

Министерство образования и науки Российской Федерации  
Федеральное агентство по образованию  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Лабораторная работа №6  
по курсу «Программирование»

**Реализация элементарных структур данных на основе динамической  
памяти**

Выполнил студент группы ИВТ-11 \_\_\_\_\_/Рзаев А. Э./  
Проверил преподаватель \_\_\_\_\_/Чистяков Г. А./

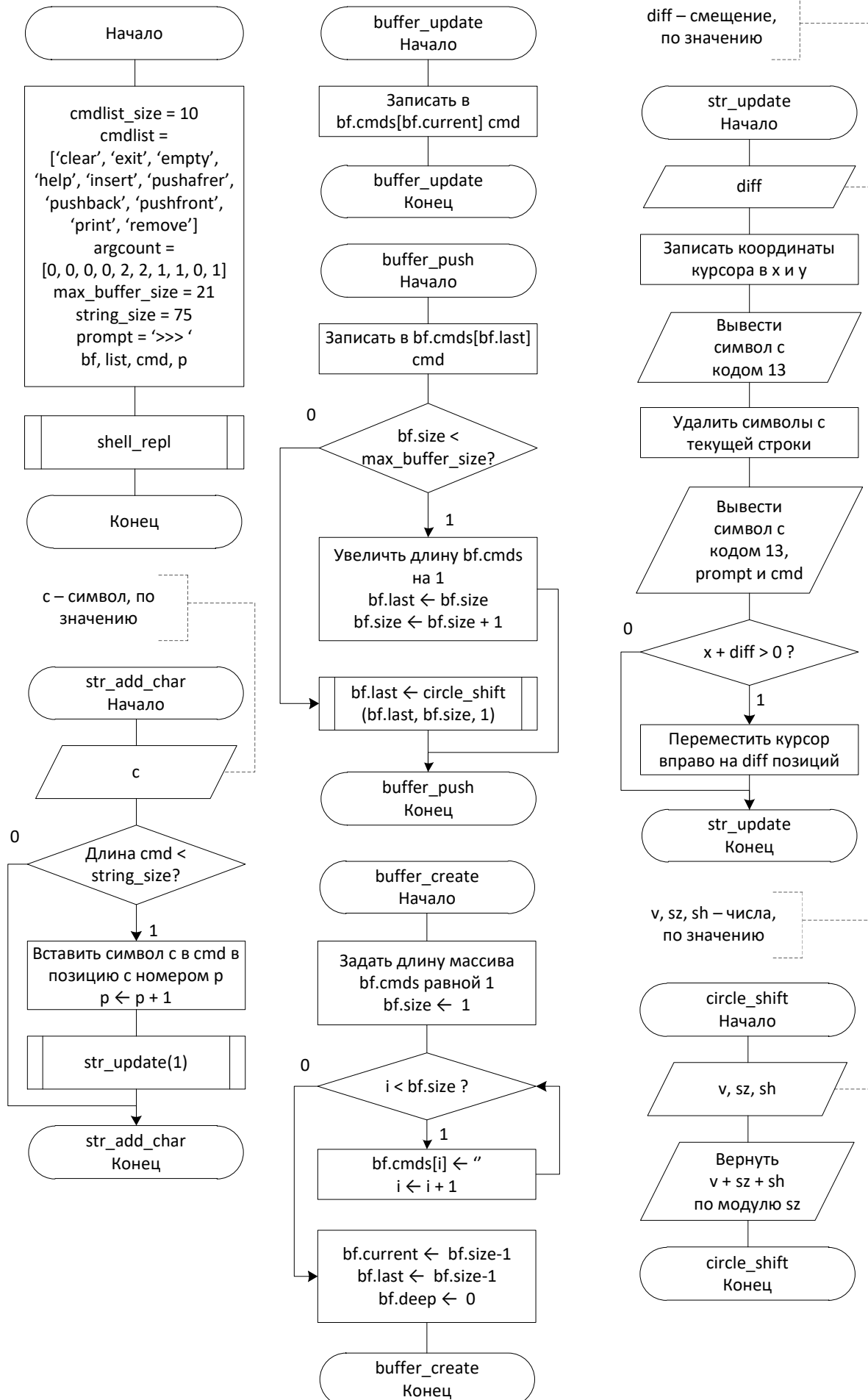
Киров 2016

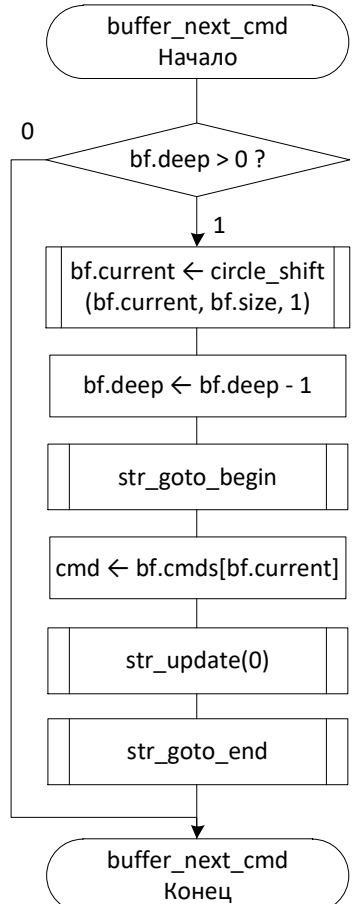
