Министерство науки и высшего образования Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Вятский государственный университет»**

**Колледж ВятГУ**

ОТЧЕТ

ПО ДОМАШНЕЙ КОНТРОЛЬНОЙ РАБОТЕ №6

«РЕАЛИЗАЦИЯ ЭЛЕМЕНТАРНЫХ СТРУКТУР ДАННЫХ НА ОСНОВЕ СТАТИЧЕСКОЙ ПАМЯТИ»

ПО «МДК 05.02 РАЗРАБОТКА КОДА ИНФОРМАЦИОННЫХ СИСТЕМ»

Выполнил:

Студентка учебной группы ИСПк-202 Артюхина Дарья

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2024

**1. Цель работы**

Цель работы заключается в изучении принципов работы с базовыми структурами данных, получении навыков организации case-меню.

**2. Формулировка задания**

Задание:

1. Написать программу для работы со структурой данных "Очередь".

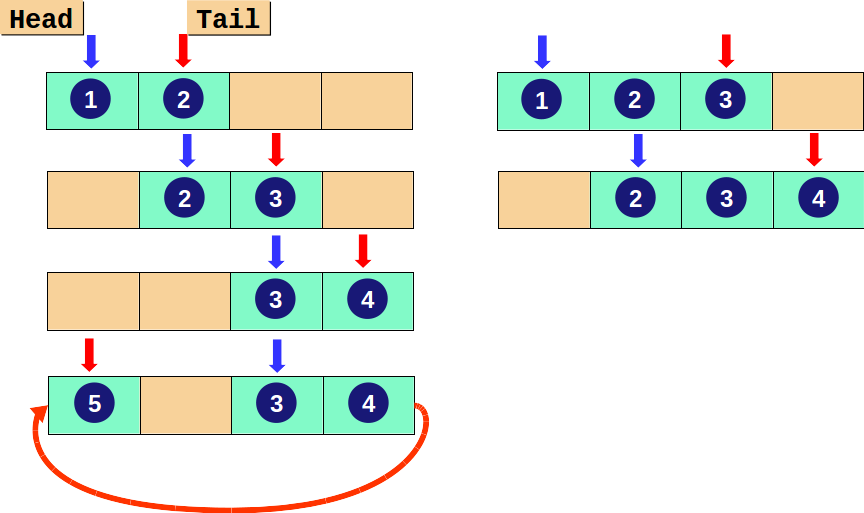
2. Структура данных должна быть реализована на основе статической и динамической памяти.

3. Работа со структурой должна осуществляться с помощью case-меню. Предусмотреть наглядную визуализацию содержимого структуры.

**3. Описание алгоритма**

**Очередь** — это динамическая структура данных, у которой доступны для работы только два элемента: первый и последний. Добавлять в очередь элементы можно только в конец очереди, удаление элементов – только с начала очереди.

Работа с очередью с помощью кольцевого массива: при удалении элемента из очереди происходит сдвиг оставшихся.



**Очередь со статическим типом памяти:**

Stat – запись, содержащая массив date, представляющий очередь, и переменные head и tail, указывающие на начало и конец очереди соответственно. В очередь можно добавлять элементы с помощью процедуры PushStatic и извлекать элементы с помощью процедура PopStatic.

Процедура PushStatic добавляет элемент x в очередь, если она не переполнена. Проверка на переполнение осуществляется сравнением head и tail с учетом кольцевой структуры массива. Если очередь переполнена, выводиться "Очередь переполнена".

Процедура PopStatic извлекает элемент из начала очереди. Проверка на пустоту очереди также осуществляется с учетом кольцевой структуры массива. Если очередь пуста выводит сообщение "Очередь пуста". При удалении элемента из начала очереди, соответствующий элемент массива сменяется на 0.

Оба метода обновляют переменные head и tail для отслеживания начала и конца очереди.

**Очередь с динамическим типом памяти:**

Запись Node является узелом связного списка, содержащий данные data и указатель next на следующий узел. Структура Dynamic представляет собой запись, содержащую указатели на начало head и конец tail очереди.

