|  |
| --- |
| Kyrgyz State Technical University |
| Algorithms and Data Structures |
| Laboratory work 5 |

|  |
| --- |
| Abakirov Nursultan SE(eng)1-16 |

**Лабораторная работа №5**

**Работа с очередью**

### Контрольные вопросы по теме "Очередь"

1. Какой метод доступа к элементам используется в списках? Опишите особенности этого метода.

2. Какие операции над элементами характерны для очередей?

3. Перечислите основные отличия очереди от массива.

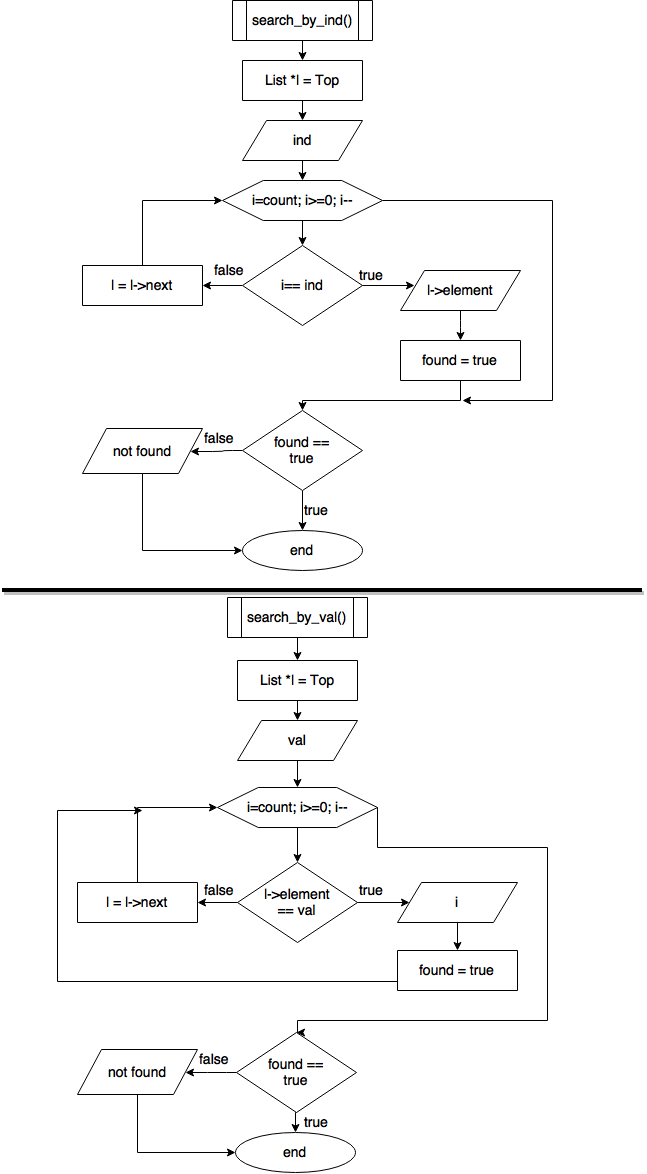
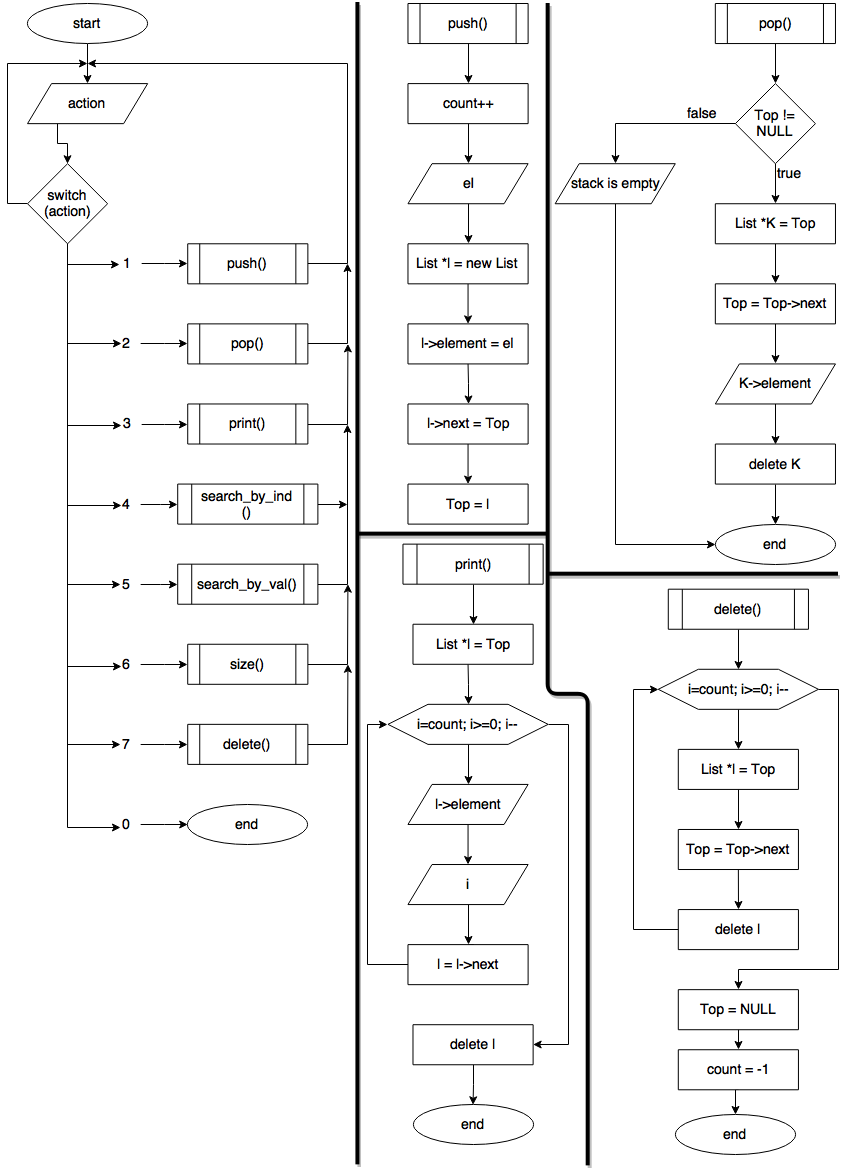
4. Для решения каких задач применяются очереди?

