элемента в конец СД

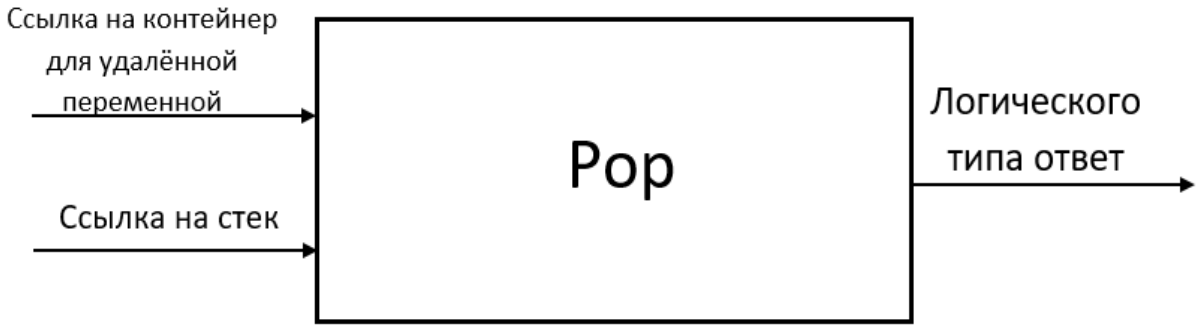

Рисунок 7 — функция удаления последнего элемента из стека

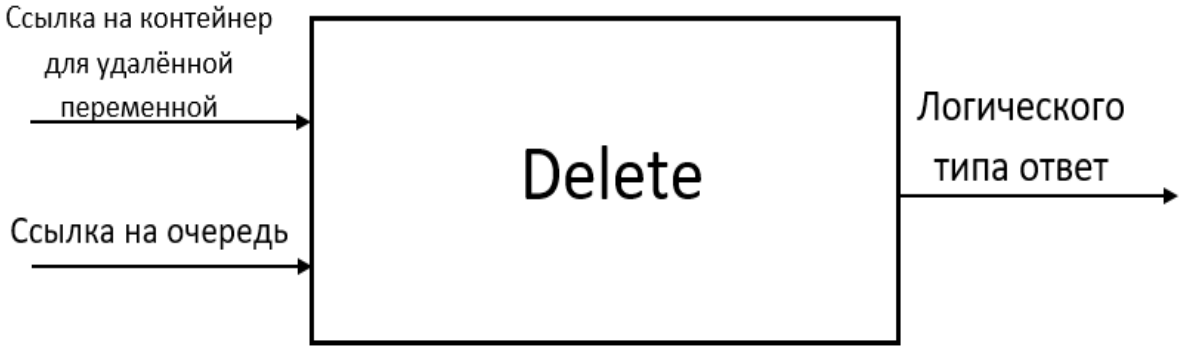

Рисунок 8 — функция удаления первого элемента из очереди

3 Технологическая часть

3.1 Выбор средств реализации

Для программной реализации СД использовалась среда разработки Visual Studio, язык программирования, на котором была выполнена реализации СД, — C++. Исследование проводилось на ноутбуке (64-разрядная операционная система, процессор x64, частота процессора 3.10 ГГц, оперативная память 16 ГБ)

3.2 Реализация СД

В листинге 1 можно увидеть программную реализацию описанных СД.

