|  |
| --- |
| 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.

Доступ осуществляется по очереди, один за одним

2. Какие операции над элементами характерны для очередей?

Добавление в конец, удаление с начала

3. Перечислите основные отличия очереди от массива.

Очередь – это односвязный список, доступ с первого по последний

Массив – это несвязанный список, доступ по индексу

4. Для решения каких задач применяются очереди?

Диспетчер задач

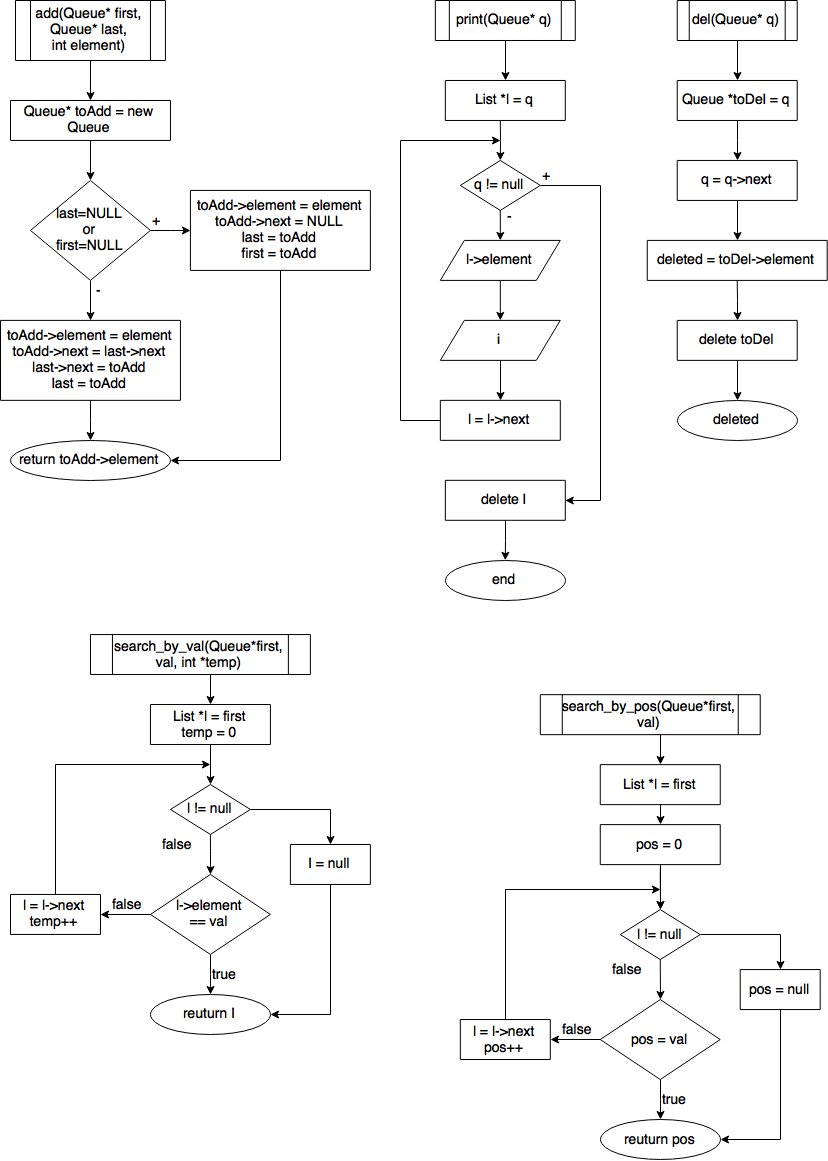
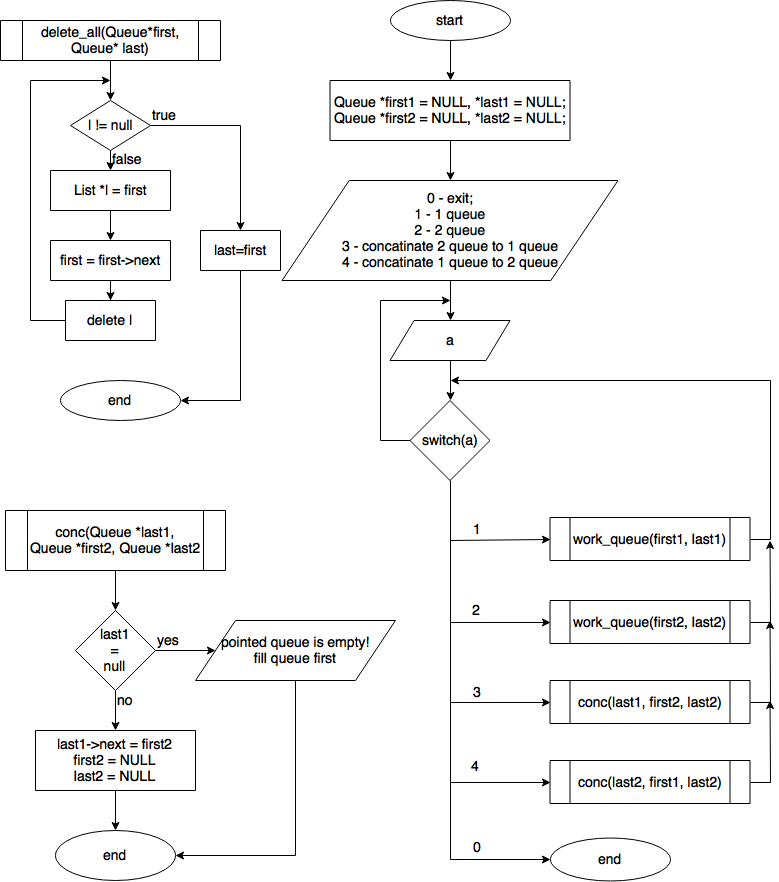
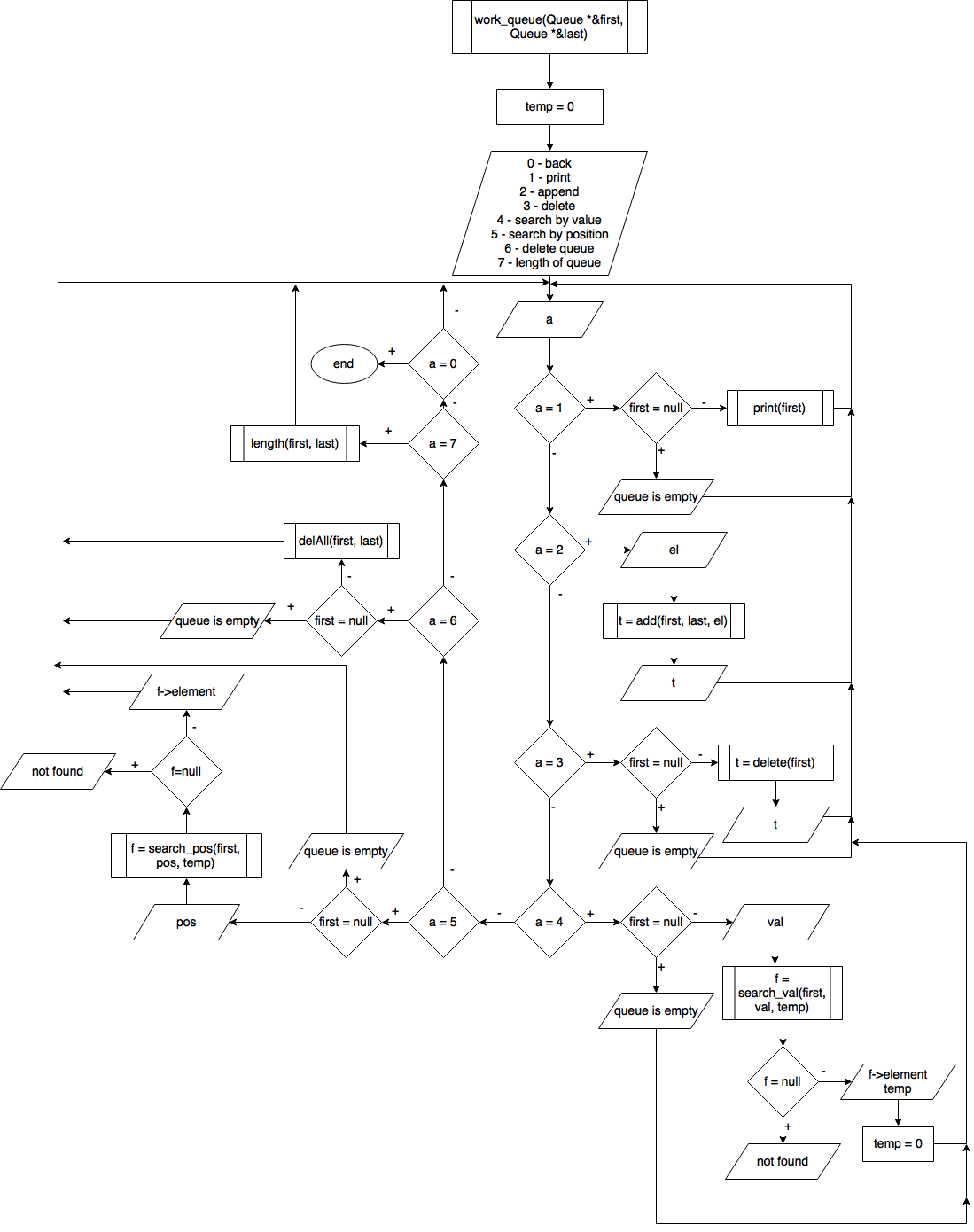
Буферизация ввода-вывода

