МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение

Высшего профессионального образования

**«Вятский государственный университет»**

**(ФГБОУ ВПО «ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Реализация элементарных структур данных на основе динамической памяти

Отчет по лабораторной работе №6 дисциплины

«Программирование»

Выполнил студент группы ИВТб-11\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Бурдукова А.А./

Проверил преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Чистяков Г.А./

Киров 2019

**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:**

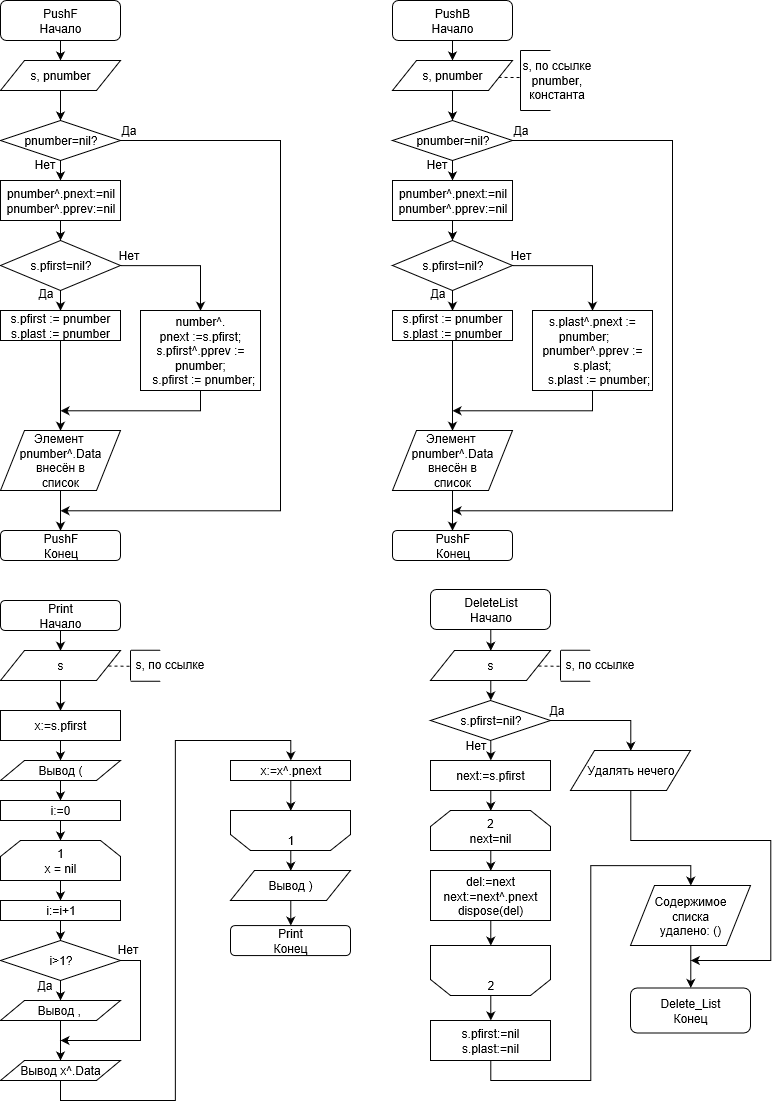
**1.Написать программу для работы со структурой данных “Двусвязный список”.**