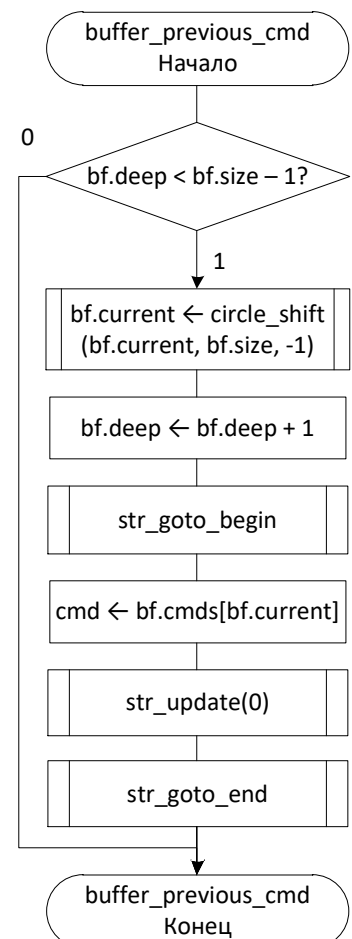
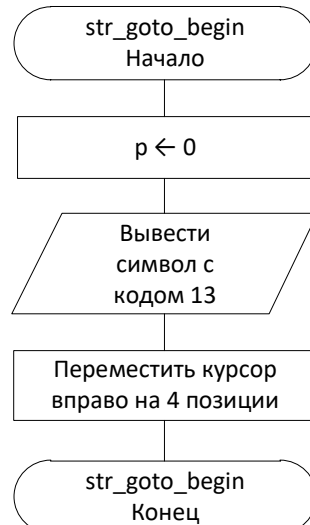
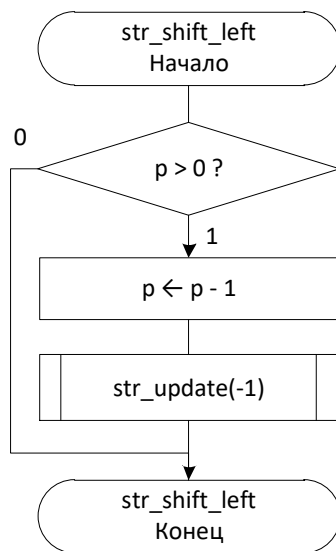
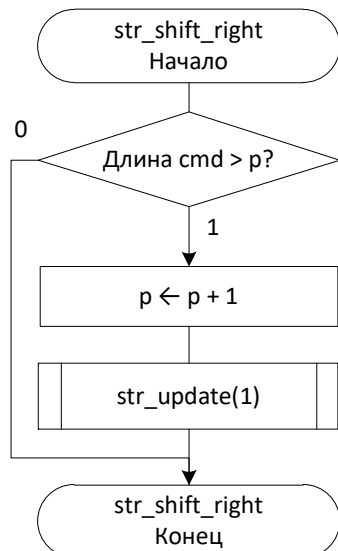
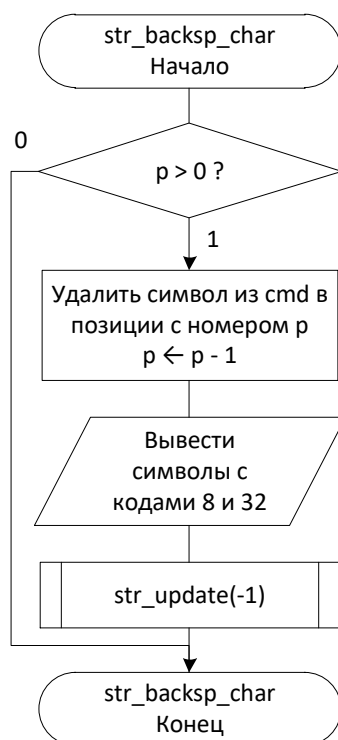
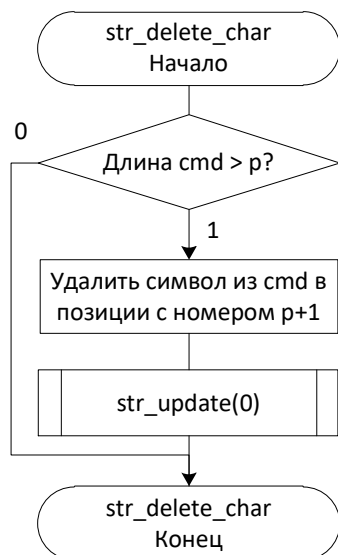
**Цель работы:** изучение структуры и принципов организации программных модулей, закрепление навыков работы с динамической памятью, получение базовых навыков организации работы в режиме командной строки.

