|  |
| --- |
| Kyrgyz State Technical University |
| Algorithms and Data Structures |
| Laboratory work 3 |

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

1. Definition of list.

L**ist** or **sequence** is an [abstract data type](https://en.wikipedia.org/wiki/Abstract_data_type) that represents a countable number of ordered [values](https://en.wikipedia.org/wiki/Value_(computer_science)), where the same value may occur more than once

**Linked list** is a linear collection of data elements, in which linear order is not given by their physical placement in memory. Each pointing to the next node by means of a [pointer](https://en.wikipedia.org/wiki/Pointer_(computer_programming)). It is a [data structure](https://en.wikipedia.org/wiki/Data_structure) consisting of a group of [nodes](https://en.wikipedia.org/wiki/Node_(computer_science)) which together represent a [sequence](https://en.wikipedia.org/wiki/Sequence). Under the simplest form, each node is composed of data and a [reference](https://en.wikipedia.org/wiki/Reference_(computer_science)) (in other words, a *link*) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration.

1. How is list classified?
2. Definition of linear list.

**linear list** stores a collection of objects of a certain type, usually denoted as the **elements** of the list. The elements are ordered within the linear list in a linear sequence. Linear lists are usually simply denoted as **lists**.

1. What common operations are admitted to the elements of list?

* Adding an element to the list
* Delete by position and value
* Output of the list elements
* Delete list
* Search by position and value

1. How is access to elements of list made?

List is consistently accessible

1. What is static linear list?

Static linear list is usually realized by an array.

1. How is operation of adding an element to the list realized?
2. How is operation of deleting an element to the list realized?
3. How can we realize search of an element in the static linear list?
4. List the advantages and disadvantages of the static linear list.

Advantages:

Programming is very easy

It is very easy to move from a child to its parents

This representation is very easy to understand

Disadvantages:

Lot of memory area wasted.

Insertion and deletion of nodes needs lot of data movement

Design program

With functions

* Initialization of the list
* Adding an element to the list
* Deleting an element by position
* Deleting an element by set value
* Search an element by value in the list
* Search an element by position in the list
* Output of the list elements
* Deleting of the list.

**Requirements:**

Type of Input must be an integer.

**Design of diagrams.**

**Design** - The structural method – decomposition.

struct List {

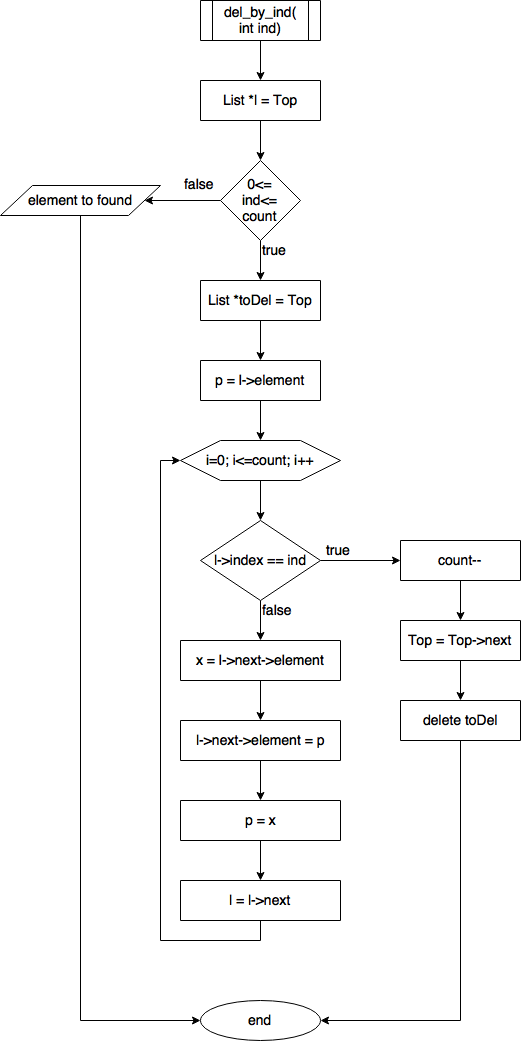
int element;