**2.Структура данных должна быть реализована на основе динамической памяти.**

**3.Структура данных (поля и методы) должна быть описана в отдельном модуле.**

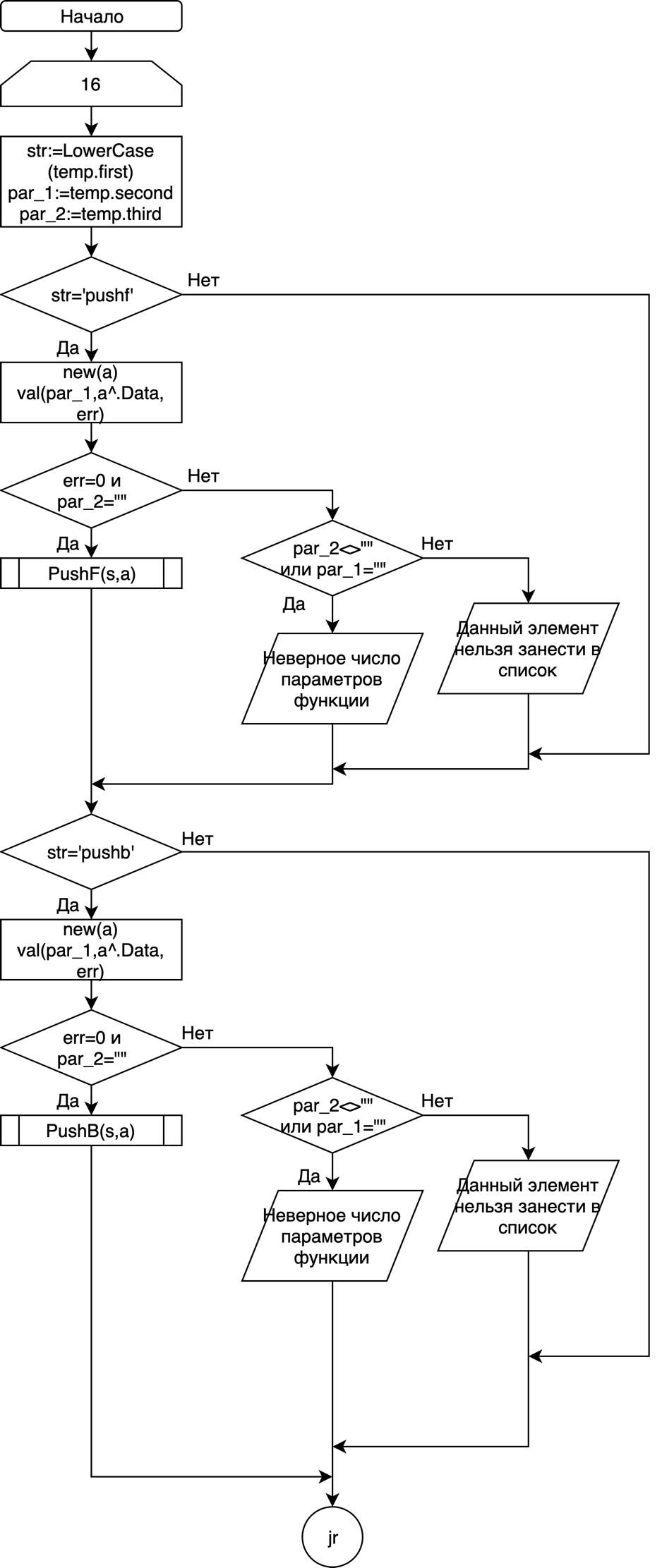
**4.Работа со структурой должна осуществляться в режиме командной строки (с реализацией автодополнения и истории команд). Предусмотреть наглядную визуализацию содержимого структуры**