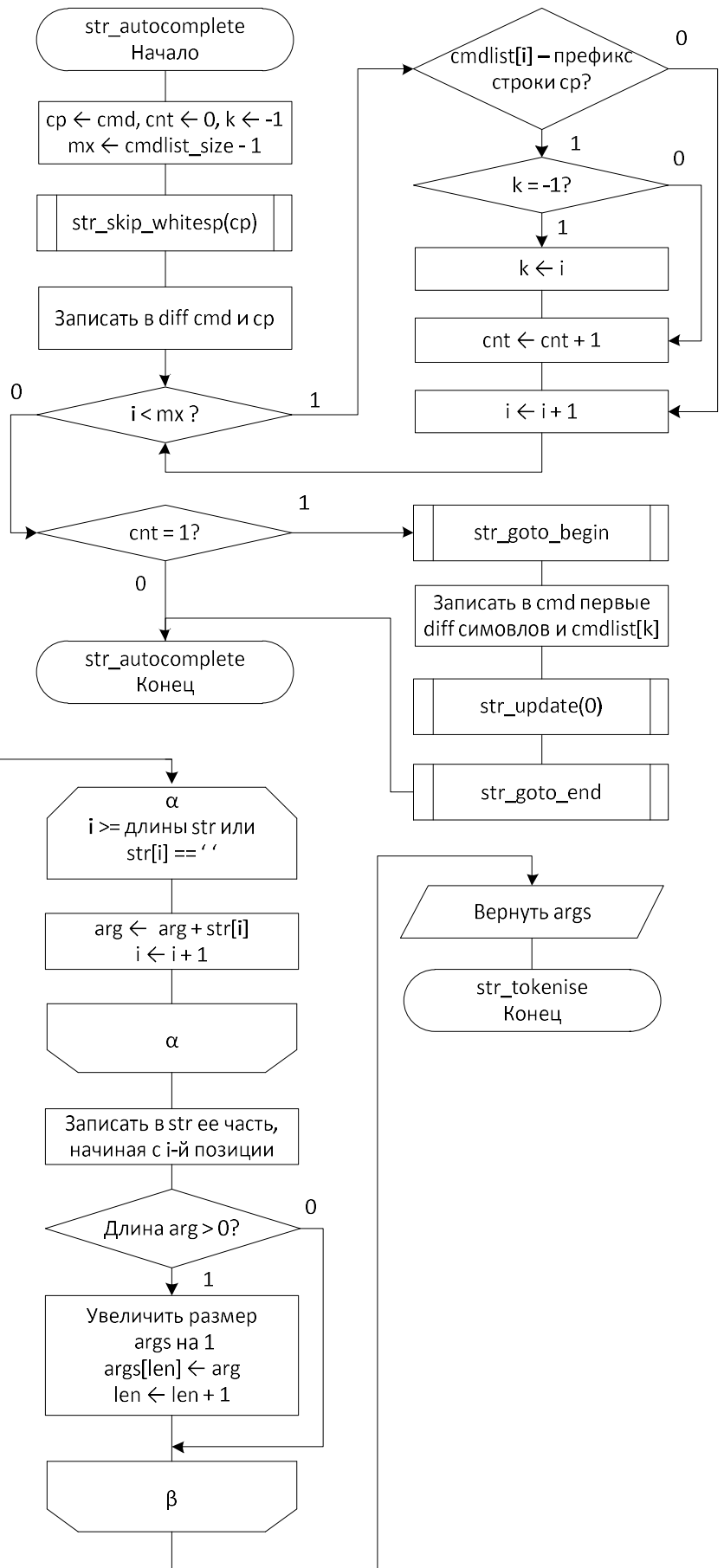
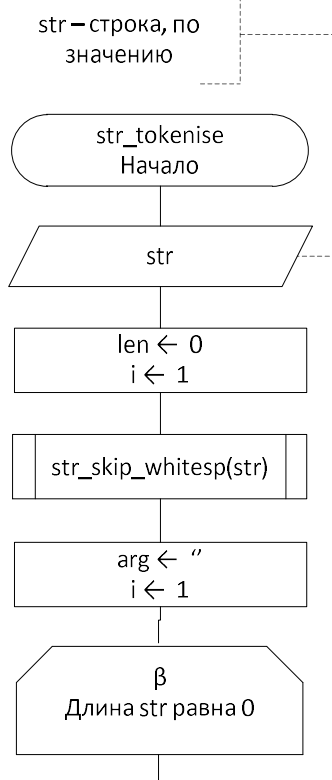
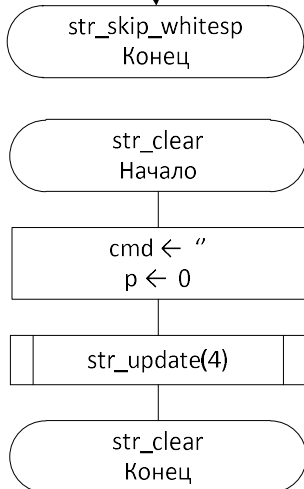
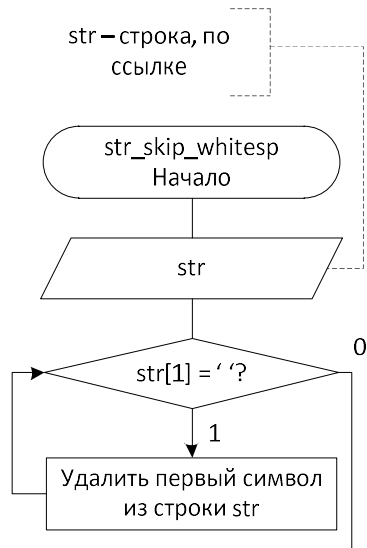
**Задание:**

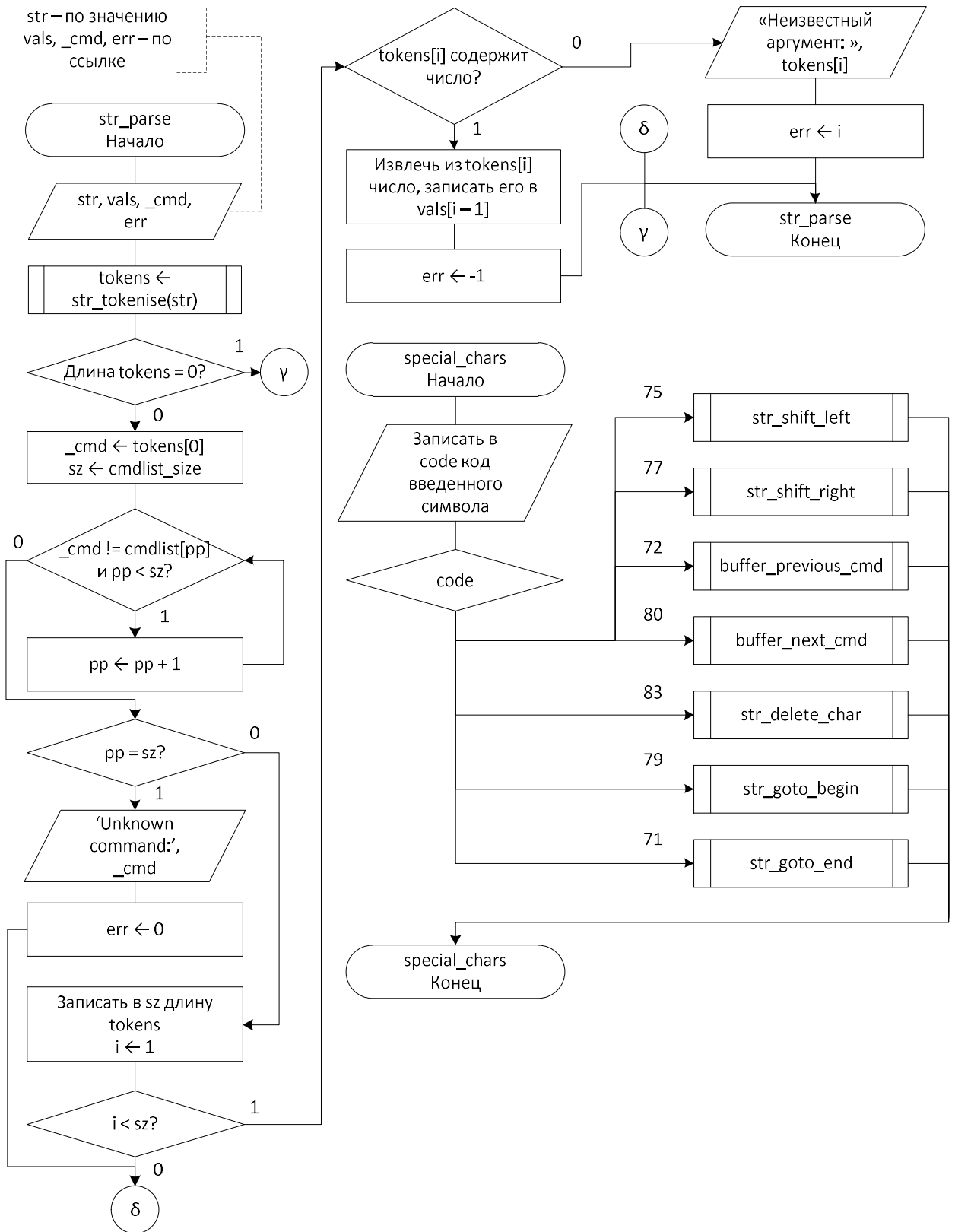
1. Написать программу для работы со структурой данных "Односвязный список".
2. Структура данных должна быть реализована на основе динамической памяти.
3. Структура данных (поля и методы) должна быть описана в отдельном модуле.
4. Работа со структурой должна осуществляться в режиме командной строки.