Листинг 1 — Программная реализация

```
struct Turn{
        int capacity = 0;
        int col_elem = 0;
        int array[SIZE];
        int start = 0;
        int end = 0;

        Turn(int capacity) {
                this->capacity = capacity;
                for (int i = 0; i < capacity; i++) {
                        this->array[i] = -1;
                }
        }
};


struct Stack {
        int capacity = 0;
        int col_elem = 0;
        int array[SIZE];
        int end = 0;
        int start = 0;

        Stack(int capacity) {
                this->capacity = capacity;
                for (int i = 0; i < capacity; i++) {
                        this->array[i] = -1;
                }
        }
};

template<typename DS>
bool IsEmpty(DS* turn) {
        if (turn->col_elem == 0) return true;
```

```
35          return false;
36  }
37
38  bool Pop(Stack* stack, int* contain) {
39          if (IsEmpty(stack)) return false;
40          *contain = stack->array[stack->end];
41          stack->array[stack->end] = -1;
42          stack->end -= 1;
43          cout << "Number " << (*contain) << " extracted from stack" <<
    ↪  endl;
44          stack->col_elem -= 1;
45          return true;
46  }
47
48  template<typename DS>
49  bool IsFull(DS* turn) {
50          if (turn->capacity == turn->col_elem) return true;
51          return false;
52  }
53
54
55  bool Push(Turn* turn, int* elem) {
56          if (IsFull(turn)) {
57                  cout << "Array is full" << endl;
58                  return 0;
59          }
60          int iter;
61          for (int i = turn->start; i <= (turn->start + turn->capacity
    ↪  - 1); i++) {
62                  iter  = i % turn->capacity;
63                  if (turn->array[iter] == -1) {
64                          if (IsEmpty(turn)) {
65                                  cout << "Stack was empty" << endl;
66                                  turn->col_elem += 1;
67                                  turn->start = iter;
68                                  turn->end = iter;
69                                  turn->array[iter] = (*elem);
70                                  return 1;
71                          }
72                          cout << "Stack was NOT empty" << endl;
73                          turn->col_elem += 1;
74                          turn->end = iter;
75                          turn->array[iter] = (*elem);
76                          return 1;
77                  }
78          }
79          return 0;
80  }
81
82  bool Push_stack(Stack* turn, int* elem) {
83          if (IsFull(turn)) {
84                  cout << "Array is full" << endl;
```

```
85                  return 0;
86          }
87          int iter;
88          for (int i = 0; i < turn->capacity; i++) {
89                  if (turn->array[i] == -1) {
90                          if (IsEmpty(turn)) {
91                                  cout << "Stack was empty" << endl;
92                                  turn->col_elem += 1;
93                                  turn->end = i;
94                                  turn->array[i] = (*elem);
95                                  return 1;
96                          }
97                          cout << "Stack was NOT empty" << endl;
98                          turn->col_elem += 1;
99                          turn->end = i;
100                         turn->array[i] = (*elem);
101                         return 1;
102                 }
103         }
104         return 0;
105 }
106
107 bool Delete(Turn* turn, int* contain){
108         if (IsEmpty(turn)) return false;
109         if (turn->start == turn->end) {
110                 *contain = turn->array[turn->start];
111                 turn->array[turn->start] = -1;
112                 turn->start = 0;
113                 turn->end = 0;
114                 cout << "Number " << (*contain) << " extracted from
    ↪ stack, stack is now empty" << endl;
115                 turn->col_elem -= 1;
116                 return true;
117         }
118         *contain = turn->array[turn->start];
119         turn->array[turn->start] = -1;
120         turn->start = ((turn->start + 1) % turn->capacity);
121         cout << "Number " << (*contain) << " extracted from stack" <<
    ↪  endl;
122         turn->col_elem -= 1;
123         return true;
124 }
125
126 template<typename DS_2>
127 void print_basic_array(DS_2* turn) {
128         for (int i = 0; i < turn->capacity; i++) {
129                 if ((i == turn->start) && (i == turn->end)) {
130                         cout << turn->array[i] << " --- BOTH" << endl
    ↪ ;
131                         continue;
132                 }
133                 try
```

```
134                 {
135                         if (i == turn->start) {
136                                 cout << turn->array[i] << " --- START
    ↪ " << endl;
137                                 continue;
138                         }
139                 }
140                 catch (const std::exception&)
141                 {
142                         cout << "";
143                 }
144                 if (i == turn->end) {
145                         cout << turn->array[i] << " --- END" << endl;
146                         continue;
147                 }
148                 cout << turn->array[i] << endl;
149         }
150         cout << turn->col_elem << " --- elemnts" << endl;
151 }
152
153 template<typename DS_2>
154 void print(DS_2* turn) {
155         int colis = 0;
156         for (int i = 0; i < (turn->capacity); i++) {
157                 int ind = (i + turn->start) % turn->capacity;
158                 if (turn->array[ind] == -1) {
159                         colis += 1;
160                         continue;
161                 }
162                 cout << "[" << i - colis << "]" << " --- " << turn->
    ↪ array[ind] << endl;
163         }
164         cout << turn->col_elem << " --- elemnts" << endl;
165 }
166
167 void read(int* number) {
168         cout << "Input a nubmer: ";
169
170         while (true) {
171                 cin >> (*number);
172                 if (cin.fail()) {
173                         cin.clear();                                    cin.
    ↪ ignore(numeric_limits<streamsize>::max(), '\n');
174                         cout << "Error! Please, input a valid number:
    ↪  ";
175                 }
176                 else {
177                         break;
178                 }
179         }
180 }
```

4 Исследовательская часть

4.1 Тестирование программы

В таблицах 1 представлены описания тестов по методологии чёрного ящика, все тесты пройдены успешно.

	Описание теста	**Входные данные**	**Ожидаемый результат**	**Полученный результат**
1	проверка на обработку не валидных данных	3	оповещение о некорректности данных и запрос новых	оповещение о некорректности данных и запрос новых
2	проверка на обработку не валидных данных	1 -1	оповещение программы и запрос новых данных	оповещение программы и запрос новых данных
3	работа очереди	2 3 1 2 3 Delete Push 10 IsEmpty IsFull	10-end 2-start 3 No it's not empty Yes it's full	10-end 2-start 3 No it's not empty Yes it's full
4	работа стека	2 3 1 2 3 Pop Push 10 IsEmpty IsFull	2-start 2 10-end No it's not empty Yes it's full	2-start 2 10-end No it's not empty Yes it's full

Таблица 1

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы была достигнута цель - рассмотрены и разобраны две СД (циклическая очередь, стек). Были выполнены все поставленные задачи:

1. Описаны необходимые структуры данных.
2. Реализованны структуры данных.
3. Реализованы методы интерфейса для взаимодействия с СД.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Т.Кормен, Ч.Лейзерсон, Р.Ривест, К.Штайн - Алгоритмы.

2. evileg.com [Электронный ресурс] -
   URL:https://evileg.com/ru/post/472/
   (дата обращения: 25.09.2024).