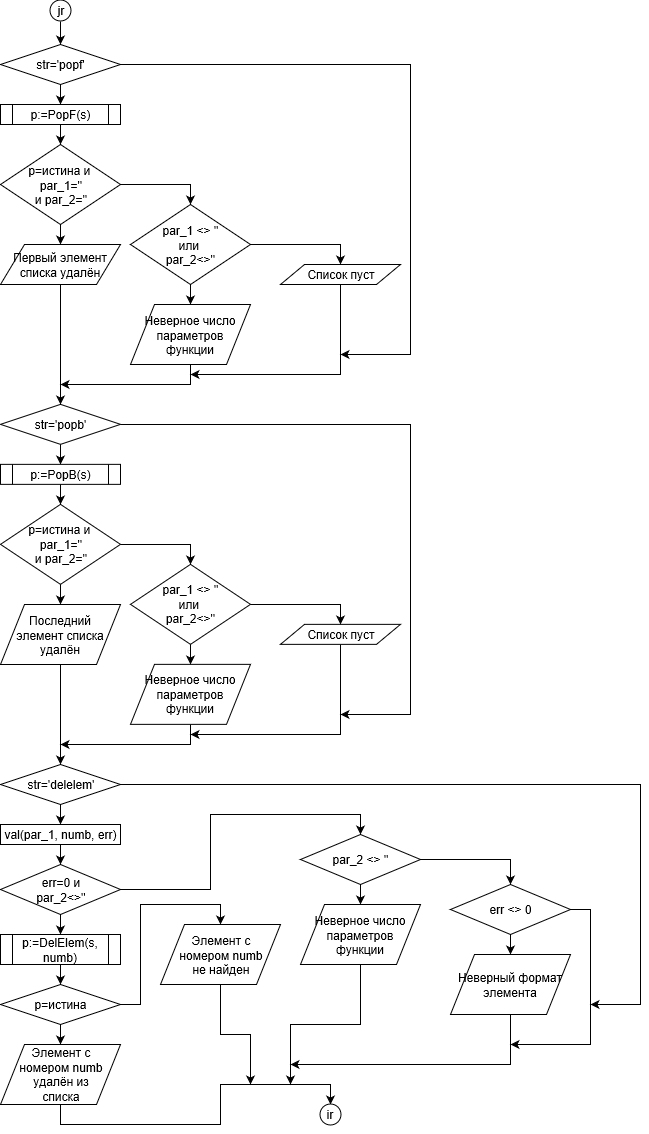
**Схема алгоритмов:**

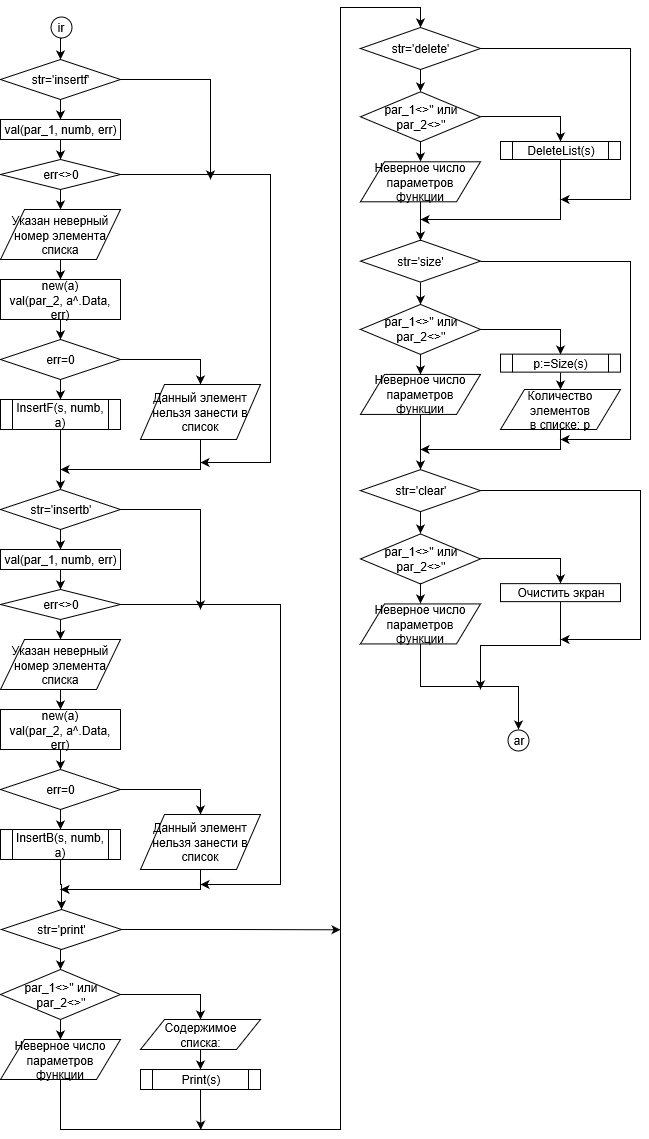
****

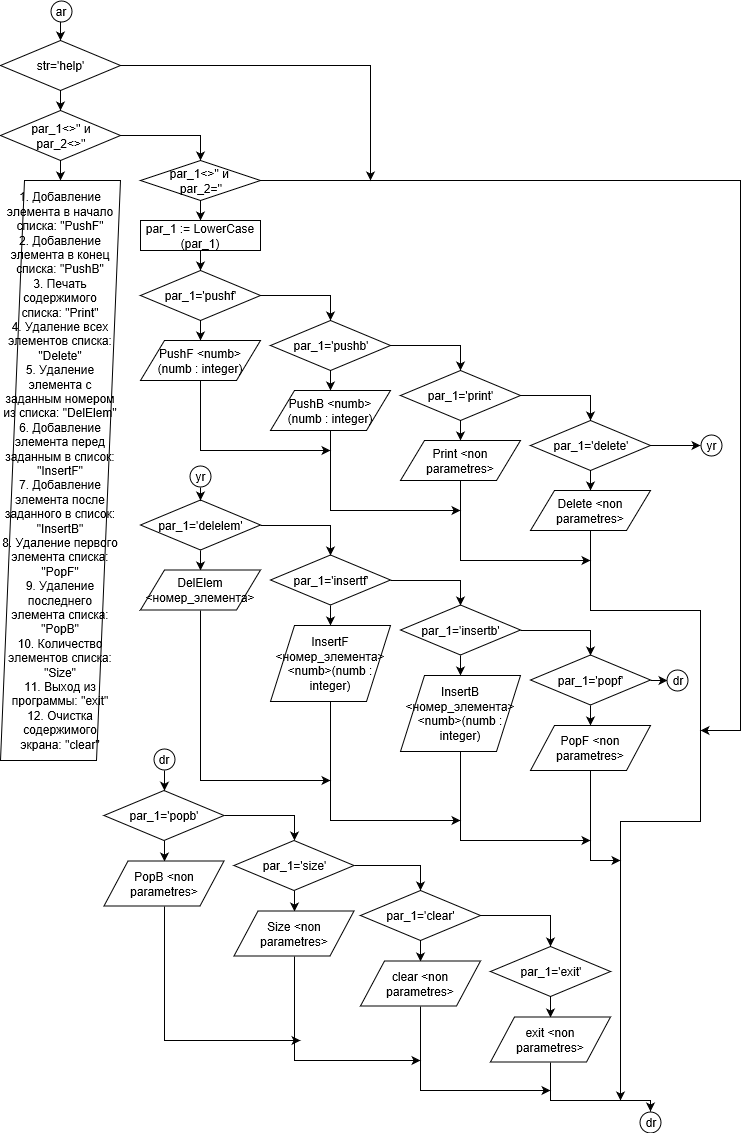
**Изображение выглядит как текст, карта

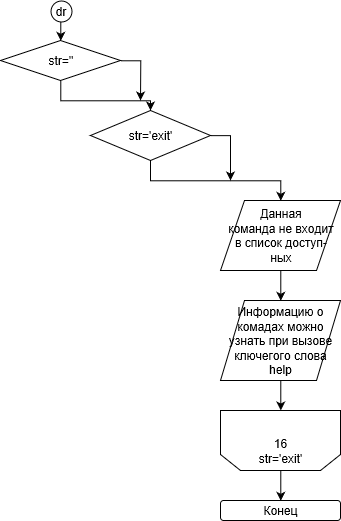
Автоматически созданное описание**

****

****

****

****

****

**Исходный текст программы:**

**unit Lists;**

**interface**

**type pElem = ^Elem;**

**Elem = record**

**Data : integer;**

**pnext : pElem;**

**pprev : pElem;**

**end;**

**List = record**

**pfirst : pElem;**

**plast : pElem;**

**end;**

**procedure PushF(var s : List; const pnumber : pElem);**

**procedure PushB(var s : List; const pnumber : pElem);**

**procedure Print(var s : List);**

**procedure DeleteList(var s : List);**

**function Size(var s : List) : integer;**

**function PopF(var s : List) : boolean;**

**function PopB(var s : List) : boolean;**

**implementation**

**procedure PushF(var s : List; const pnumber : pElem);**

**begin**

**if (pnumber = nil) then exit;**

**pnumber^.pnext := nil;**

**pnumber^.pprev := nil;**

**if (s.pfirst = nil) then begin**

**s.pfirst := pnumber;**

**s.plast := pnumber;**

**end else begin**

**pnumber^.pnext := s.pfirst;**

**s.pfirst^.pprev := pnumber;**

**s.pfirst := pnumber;**

**end;**

**end;**

**procedure PushB(var s : List; const pnumber : pElem);**

**begin**

**if (pnumber = nil) then exit;**

**pnumber^.pnext := nil;**

**pnumber^.pprev := nil;**

**if (s.pfirst = nil) then begin**

**s.pfirst := pnumber;**

**s.plast := pnumber;**

**end else begin**

**s.plast^.pnext := pnumber;**

**pnumber^.pprev := s.plast;**

**s.plast := pnumber;**

**end;**

**end;**

**procedure Print(var s : List);**

**var x : pElem;**

**i : integer;**

**begin**

**x := s.pfirst;**

**write('(');**

**i := 0;**

**while (x <> nil) do begin**

**inc(i);**

**if (i > 1) then write(', ');**

**write(x^.Data);**

**x := x^.pnext;**

**end;**

**writeln(')');**

**end;**

**procedure DeleteList(var s : List);**

**var next, del : pElem;**

**begin**

**if (s.pfirst = nil) then exit;**

**next := s.pfirst;**

**while (next <> nil) do begin**

**del := next;**