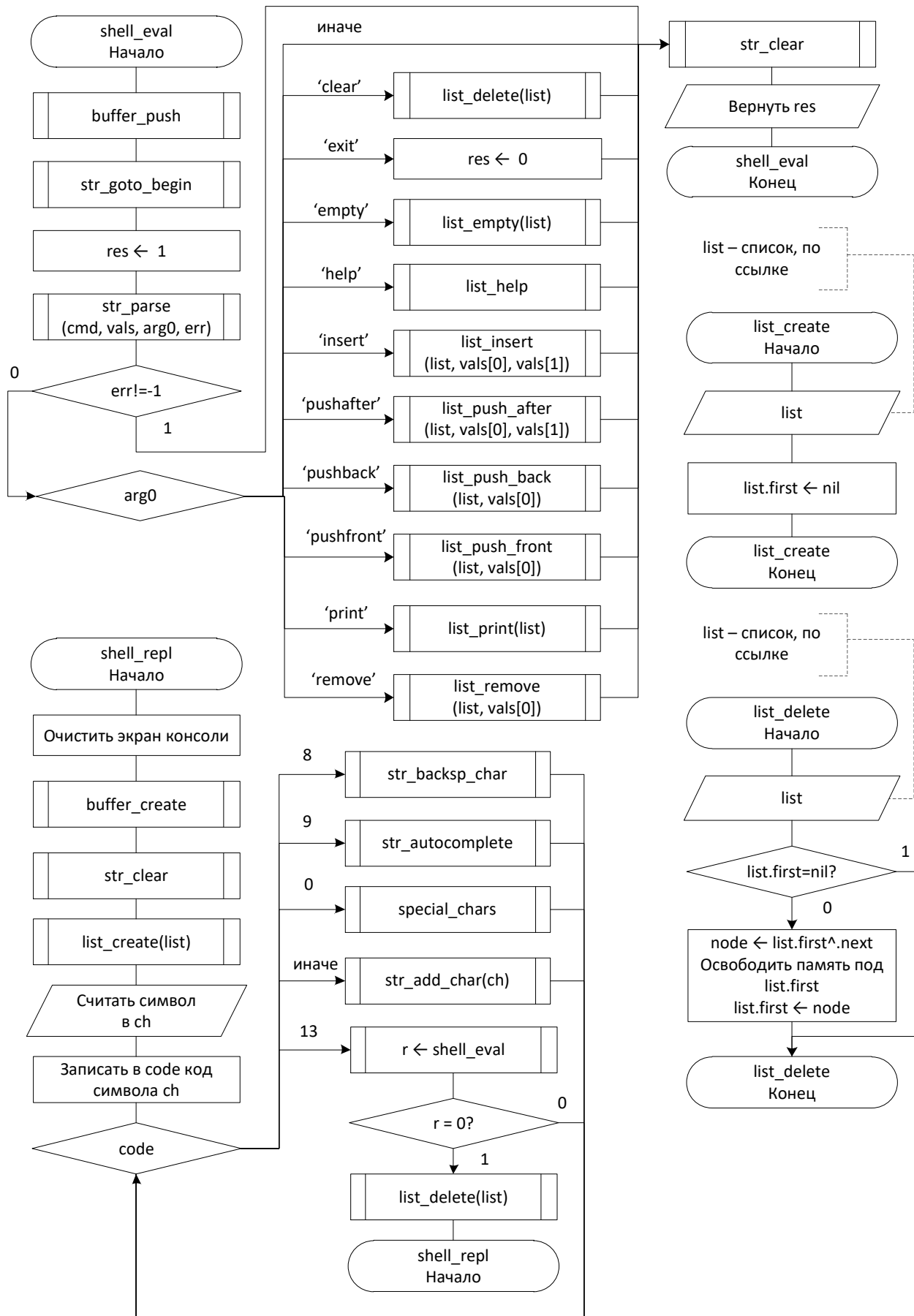
## Схема алгоритма:

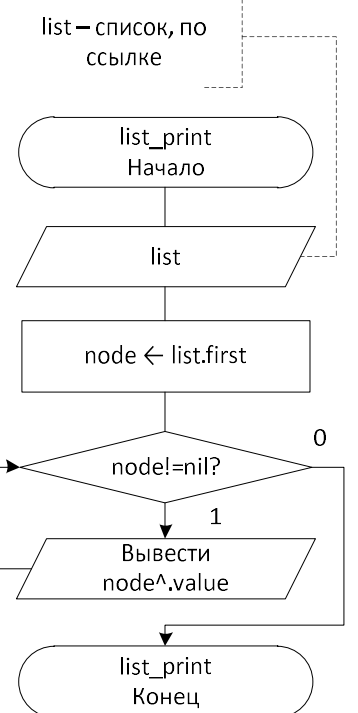
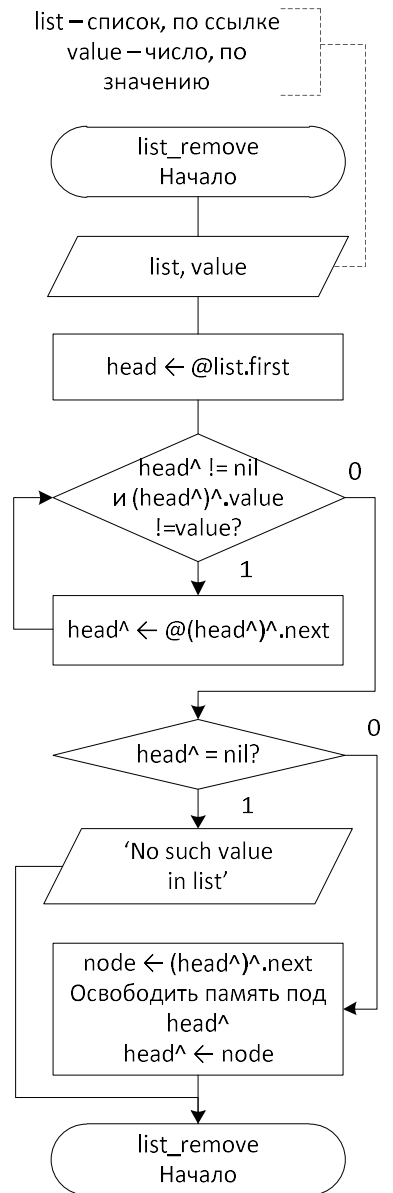
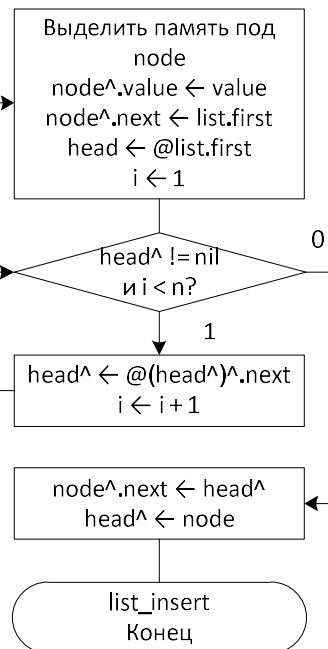
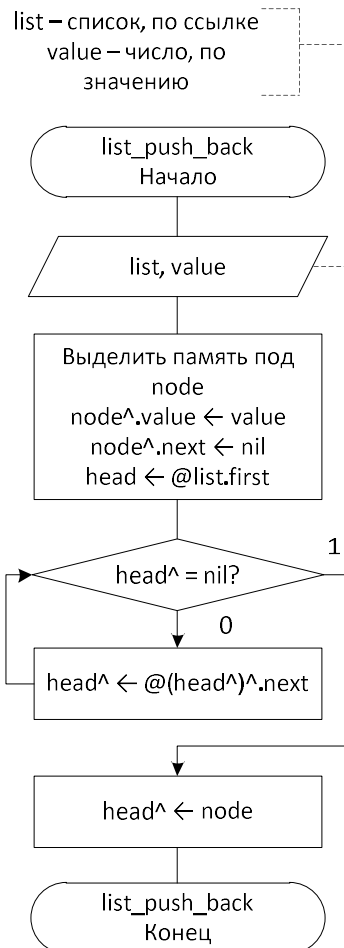
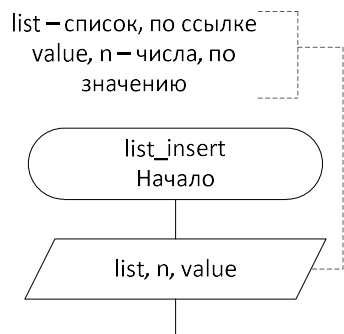
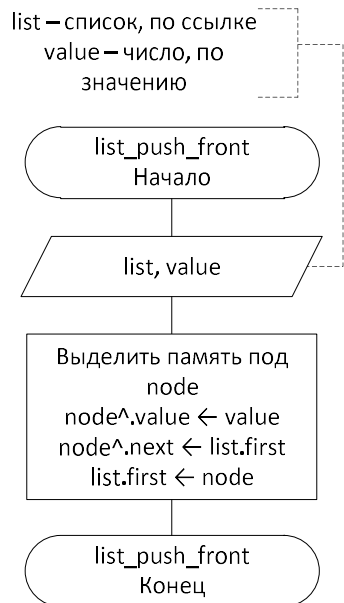
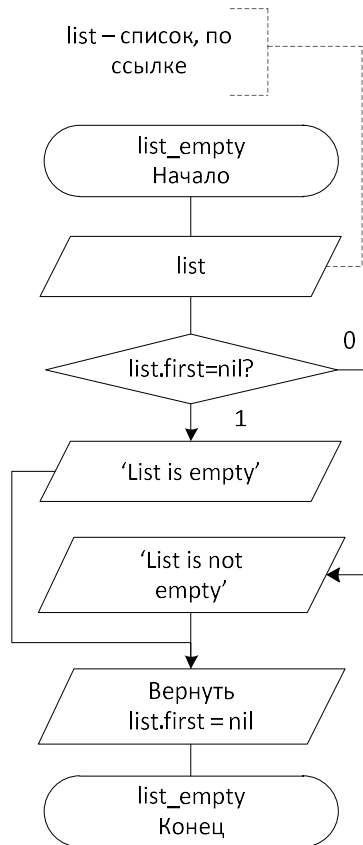