Процедура PushDynamic добавляет новый элемент x в конец очереди. Создается новый узел NewNode, в который записывается значение x. Затем проверяется, не пуст ли список - если он не пуст, то устанавливается ссылка на новый узел в текущий хвост очереди. После чего обновляется указатель хвоста на новый узел. Если очередь была пуста, то как результат обновляется и указатель на начало очереди.

Процедура PopDynamic извлекает элемент из начала очереди. Проверяется, не является ли очередь пустой - если она пуста, процедура выводит "Очередь пуста". В противном случае, устанавливается указатель top на начало очереди, обновляется указатель на начало очереди и, если после удаления элемента список пуст, обновляется и указатель хвоста. Удаляется узел top при помощи процедуры Dispose.

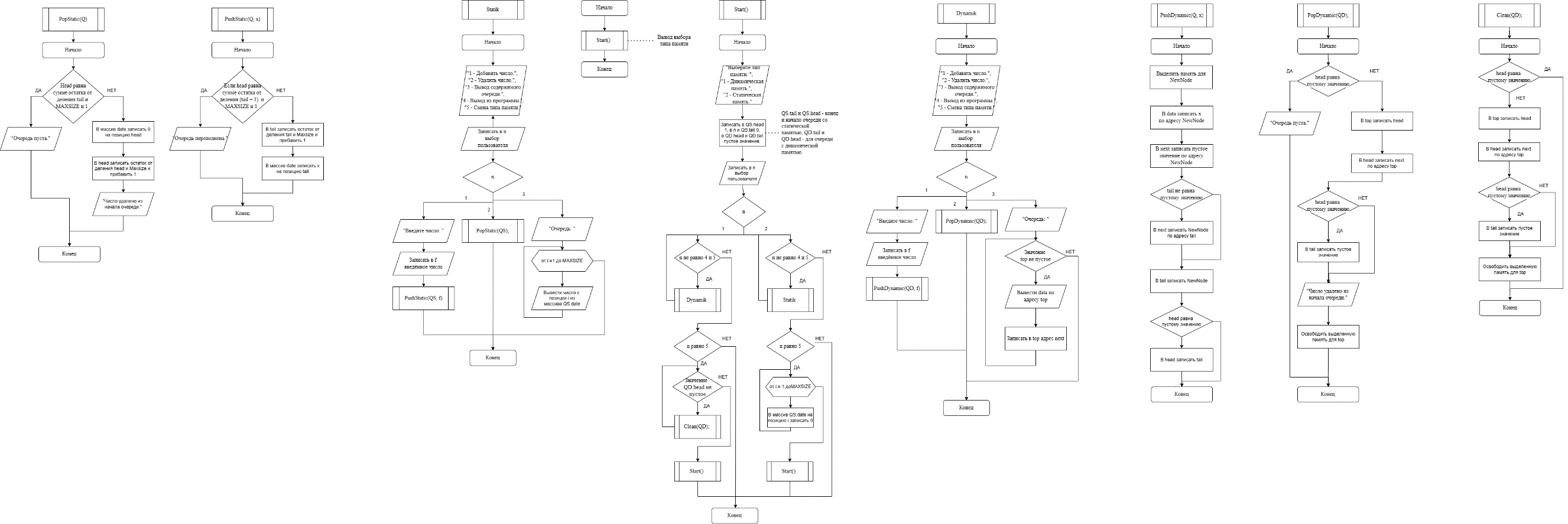
**4. Схема алгоритма**

Рисунок 1 – Основная программа

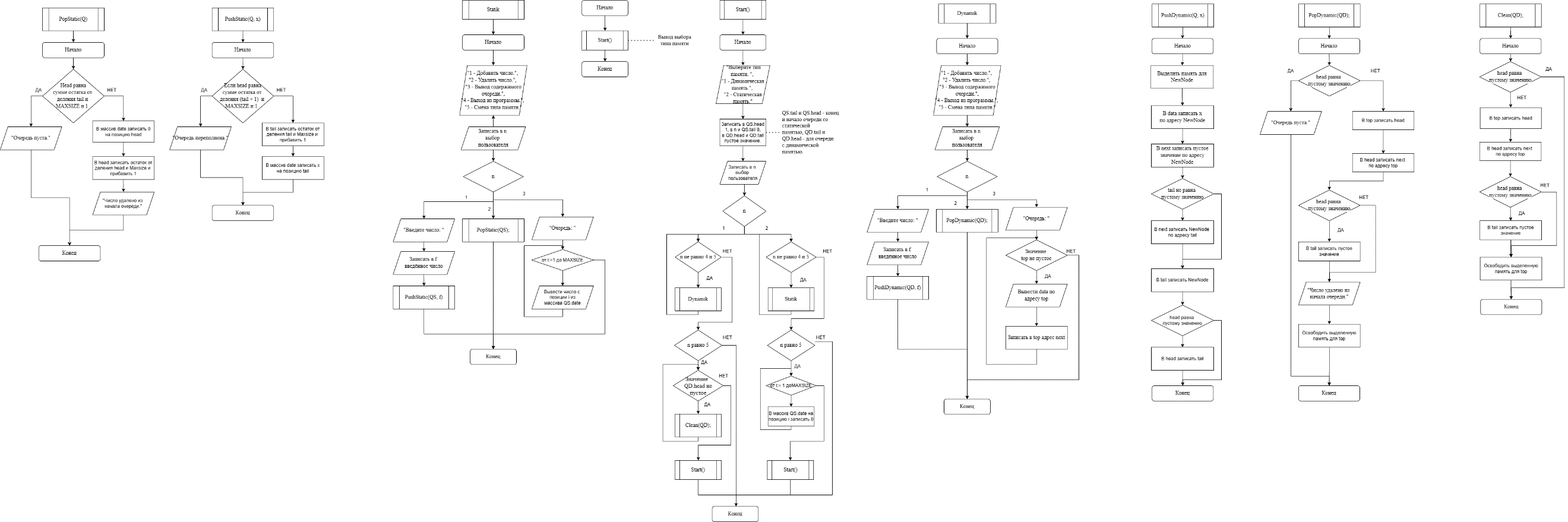
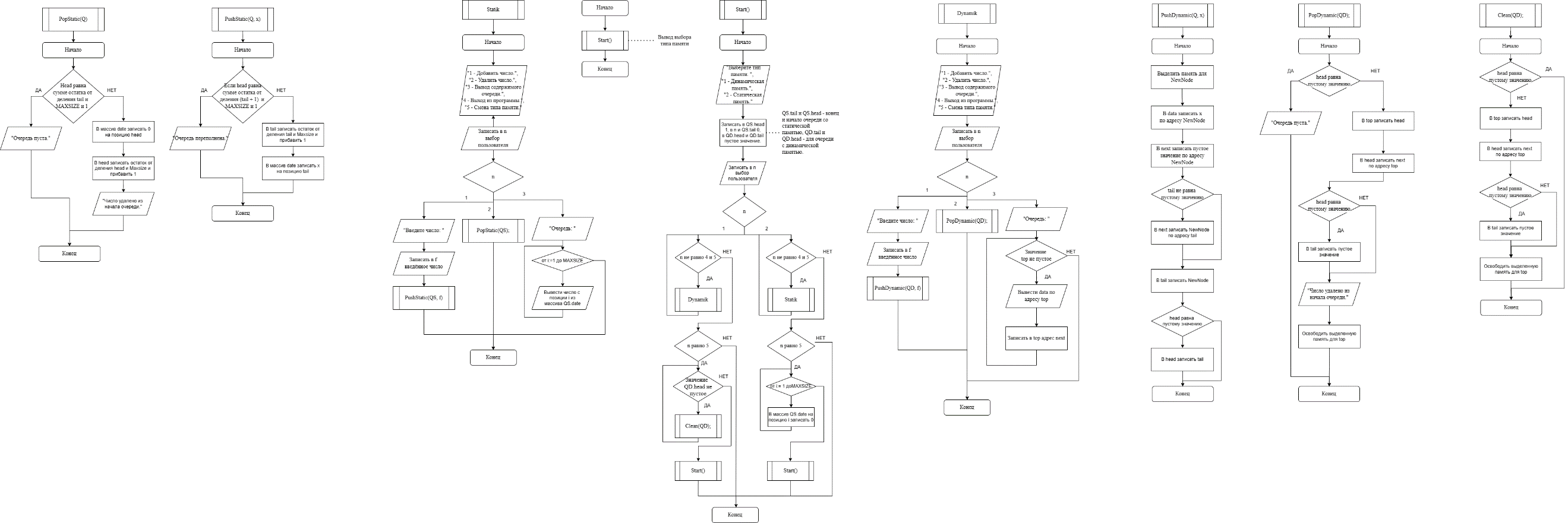
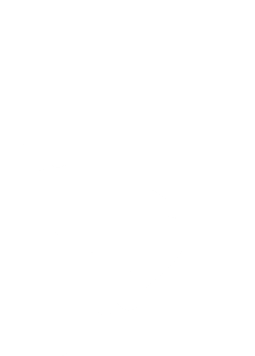