**next := next^.pnext;**

**dispose(del);**

**end;**

**s.pfirst := nil;**

**s.plast := nil;**

**end;**

**function GetElem(var s : List; const number : integer) : pElem;**

**var i : integer;**

**next, res : pElem;**

**begin**

**res := nil;**

**i := 1;**

**next := s.pfirst;**

**while (i <= number) and (next <> nil) do begin**

**if (i = number) then begin**

**res := next;**

**break;**

**end;**

**inc(i);**

**next := next^.pnext;**

**end;**

**GetElem := res;**

**end;**

**function DelByPoint(var s : List; var pElement : pElem) : boolean;**

**begin**

**DelByPoint := false;**

**if (pElement = nil) then exit;**

**if (pElement = s.pfirst) then begin**

**s.pfirst := pElement^.pnext;**

**if (s.pfirst = nil) then**

**s.plast := nil**

**else**

**s.pfirst^.pprev := nil;**

**end else if (pElement = s.plast) then begin**

**s.plast := pElement^.pprev;**

**if (s.plast = nil) then**

**s.pfirst := nil**

**else**

**s.plast^.pnext := nil;**

**end else begin**

**pElement^.pprev^.pnext := pElement^.pnext;**

**pElement^.pnext^.pprev := pElement^.pprev;**

**end;**

**dispose(pElement);**

**pElement := nil;**

**DelByPoint := true;**

**end;**

**function PopF(var s : List) : boolean;**

**var x : pElem;**

**begin**

**x := s.pfirst;**

**PopF := DelByPoint(s, x);**

**end;**

**function PopB(var s : List) : boolean;**

**var x : pElem;**

**begin**

**x := s.plast;**

**PopB := DelByPoint(s, x);**

**end;**

**function Size(var s : List) : integer;**

**var count : integer;**

**pElement : pElem;**

**begin**

**count := 0;**

**pElement := s.pfirst;**

**while (pElement <> nil) do begin**

**inc(count);**

**pElement := pElement^.pnext;**

**end;**

**Size := count;**

**end;**

**end.**

**uses Lists, crt;**

**const \_SIZE = 8;**

**width = 120;**

**type mas = array [1..