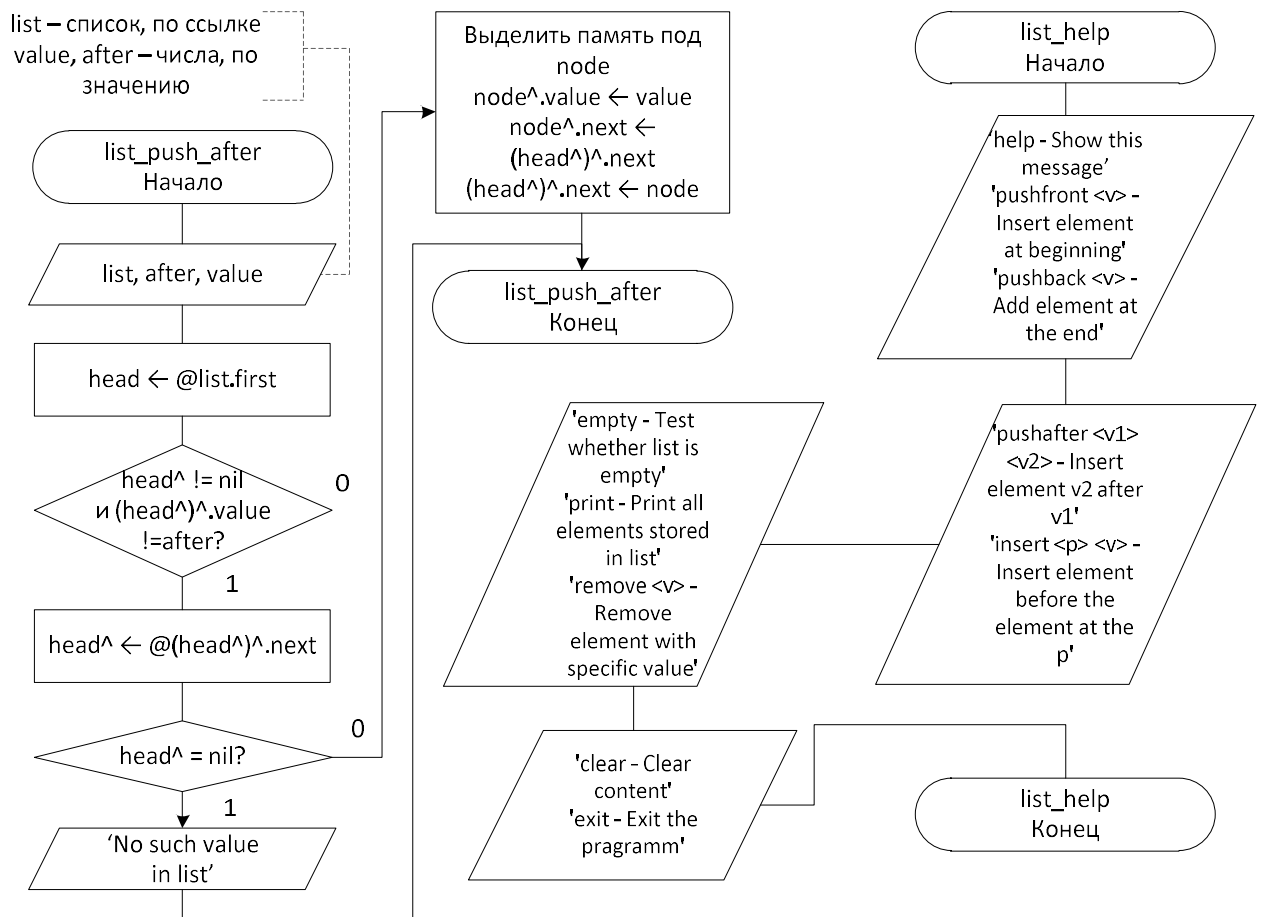












## Листинг кода: Основной модуль

```

uses Shell;

begin
    shell_repl;
end.
Модуль Shell
unit Shell;

interface

uses
    crt, sysutils, LinkedList;
const
    cmdlist_size = 10;
    cmdlist : array [0..(cmdlist_size - 1)] of String = (
        'clear',
        'exit',
        'empty',
        'help',
        'insert',
        'pushafter',
        'pushback',
        'pushfront',
        'print',
        'remove'
    );
  
```

```

    argcount : array [0..(cmdlist_size - 1)] of Integer = (
        0, 0, 0, 0, 2, 2, 1, 1, 0, 1
    );
    max_buffer_size = 21;
    string_size = 75;
    prompt : String = '>>> ';

type
    TBuffer = record
        cmds : array of String;
        size : Integer;
        current : Integer;
        deep : Integer;
        last : Integer;
    end;

    TArgs = array of String;
    TVals = array [0..1] of Integer;

var
    bf : TBuffer;
    list : TLinkedList;
    cmd : String;
    p : Integer;

procedure shell_repl();

implementation

procedure buffer_create;
var
    i : Integer;
begin
    setlength(bf.cmds, 1);
    bf.size := 1;
    for i := 0 to bf.size - 1 do
        bf.cmds[i] := '';
    end;
    bf.current := bf.size - 1;
    bf.last := bf.size - 1;
    bf.deep := 0;
end;

function circle_shift(v, sz, sh : Integer) : Integer;
begin
    circle_shift := (v + sz + sh) mod sz;
end;

procedure buffer_update;
begin
    bf.cmds[bf.current] := cmd;
end;

procedure buffer_push;
begin

```

```

    bf.cmds[bf.last] := cmd;
    if bf.size < max_buffer_size then
    begin
        setlength(bf.cmds, bf.size + 1);
        bf.last := bf.size;
        inc(bf.size);
    end
    else
        bf.last := circle_shift(bf.last, bf.size, 1);
    bf.deep := 0;
    bf.current := bf.last;
end;

procedure str_update(diff : Integer);
var
    x, y: Integer;
begin
    x := wherex;
    y := wherey;
    write(chr(13));
    clreol;
    write(chr(13), prompt, cmd);
    buffer_update;
    if x + diff > 0 then
        gotoxy(x + diff, y);
    end;
end;

procedure str_add_char(c : Char);
begin
    if Length(cmd) < string_size then
    begin
        cmd := copy(cmd, 1, p) + c + copy(cmd, p + 1, Length(cmd));
        inc(p);
        str_update(1);
    end;
end;

procedure str_delete_char;
begin
    if p < Length(cmd) then
    begin
        delete(cmd, p + 1, 1);
        str_update(0);
    end;
end;

procedure str_backsp_char;
begin
    if p > 0 then
    begin
        delete(cmd, p, 1);
        dec(p);
        write(chr(8), ' ');
        str_update(-1);
    end;
end;

```

```

        end;
end;

procedure str_shift_left;
begin
    if p > 0 then
        begin
            dec(p);
            str_update(-1);
        end;
    end;
end;

procedure str_shift_right;
begin
    if p < Length(cmd) then
        begin
            inc(p);
            str_update(1);
        end;
    end;
end;

procedure str_goto_begin;
var
    x, y : Integer;
begin
    p := 0;
    write(chr(13));
    x := wherex;
    y := wherey;
    gotoxy(x + Length(prompt), y);
end;

procedure str_goto_end;
var
    x, y, diff : Integer;
begin
    x := wherex;
    y := wherey;
    diff := Length(cmd) - p;
    p := Length(cmd);
    if x + diff > 0 then
        gotoxy(x + diff, y);
    end;
end;

procedure buffer_previous_cmd;
begin
    if bf.deep < bf.size - 1 then
        begin
            bf.current := circle_shift(bf.current, bf.size, -1);
            inc(bf.deep);
            str_goto_begin;
            cmd := bf.cmds[bf.current];
            str_update(0);
            str_goto_end;
        end;
    end;
end;