Рисунок 2 – Подпрограмма выбора типа данных

Рисунок 3 – Подпрограмма работы со статической очередью



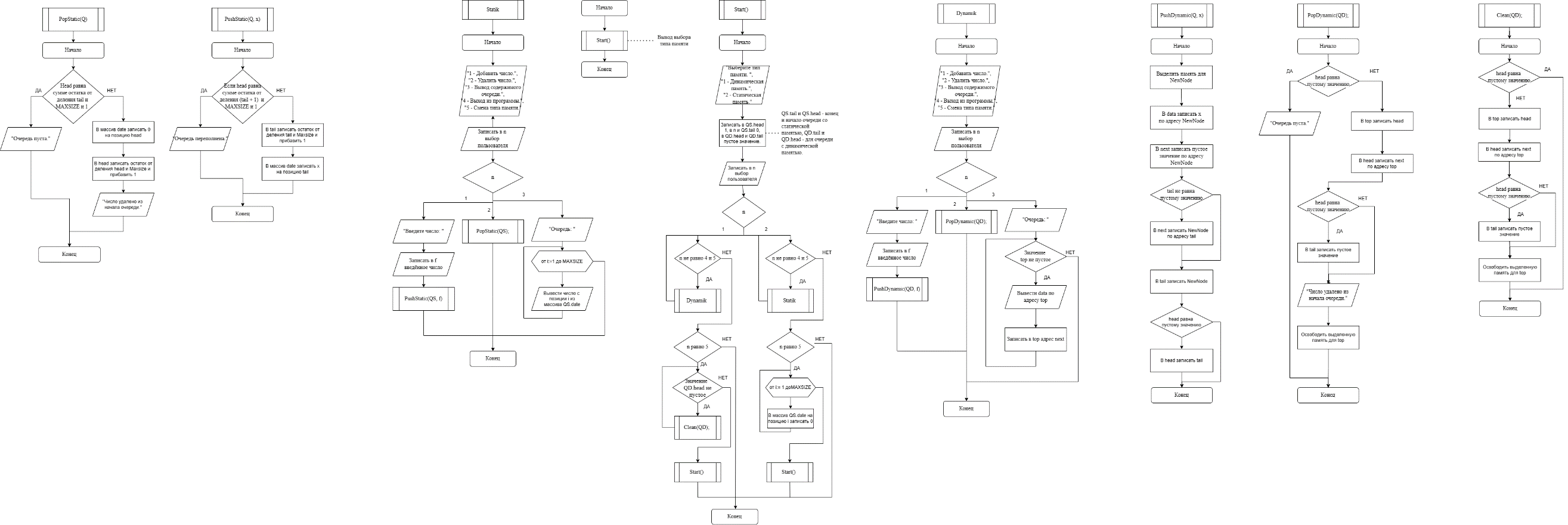
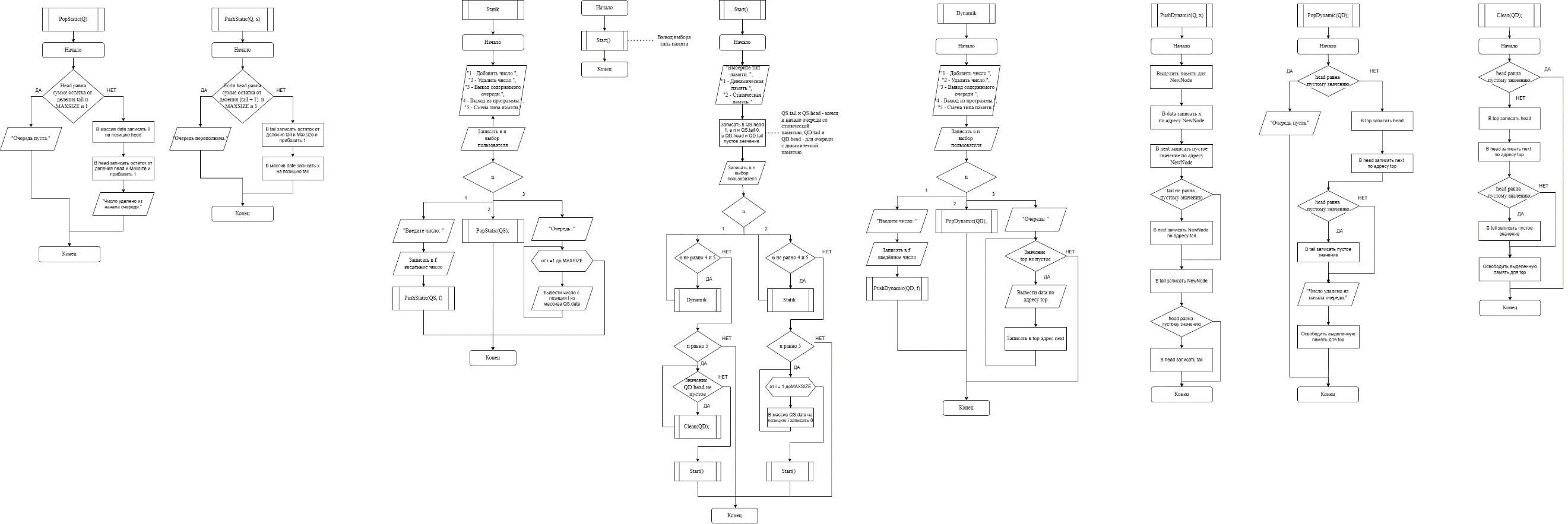