\_SIZE] of string;**

**var rk:char;**

**s, s1,s2: string;**

**beginy, posx, posy,i, s\_2, code, count, kol : integer;**

**lis : List;**

**a : array [1..100000] of string;**

**b : pelem;**

**command : mas;**

**procedure tab(const str : string);**

**var buf : string;**

**t : integer;**

**begin**

**i:=1;**

**buf := '';**

**while (i <= length(s)) and (s[i] = ' ') do inc(i);**

**if (Pos(LowerCase(copy(s, i, (wherey-beginy)\* width + wherex - 2 - i)), str) = 1) then begin**

**GoToXY((i + 2) mod width, beginy + (i + 2) div width);**

**for t := 1 to length(s) - i + 1 do write(' ');**

**GoToXY((i + 2) mod width, beginy + (i + 2) div width);**

**write(str);**

**for t := 1 to i - 1 do buf := buf + ' ';**

**s := buf + str;**

**end;**

**end;**

**procedure cmnd\_str;**

**var i, t : integer;**

**begin**

**s1 := '';**

**s2 := '';**

**s:='';**

**write('>:');**

**beginy := wherey();**

**repeat**

**rk:=ReadKey;**

**if rk=#0 then begin**

**rk:=ReadKey;**

**if (rk = #75) then begin**

**if (wherex() > 3) and (wherey() = beginy)then GoToXY(wherex() - 1, wherey);**

**if (wherex() = 1) and (beginy < wherey()) then GoToXY(width, wherey() - 1);**

**if (wherey() > beginy) then GoTOXY(wherex() - 1,wherey());**

**end;**

**if (rk=#77) then begin**

**if (wherex() + width \* (wherey() - beginy) <= length(s) + 2) then GoToXY(wherex() + 1, wherey());**

**if (wherex() = width) then GoToXY(1, wherey() + 1);**

**end;**

**if (rk=#72) then begin**

**while (wherey() <> beginy) do begin**

**DelLine();**

**GoToXY(1, wherey() - 1);**

**end;**

**DelLine();**

**GoToXY(1, wherey());**

**if (count < kol - 1) then inc(count);**

**s := a[kol-count];**

**write('>:' + s);**

**end;**

**if (rk=#80) then begin**

**while (wherey() <> beginy) do begin**

**DelLine();**

**GoToXY(1, wherey() - 1);**

**end;**

**DelLine();**

**GoToXY(1, wherey());**

**if (count > 1) then dec(count);**

**s := a[kol-count];**

**write('>:' + s);**

**end;**

**if rk=#83 then begin**

**delete(s,(wherey-beginy)\* width + (wherex - 2),1);**

**posx := wherex();**

**posy := wherey();**

**GoToXY(3, beginy);**

**for i := 1 to (length(s) + 1) do write(' ');**

**GoToXY(3,beginy);**

**write(s);**

**GoToXY(posx, posy)**

**end;**

**end else if (rk>#31) and (rk<#123) then**

**begin**

**insert(rk,s,(wherey-beginy)\* width + (wherex - 2));**

**posx := wherex() + 1;**

**posy := wherey();**

**if (wherey() > 30) then