```

```

        end;
end;

procedure buffer_next_cmd;
begin
    if bf.deep > 0 then
    begin
        bf.current := circle_shift(bf.current, bf.size, 1);
        dec(bf.deep);
        str_goto_begin;
        cmd := bf.cmds[bf.current];
        str_update(0);
        str_goto_end;
    end;
end;

```

```

procedure str_skip_whitesp(var str : String);
var i : Integer;
begin
    i := 1;
    while i <= Length(str) do
        if str[i] = ' ' then
            delete(str, i, 1)
        else
            break;
        end;
    end;
end;

```

```

procedure str_autocomplete;
var
    i, diff, cnt, k : Integer;
    cp : String;
begin
    cp := cmd;
    str_skip_whitesp(cp);
    diff := Length(cmd) - Length(cp);
    cnt := 0; k := -1;
    for i := 0 to cmdlist_size - 1 do
        if pos(cp, cmdlist[i]) = 1 then
            begin
                if k = -1 then k := i;
                inc(cnt);
            end;
    end;
    if cnt = 1 then
    begin
        str_goto_begin;
        cmd := copy(cmd, 1, diff) + cmdlist[k];
        str_update(0);
        str_goto_end;
    end;
end;

```

```

function str_tokenize(str : String) : TArgs;
var
    i, len : Integer;

```

```

    arg : String;
    args : TArgs;
begin
    setlength(args, 0);
    len := 0;
    while true do
    begin
        str_skip_whitesp(str);
        arg := '';
        i := 1;
        while (i <= Length(str)) and (str[i] <> ' ') do
        begin
            arg := arg + str[i];
            inc(i);
        end;
        str := copy(str, i, Length(str));
        if Length(arg) > 0 then
        begin
            setlength(args, len + 1);
            args[len] := arg;
            inc(len);
        end;
        if Length(str) = 0 then
            break;
    end;
    str_tokenize := args;
end;

```

```

procedure str_parse(str : String; var vals : TVals; var _cmd : String;
var err : Integer);
var
    tokens : TArgs;
    i, j, pp : Integer;
begin
    tokens := str_tokenize(str);
    if Length(tokens) = 0 then exit;
    _cmd := tokens[0];
    pp := 0;
    while pp < cmdlist_size do
        if _cmd = cmdlist[pp] then
            break
        else
            inc(pp);
    if pp = cmdlist_size then
    begin
        writeln('Unrecognised command: ', _cmd);
        err := 0;
        exit;
    end;

    for i := 1 to Length(tokens) - 1 do
    begin
        val(tokens[i], vals[i - 1], j);
        if j <> 0 then

```

```

        begin
            writeln('Unrecognised arg: ', tokens[i]);
            err := i;
            exit;
        end;
    end;
end;
if argcount[pp] <> Length(tokens) - 1 then
begin
    writeln('Invalid count of args');
    err := 0;
    exit;
end;
err := -1;
end;

procedure str_clear;
begin
    cmd := '';
    p := 0;
    str_update(Length(prompt));
end;

procedure special_chars;
var
    code : Integer;
begin
    code := ord(readkey);
    case code of
        75 : //left
            str_shift_left;
        77 : //right
            str_shift_right;
        72 : //up
            buffer_previous_cmd;
        80 : //down
            buffer_next_cmd;
        83 : //delete
            str_delete_char;
        79 : //end
            str_goto_end;
        71 : //home
            str_goto_begin;
    end;
end;

function shell_eval : Integer;
var
    err : Integer;
    arg0 : String;
    vals : TVals;
begin
    buffer_push;
    str_goto_begin;
    shell_eval := 1;

```

```

writeln;
str_parse(cmd, vals, arg0, err);
if err <> -1 then
begin
    str_clear;
    exit;
end;

case arg0 of
'clear' :
    list_delete(list);
'exit' :
    shell_eval := 0;
'empty' :
    list_empty(list);
'help' :
    list_help;
'insert' :
    list_insert(list, vals[0], vals[1]);
'pushafter' :
    list_push_after(list, vals[0], vals[1]);
'pushback' :
    list_push_back(list, vals[0]);
'pushfront' :
    list_push_front(list, vals[0]);
'print' :
    list_print(list);
'remove' :
    list_remove(list, vals[0]);
end;
str_clear;
end;

procedure shell_repl();
var
    ch : Char;
    code : Integer;
begin
    clrscr;
    buffer_create;
    str_clear;
    list_create(list);
    while true do
    begin
        ch := readkey;
        code := ord(ch);

        case code of
            13, 10 : //enter
                if shell_eval = 0 then
                    break;
            8 : //backspace
                str_backsp_char;
            9 : //tab, autocomplete

```



```

        str_autocomplete;
    0 :
        special_chars
    else //alphanum
        str_add_char(ch);
    end;
end;
list_delete(list);
end;

end.

```

### **Модуль LinkedList**

```
unit LinkedList;
```

```
interface
```

```
type
```

```
    TValue = Integer;
```

```
    TNode = record
```

```
        value : TValue;
```

```
        next : ^TNode;
```

```
    end;
```

```
    TLinkedList = record
```

```
        first : ^TNode;
```

```
    end;
```

```
procedure list_create(var list : TLinkedList);
```

```
procedure list_delete(var list : TLinkedList);
```

```
function list_empty(list : TLinkedList) : Boolean;
```

```
procedure list_push_front(var list : TLinkedList; value : TValue);
```

```
procedure list_push_back(var list : TLinkedList; value : TValue);
```

```
procedure list_insert(var list : TLinkedList; n : Integer; value :
TValue);
```

```
procedure list_push_after(var list : TLinkedList; after, value :
TValue);
```

```
procedure list_remove(var list : TLinkedList; value : TValue);
```

```
procedure list_print(list : TLinkedList);
```

```
procedure list_help;
```

```
implementation
```

```
procedure list_create(var list : TLinkedList);
```

```

begin
    list.first := nil;
end;

procedure list_delete(var list : TLinkedList);
var
    node : ^TNode;
begin
    while (list.first <> nil) do
        begin
            node := list.first^.next;
            dispose(list.first);
            list.first := node;
        end;
    end;
end;

function list_empty(list : TLinkedList) : Boolean;
begin
    if list.first = nil then
        writeln('List is empty')
    else
        writeln('List is not empty');
    list_empty := list.first <> nil;
end;

procedure list_push_front(var list : TLinkedList; value : TValue);
var node : ^TNode;
begin
    new(node);
    node^.value := value;
    node^.next := list.first;
    list.first := node;
end;

procedure list_push_back(var list : TLinkedList; value : TValue);
type
    TPtr = ^TNode;
var
    node : ^TNode;
    head : ^TPtr;
begin
    new(node);
    node^.value := value;
    node^.next := nil;
    head := @list.first;
    while head^ <> nil do
        begin
            head := @(head^).next;
        end;
    head^ := node;
end;

procedure list_insert(var list : TLinkedList; n : Integer; value :
TValue);