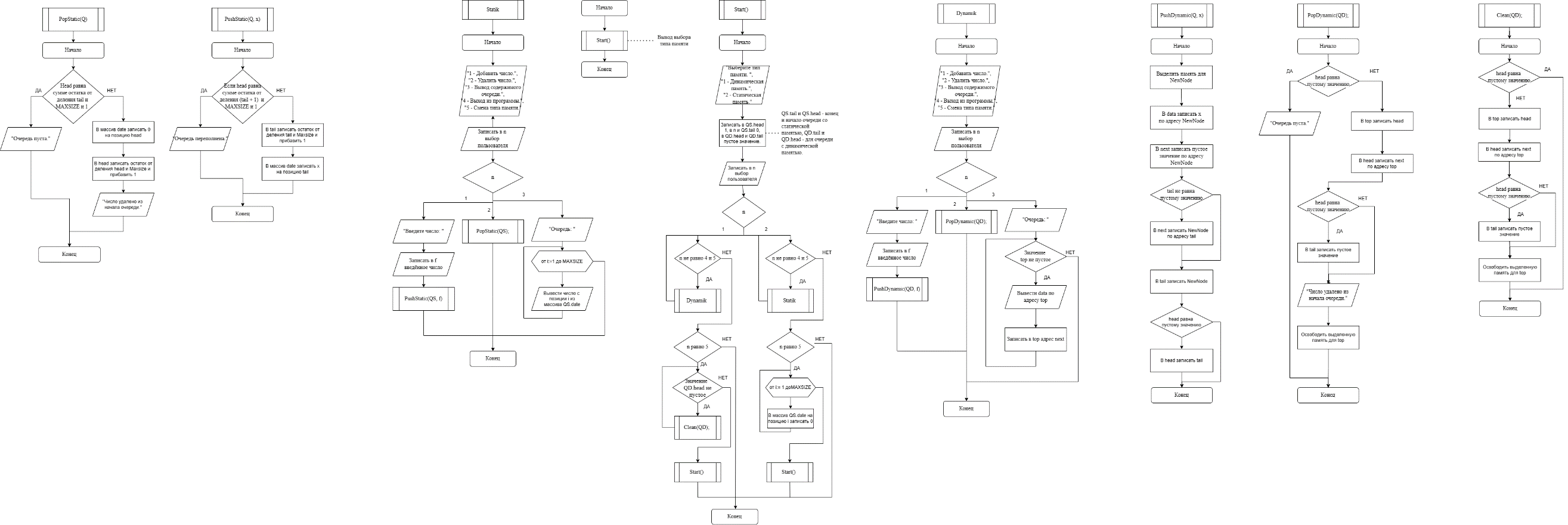
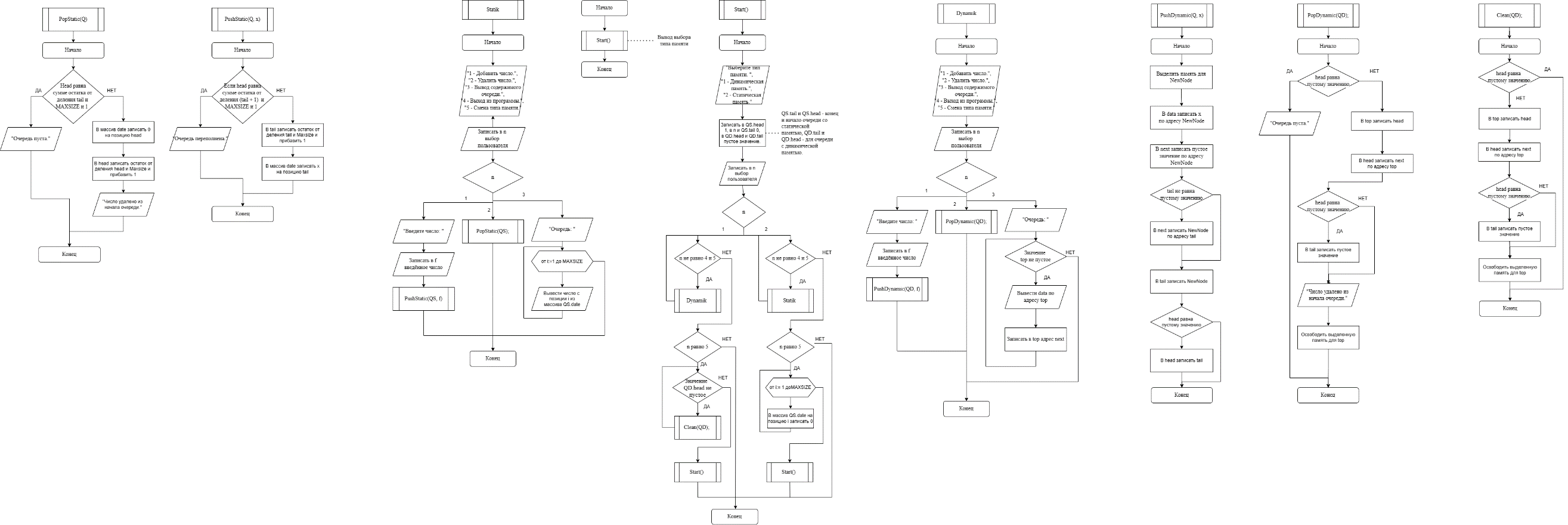