begin clrscr(); write('>:'); beginy := 1; end else GoToXY(3,beginy);**

**write(s);**

**if (posx = width + 1) then GoToXY(1, posy + 1) else GoToXY(posx, posy);**

**end;**

**if rk=#8 then begin**

**delete(s,(wherey-beginy)\* width + (wherex - 3),1);**

**posx := wherex();**

**posy := wherey();**

**GoToXY(3, beginy);**

**for i := 1 to length(s) + 1 do write(' ');**

**GoToXY(3,beginy);**

**write(s);**

**if (posy > beginy) and (posx = 1) then GoToXY(width, (posy - 1)) else**

**if (posy > beginy) then GoToXY((posx-1), posy) else**

**if (posy = beginy) and (posx = 3) then GoToXY(posx, posy) else**

**if (posy = beginy) and (posx > 3) then GoToXY(posx-1,posy);**

**end;**

**if (rk = #9) then begin**

**for t := 1 to \_SIZE do tab(command[t]);**

**end;**

**until rk = #13;**

**if (kol > 100000) then kol := 1;**

**a[kol] := s;**

**inc(kol);**

**count := 0;**

**writeln;**

**if (s <> '') then begin**

**i:=1;**

**while (i <= length(s))and (s[i] = ' ') do inc(i);**

**if (Pos(' ', copy(s, i, length(s) - i + 1)) = 0) then begin**

**s1 := copy(s, i, length(s) - i + 1);**

**s2 := '';**

**end else begin**

**s1 := copy(s, i, Pos(' ', copy(s, i, length(s) - i + 1)) - 1);**

**i:= i + Pos(' ', copy(s, i, length(s) - i + 1)) - 1;**

**while (i <= length(s)) and (s[i] = ' ') do inc(i);**

**if (Pos(' ', copy(s, i, length(s) - i)) = 0)**

**then s2 := copy(s, i, length(s) - i + 1)**

**else s2 := copy(s, i, Pos(' ', copy(s, i, length(s) - i)) - 1);**

**end;**

**end;**

**end;**

**procedure help;**

**begin**

**writeln('help ');**

**writeln('PushF[] - занесение элемента в начало списка ');**

**writeln('PushB[] - занесение элемента в конец списка ');**

**writeln('delete - удаление списка ');**

**writeln('size - узнать количество элементов в списке ');**

**writeln('output - вывести все элементы очереди ');**

**writeln('clrscr - очистка экрана ');**

**writeln('help - помощь ');**

**writeln('стрелки вверх/вниз - история команд ');**

**writeln('exit - выход из программы ');**

**end;**

**begin**

**command[1] := 'help';**

**command[2] := 'pushf';**

**command[3] := 'pushb';**