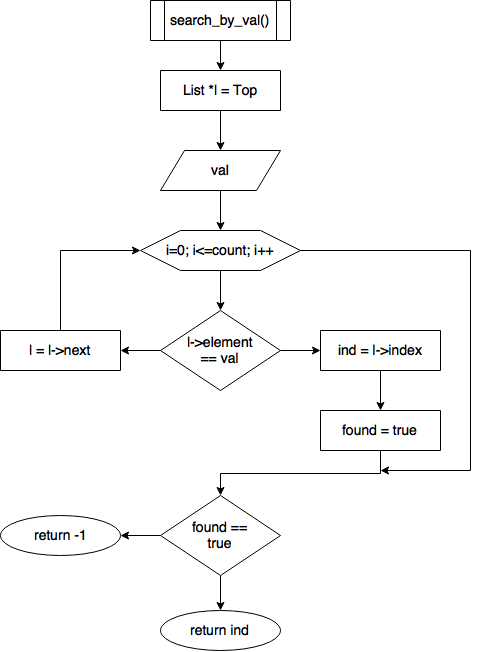
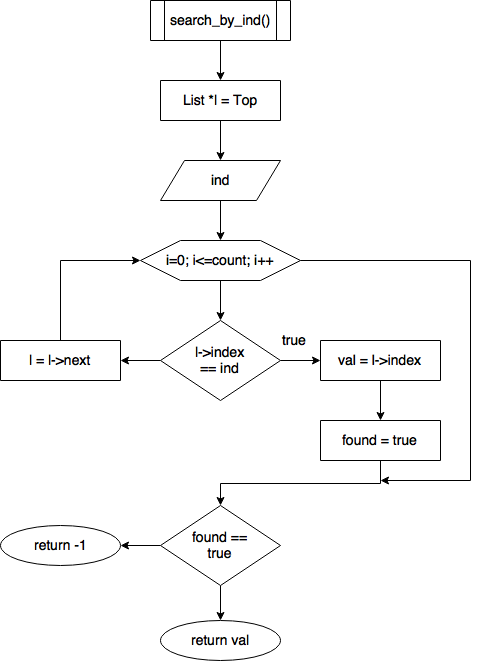
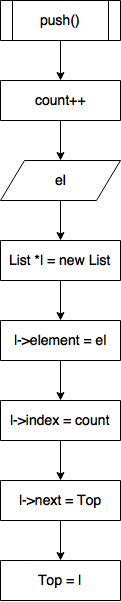
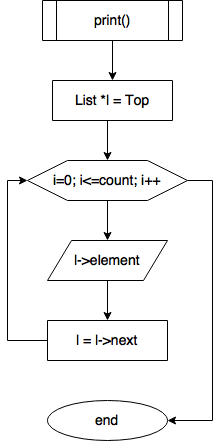
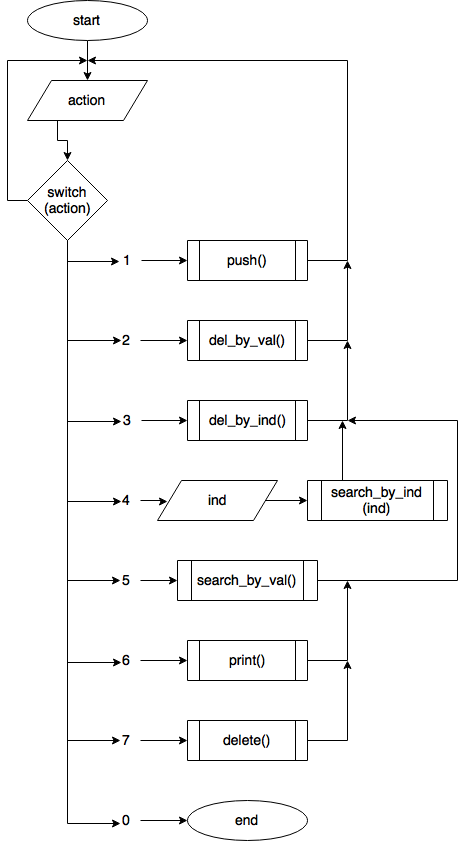
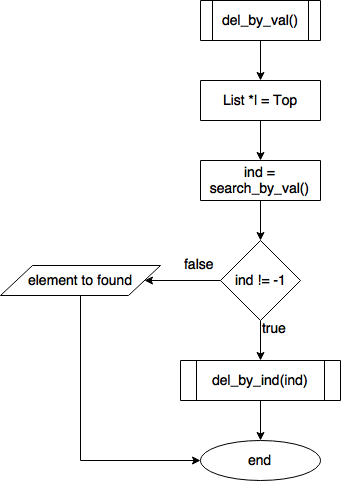
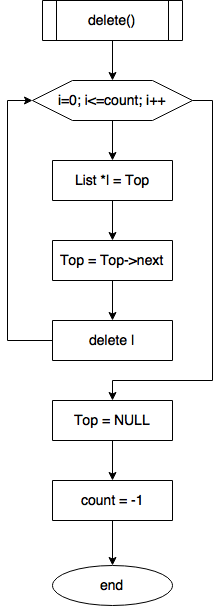
List \*next;

int index;};

List \*Top = NULL;

int count = -1;

****



#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++;

cout << "input element: ";

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 (I == -1) {

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

return;}

int ind;

cout << "delete by\t1-value\n\t\t2-index\n";

int k = getINT();

if (k != 1 && k != 2) {

cout << "incorrect choice!\n";

return;}

bool found = false;

List \*l = Top;

if (k == 1) {

cout << "input value: ";

int val = getINT();

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

if (l->element == val) {

ind = l->index;

found = true;

break;}

l = l->next;}}

else {

cout << "input index: ";

ind = getINT();

if (ind >= 0 && ind <= I) {

found = true;}

else {

found = false;}}

if (found) {

l = Top;

int x, p;

List \*toDel;

p = l->element;

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

if (l->index == ind) {

break;}

x = l->next->element;

l->next->element = p;

p = x;

l = l->next; }

I--;

toDel = Top;

Top = Top->next;

delete toDel;}

else {

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

return;}}

void print() {

if (I == -1) {

cout << "list 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 (I == -1) {

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

return;}

cout << "input value: ";

int val = getINT();

List \*l = Top;

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

if (l->element == val) {

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

l = l->next;}

delete l;}

void s\_ind() {

if (I == -1) {

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

return;}

cout << "input index: ";

int ind = getINT();

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

cout << "index out of range\n";

return;}

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 (I == -1) {

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

return;}

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

List \*l = Top;

Top = l->next;

delete l;}

Top = NULL;

I = -1;

cout << "list is clean\n";}

void size() {

if (I == -1) {

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

return;}

cout << "list 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 list\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);

}