Рисунок 4 – Процедуры работы со статической очередью

Рисунок 5 – Подпрограмма работы с динамической очередью

Рисунок 6 – Процедуры для работы с динамической очередью

Рисунок 7 – Процедура для очистки динамической очереди при смене типа памяти

**5. Код программы**

**Unit** Stati;

**const** MAXSIZE = 10;

**type** Stat = **record**

date: **array**[1..MAXSIZE] **of** integer;

head, tail: integer;

**end**;

**procedure** PushStatic( **var** Q: Stat; x: integer);

**begin**

**if** Q.head = (Q.tail+1) **mod** MAXSIZE + 1 **then**

**begin**

writeln('Очередь переполнена.');

writeln('');

**Exit**;

**end**;

Q.tail := Q.tail **mod** MAXSIZE + 1;

Q.date[Q.tail] := x;

**end**;

**procedure** PopStatic ( **var** Q: Stat);

**begin**

**if** Q.head = Q.tail **mod** MAXSIZE + 1 **then begin**

writeln('Очередь пуста.');

writeln('');

**Exit**;

**end**;

Q.date[Q.head]:= 0;

Q.head := Q.head **mod** MAXSIZE + 1;

writeln('Число удалено из начала очереди.');

writeln('');

**end**;

**begin**

**end**.

**Unit** Dynamic;

**type** PNode = ^Node;

Node = **record**

data: integer;

next: PNode;

**end**;

Dyinamic = **record**

head, tail: PNode;

**end**;

**procedure** PushDynamic( **var** Q: Dyinamic; x: integer );

**var** NewNode: PNode;

**begin**

**New**(NewNode);

NewNode^.data := x;

NewNode^.next := nil;

**if** Q.tail <> nil **then**

Q.tail^.next := NewNode;

Q.tail := NewNode;

**if** Q.head = nil **then** Q.head := Q.tail;

**end**;

**procedure** PopDynamic ( **var** Q: Dyinamic );

**var** top: PNode;

**begin**

**if** Q.head = nil **then begin**

writeln('Очередь пуста.');

writeln('');

**Exit**;

**end**;

top := Q.head;

Q.head := top^.next;

**if** Q.head = nil **then** Q.tail := nil;

writeln('Число удалено из начала очереди.');

writeln('');

Dispose(top);

**end**;

**begin**

**end**.

**Program** Queue;

**uses** Crt, Stati, Dynamic;

**var** n: integer;

QS: Stat;

QD:Dyinamic;

**procedure** Statik;

**var** f, i: integer;

**begin**

writeln('1 - Добавить число.');

writeln('2 - Удалить число.');

writeln('3 - Вывод содержимого очереди.');

writeln('4 - Выход из программы.');

writeln('5 - Смена типа памяти.');

readln(n);

**case** n **of**

1:**begin** ClrScr; write('Введите число: '); readln(f); ClrScr; PushStatic(QS, f); **end**;

2:**begin** ClrScr; PopStatic(QS); writeln(''); **end**;

3:**begin** ClrScr; write('Очередь: '); **for** i:=1 **to** MAXSIZE **do** write(QS.date[i], ' '); writeln(''); writeln(''); **end**;

**end**;

**end**;

**procedure** Dynamik;

**var** f: integer;

top: PNode;

**begin**

writeln('1 - Добавить число.');

writeln('2 - Удалить число.');

writeln('3 - Вывод содержимого очереди.');

writeln('4 - Выход из программы.');

writeln('5 - Смена типа памяти.');

readln(n);

top := QD.head;

**case** n **of**

1:**begin** ClrScr; write('Введите число: '); readln(f); ClrScr; PushDynamic(QD, f); **end**;

2:**begin** ClrScr; PopDynamic(QD); **end**;

3:**begin** ClrScr; write('Очередь: '); **while** top <> nil **do begin** write(top^.data, ' '); top := top^.next; **end**; writeln(' '); writeln(''); **end**;

**end**;

**end**;

**procedure** Clean ( **var** Q: Dyinamic );

**var** top: PNode;

**begin**

**if** Q.head = nil **then begin**

**Exit**;

**end**;

top := Q.head;

Q.head := top^.next;

**if** Q.head = nil **then** Q.tail := nil;

Dispose(top);

**end**;

**procedure** Start();

**var** i:integer;

**begin**

ClrScr();

n:= 0;

writeln('Выберите тип памяти. ');

writeln('1 - Динамическая память.');

writeln('2 - Статическая память.');

QS.head:= 1;

QS.tail:= 0;

QD.head:= nil;

QD.tail:= nil;

readln(n);

**case** n **of**

1:**begin** ClrScr; **while** (n <> 4) **do** Dynamik; **if** n = 5 **then begin while** QD.head <> nil **do begin** Clean(QD); **end**; Start(); **end**; **end**; **end;**

2:**begin** ClrScr; **while** (n <> 4) **do** Statik; **if** n = 5 **then begin for** i:=1 **to** MAXSIZE **do** QS.date[i]:=0; Start(); **end**; **end**; **end;**