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

Первому и последнему

**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 сцепления (конкатенации) двух очередей



#include <iostream>

#include <string>

using namespace std;

const string INIT = "queue is initialized\n";

const string EMPTY = "queue is empty\n";

struct Queue {

int element;

Queue \*next;};

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;}

int add(Queue \*&first, Queue \*&last, int element) {

Queue \*toAdd = new Queue;

if (last == NULL || first == NULL) {

toAdd->element = element;

toAdd->next = NULL;

last = toAdd;

first = toAdd;

return toAdd->element;}

toAdd->element = element;

toAdd->next = last->next;

last->next = toAdd;

last = toAdd;

return toAdd->element;}

int del(Queue \*&first) {

Queue \*toDel = first;

first = first->next;

int deleted = toDel->element;

delete toDel;

return deleted;}

void print(Queue \*first) {

Queue \*i = first;

cout << "\nqueue: ";

while (i != NULL) {

cout << i->element << " ";

i = i->next;}

i = NULL;

delete i;

cout << endl;}

Queue\* search\_val(Queue \*&first, int element, int &temp) {

Queue\*found = first;

temp = 0;

while (found != NULL) {

if (found->element == element) {

return found;

}

temp++;

found = found->next;

}

found = NULL;

return found;}

Queue \*search\_pos(Queue \*&first, int element, int &temp) {

Queue\* found = first;

temp = 0;

while (found != NULL) {

if (temp == element) {

return found;}

temp++;

found = found->next;}

found = NULL;

return found;}

void delAll(Queue \*&first, Queue\*&last) {

while (first != NULL) {

Queue\* del = first;

first = first->next;

delete del;}

last = first;}

int work\_queue(Queue \*&first, Queue \*&last) {

int temp = 0;

int action;

do {

cout << "\t0 - back\n";

cout << "\t1 - print\n";

cout << "\t2 - append\n";

cout << "\t3 - delete\n";

cout << "\t4 - search by value\n";

cout << "\t5 - search by position\n";

cout << "\t6 - delete queue\n";

cout << "\t7 - length of queue\n";

action = getINT();

if (action == 0) {

break;}

else if (action == 1) {

if (first == NULL) {

cout << EMPTY;}

else {

print(first);}}

else if (action == 2) {

int element = getINT();

int t = add(first, last, element);

cout << t << endl;}

else if (action == 3) {

if (first == NULL) {

cout << EMPTY;}

else {

int deleted = del(first);

cout << "deleted element " << deleted << endl;}}

else if (action == 4) {

if (first == NULL) {

cout << EMPTY;}

else {

int element = getINT();

Queue\* found = search\_val(first, element, temp);

if (found == NULL) {

cout << "element not found\n";}

else {

cout << "found element: " << found->element << " index: " << temp << endl;}

temp = 0;}}

else if (action == 5) {

if (first == NULL) {

cout << EMPTY;}

else {

int element = getINT();

if (element >= 0) {

Queue\*found = search\_pos(first, element, temp);

if (found == NULL) {

cout << "element not found\n";}

else {

cout << "found element: " << found->element << " index: " << temp << endl;}

temp = 0;}

else {

cout << "position must be positive " << endl;}}}

else if (action == 6) {

if (first != NULL) {

delAll(first, last);}

cout << EMPTY;}

else if (action == 7) {

Queue \*pv = first;

int count = 0;

while (pv != NULL) {

count++;

pv = pv->next; }

cout << "length: " << count << endl;

pv = NULL;

delete pv;

}

} while (action != 0);

return 0;

}

void conc(Queue \*&last1, Queue \*&first2, Queue \*&last2) {

if (last1 == NULL) {

cout << "pointed queue is empty!\nfill queue first\n";

return;

}

last1->next = first2;

first2 = NULL;

last2 = NULL;

}

int main() {

Queue \*first1 = NULL, \*last1 = NULL;

Queue \*first2 = NULL, \*last2 = NULL;

int action;

do {

cout << "\t0 - exit\n";

cout << "\t1 - 1 queue\n";

cout << "\t2 - 2 queue\n";

cout << "\t3 - concatinate 2 queue to 1 queue\n";

cout << "\t4 - concatinate 1 queue to 2 queue\n";

action = getINT();

if (action == 0) {

break;

}

else if (action == 1) {

work\_queue(first1, last1);

}

else if (action == 2) {

work\_queue(first2, last2);

}

else if (action == 3) {

conc(last1, first2, last2);

}

else if (action == 4) {

conc(last2, first1, last2);

}

} while (true);

}