**command[4] := 'popf';**

**command[5] := 'popb';**

**command[6] := 'size';**

**command[7] := 'print';**

**command[8] := 'exit';**

**command[9] := 'delete';**

**kol := 1;**

**repeat**

**cmnd\_str;**

**s1:=LowerCase(s1);**

**case s1 of**

**'help': help;**

**'pushf':**

**begin**

**new(b);**

**val(s2,b^.data,code);**

**if code=0 then**

**PushF(lis, b)**

**else writeln('Неправильный набор параметров');**

**end;**

**'pushb':**

**begin**

**new(b);**

**val(s2,b^.data,code);**

**if code=0 then**

**PushB(lis, b)**

**else writeln('Неправильный набор параметров');**

**end;**

**'popf':**

**begin**

**if s2 = '' then popf(lis)**

**else writeln('Неправильный набор символов')**

**end;**

**'popb':**

**begin**

**if s2 = '' then popb(lis)**

**else writeln('Неправильный набор символов')**

**end;**

**'delete':**

**begin**

**if s2 = '' then DeleteList(lis)**

**else writeln('Неправильный набор символов')**

**end;**

**'size':**

**begin**

**if s2 = '' then writeln(size(lis))**

**else writeln('Неправильный набор символов')**

**end;**

**'print':**

**begin**

**if s2 = '' then print(lis)**

**else writeln('Неправильный набор символов')**

**end;**

**'clrscr': ClrScr();**

**'': write();**

**'exit': exit;**

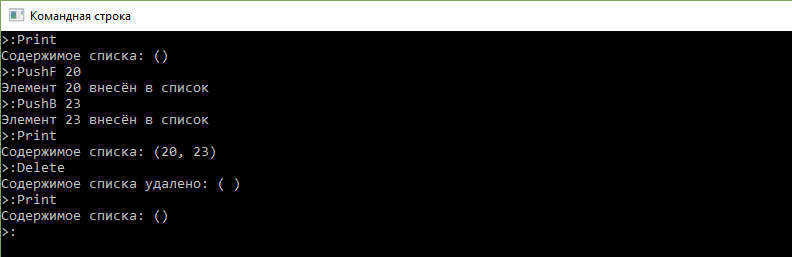
**else writeln('Неизвестная команда');**

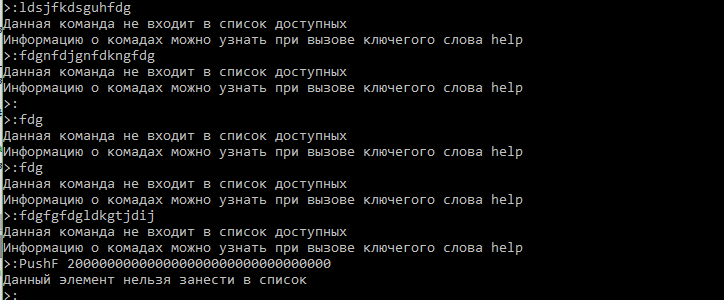
**end;**

**until (s = 'exit');**

**end.**

**Скриншоты выполнения программы**

****

****

**Вывод**

В ходе данной лабораторной работы были получены знания для работы с собственными структурами данных. В частности, с двусвязным линейным списком. Были изучены принципы организации программных модулей, закреплены навыки работы с динамической памятью, получены базовые навыки организации работы в режиме командной строки. Были осуществлены такие консольные примитивы, как удаление и добавление символов в строку, реализован принцип автодополнения по нажатию кнопки Tab, добавлена возможность просмотра истории команд на принципе добавления каждых предыдущих команд в линейный список, по которому можно перемещаться по нажатиям клавиш вверх и вниз соответственно.