**end;**

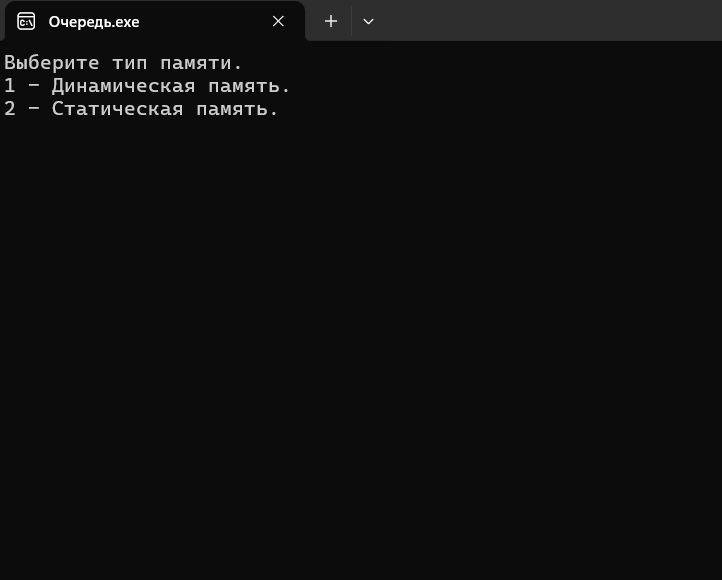
**end**;

**begin**

Start();

**end**.

**6. Результат выполнения программы**

****Рисунок 8 – Выбор типа памяти

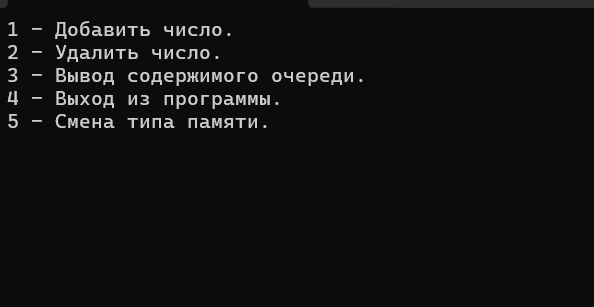
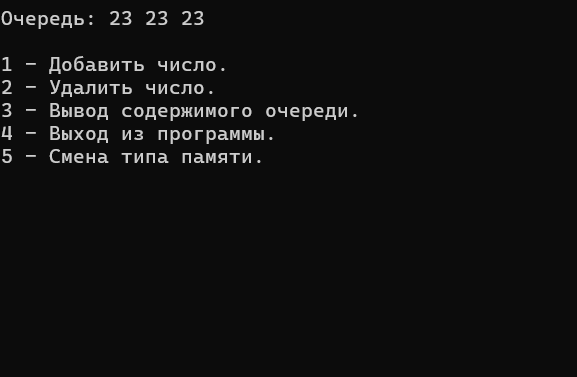
****Рисунок 9 – Выбор действия над очередью

Рисунок 10 – Вывод очереди

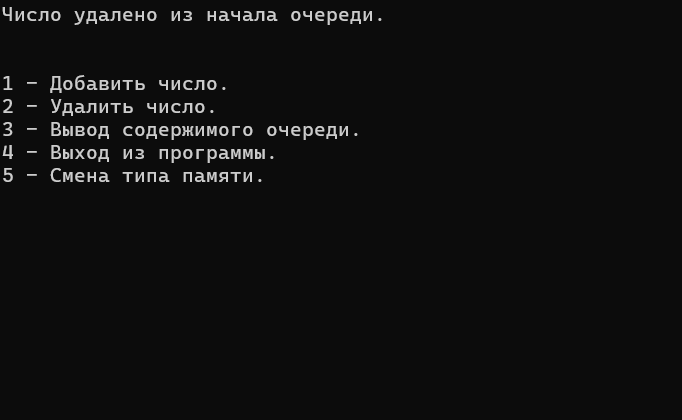
****

Рисунок 11 – Удаление числа из очереди

****Рисунок 12 – Добавление числа в очередь

**7. Вывод**

По итогам выполнения домашней контрольной работы я изучила принципы работы с базовыми структурами данных, получила навыков организации case-меню и реализовала «Очередь» с динамическим и статическим типами памяти.

Глобальных проблем не возникло во время реализации задания, только моя невнимательность, из-за чего не сразу появлялось понимание почему код не работает так как надо.

Полученные навыки пригодятся мне в дальнейших возможных заданиях.