```

```

type
    TPtr = ^TNode;
var
    node : ^TNode;
    head : ^TPtr;
    i : Integer;
begin
    i := 1;
    new(node);
    node^.value := value;
    node^.next := nil;
    head := @list.first;
    while ( (head^ <> nil) and (i < n) ) do
        begin
            head := @(head^)^.next;
            inc(i);
        end;
    node^.next := head^;
    head^ := node;
end;

procedure list_remove(var list : TLinkedList; value : TValue);
type
    TPtr = ^TNode;
var
    node : ^TNode;
    head : ^TPtr;
begin
    head := @list.first;
    while ( (head^ <> nil) and ((head^)^.value <> value) ) do
        head := @(head^)^.next;

        if (head^ = nil) then
            writeln('No such value in list')
        else
            begin
                node := (head^)^.next;
                dispose(head^);
                head^ := node;
            end;
    end;
end;

procedure list_push_after(var list : TLinkedList; after, value :
TValue);
type
    TPtr = ^TNode;
var
    node : ^TNode;
    head : ^TPtr;
begin
    head := @list.first;
    while ( (head^ <> nil) and ((head^)^.value <> after) ) do
        head := @(head^)^.next;

```

```

    if (head^ <> nil) then
    begin
        new(node);
        node^.value := value;
        node^.next := (head^)^.next;
        (head^)^.next := node;
    end
    else
        writeln('No such value in list');
end;

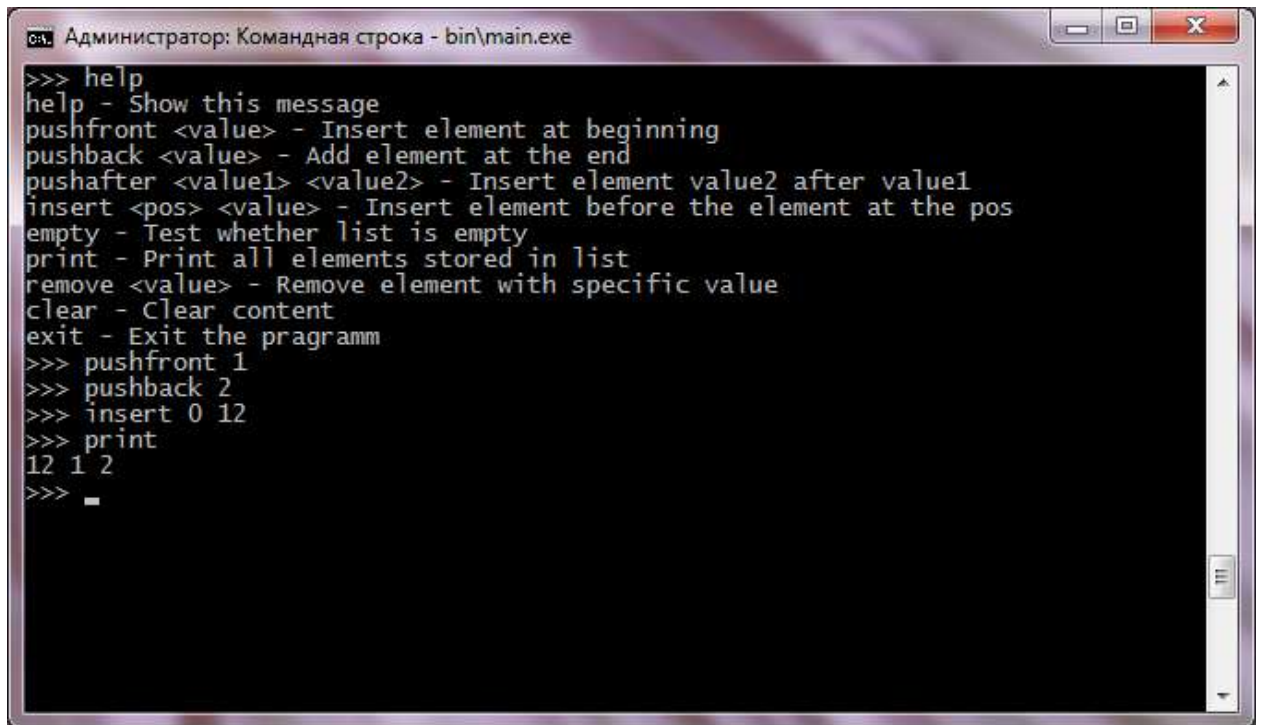
procedure list_print(list : TLinkedList);
var
    node : ^TNode;
begin
    node := list.first;
    while (node <> nil) do
    begin
        write(node^.value, ' ');
        node := node^.next;
    end;
    writeln;
end;

procedure list_help;
begin
    writeln('help - Show this message');
    writeln('pushfront <value> - Insert element at beginning');
    writeln('pushback <value> - Add element at the end');
    writeln('pushafter <value1> <value2> - Insert element value2
after value1');
    writeln('insert <pos> <value> - Insert element before the element
at the pos');
    writeln('empty - Test whether list is empty');
    writeln('print - Print all elements stored in list');
    writeln('remove <value> - Remove element with specific value');
    writeln('clear - Clear content');
    writeln('exit - Exit the programm');
end;

end.

```

## Экранная форма:



```
>>> help
help - Show this message
pushfront <value> - Insert element at beginning
pushback <value> - Add element at the end
pushafter <value1> <value2> - Insert element value2 after value1
insert <pos> <value> - Insert element before the element at the pos
empty - Test whether list is empty
print - Print all elements stored in list
remove <value> - Remove element with specific value
clear - Clear content
exit - Exit the program
>>> pushfront 1
>>> pushback 2
>>> insert 0 12
>>> print
12 1 2
>>> _
```

**Вывод:** в ходе лабораторной работы была изучена структура и принципы организации программных модулей, закреплены навыки работы с динамической памятью, получены базовые навыки организации работы в режиме командной строки.