5. В чем преимущества циклической очереди?

6. К каким позициям очереди возможен доступ при записи и чтении информации?

**Задание**

1. Разработать программу для работы с очередью (алгоритмы создания, добавления, удаления элемента, поиска элемента по ключу, поиск позиции по заданному значению, нахождения количества элементов в очереди, очистка очереди)
2. Разработать программу создания кольцевой очереди
3. Разработать программу сцепления (конкатенации) двух очередей



#include <iostream>

using namespace std;

struct List {

int element;

List \*next;

int index;

};

List \*Top = NULL;

int I = -1;

int getINT() {

int k;

int i = 0;

do {

if (i > 0) {

cout << "only integer!\n";

}

i++;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

cin >> k;

} while (cin.fail());

return k;

}

void push() {

I++;

int el = getINT();

List \*l = new List;

l->element = el;

l->index = I;

l->next = Top;

Top = l;

cout << "pushed element: " << l->element << endl;

}

void pop() {

if (Top == NULL) {

cout << "stack is empty!\n";

return;

}

List \*del = Top;

Top = Top->next;

cout << "deleted element: " << del->element << endl;;

delete del;

I--;

}

void print() {

if (Top == NULL) {

cout << "stack is empty!\n";

return;

}

List \*l = Top;

for (int i = 0; i <= I; i++) {

cout << "\tindex: " << l->index << "\n\telement: " << l->element << "\n-----------\n";

l = l->next;

}

delete l;

}

void s\_val() {

if (Top == NULL) {

cout << "stack is empty!\n";

return;

}

int val = getINT();

List \*l = Top;

bool found = false;

for (int i = 0; i <= I; i++) {

if (l->element == val) {

cout << "index of found element: " << l->index << endl;

found = true;

}

l = l->next;

}

if (!found) {

cout << "element not found\n";

}

delete l;

}

void s\_ind() {

if (Top == NULL) {

cout << "stack is empty!\n";

return;

}

int ind = getINT();

if (ind <0 || ind >I) {

cout << "index out of range " << I << endl;

}

List \*l = Top;

for (int i = 0; i <= I; i++) {

if (l->index == ind) {

cout << "element of found index: " << l->element << endl;

}

l = l->next;

}

delete l;

}

void del() {

if (Top == NULL) {

cout << "stack is empty!\n";

return;

}

for (int i = 0; i <= I; i++) {

List \*l = Top;

Top = l->next;

delete l;

}

Top = NULL;

I = -1;

cout << "stack is empty!\n";

}

void size() {

if (Top == NULL) {

cout << "stack is empty!\n";

return;

}

cout << "stack size: " << I + 1 << endl;

}

int main() {

int k;

do {

cout << "\t1-push\n\t2-pop\n\t3-print\n\t4-search element by value\n\t5-search element by index\n\t6-delete stack\n\t7-size\n\t0-exit\n";

k = getINT();

switch (k)

{

case 1:

push();

cout << endl;

break;

case 2:

pop();

cout << endl;

break;

case 3:

print();

cout << endl;

break;

case 4:

s\_val();

cout << endl;

break;

case 5:

s\_ind();

cout << endl;

break;

case 6:

del();

cout << endl;

break;

case 7:

size();

cout << endl;

break;

default:

break;

}

} while (k != 0);

}