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-12 /Крючков И. С/ Проверил /Чистяков Г. А./

Киров 2021

**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 "Дек".

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 алгоритма:**

**Основная программа:**

****

**Модуль shell**

****

****

****

****

****

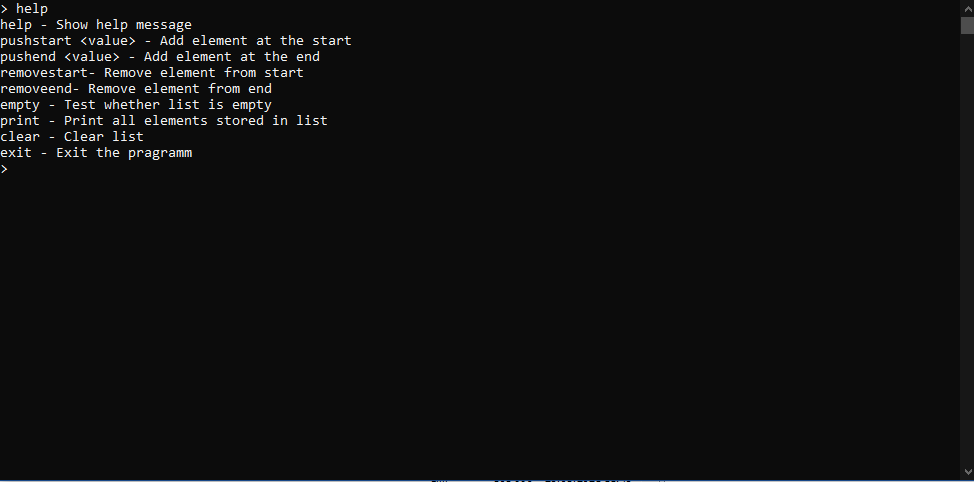
****

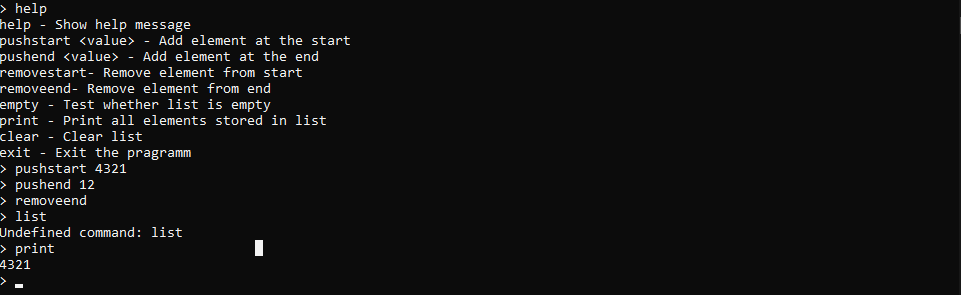
**Модуль** **LinkedList**

****

****

**Экранная форма:**

****

****

**Листинг кода:**

**Основная программа**

uses shell;

begin

start;

end.

**Модуль shell**

unit shell;

interface

uses

crt, sysutils, LinkedList;

const

// Количество команд

cmdlist\_size = 9;

// Список команд

cmdlist: array[0..(cmdlist\_size - 1)] of string = (

'clear',

'exit',

'empty',

'help',

'pushend',

'pushstart',

'removestart',

'removeend',

'print'

);

// Количество аргументов у каждой команды

arg\_num: array[0..(cmdlist\_size - 1)] of integer = (

0, 0, 0, 0, 1, 1, 0, 0, 0

);

// Максимальное количество элементов в истории команд

max\_hist\_elems = 100;

// Максимальная длина строки

string\_size = 50;

st : string = '> ';

type

THistory = 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

hr: THistory; // История команд

list: TUList;

cmd: string; // текущая команда

p: integer; // количество символов в строке

pc: integer; // вспомогательная переменная

procedure start();

implementation

procedure history\_init;

var

i: integer;

begin

setlength(hr.cmds, 1);

hr.size := 1;

for i := 0 to hr.size - 1 do

hr.cmds[i] := '';

hr.current := hr.size - 1;

hr.last := hr.size - 1;

hr.deep := 0;

end;

// Удаление старой записи из истории для добаления новой, если записей максимальное количество

function history\_shift(v, sz, sh: integer): integer;

begin

history\_shift := (v + sz + sh) mod sz;

end;

procedure history\_update;

begin

hr.cmds[hr.current] := cmd;

end;

// Добавление записи в историю

procedure history\_push;

begin

hr.cmds[hr.last] := cmd;

if hr.size < max\_hist\_elems then

begin

setlength(hr.cmds, hr.size + 1);

hr.last := hr.size;

inc(hr.size);

end

else

hr.last := history\_shift(hr.last, hr.size, 1);

hr.deep := 0;

hr.current := hr.last;

end;

procedure str\_update(offset, pre\_compl: integer; str\_pre\_coml: string);

var

x, y: integer;

begin

x := wherex;

y := wherey;

write(chr(13));

clreol;

if pre\_compl = 0 then

write(chr(13), st, cmd)

else

begin

write(chr(13), st, cmd);

Textbackground(3);

Textcolor(0);

write(str\_pre\_coml);

Textbackground(0);

Textcolor(15);

end;

history\_update;

if x + offset > 0 then

gotoxy(x + offset, y);

end;

//Удаление пробелов

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;

procedure str\_goto\_begin;

var

x, y: integer;

begin

p := 0;

write(chr(13));

x := wherex;

y := wherey;

gotoxy(x + length(st), y);

end;

procedure str\_goto\_end;

var

x, y, offset : integer;

begin

x := wherex;

y := wherey;

offset := length(cmd) - p;

p := length(cmd);

if x + offset > 0 then

gotoxy(x + offset, y);

end;

procedure str\_pre\_complete;

var

i, offset, cnt, k: integer;

cp, rm: string;

begin

cp := cmd;

str\_skip\_whitesp(cp);

offset := 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;

if cnt = 1 then

begin

pc := 1;

// str\_goto\_begin;

// cmd := copy(cmd, 0, offset) + cmdlist[k];

rm := copy(cmdlist[k], length(cp)+1, length(cmdlist[k]));

str\_update(0, 1, rm);

// str\_goto\_end;

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, 0, '');

str\_pre\_complete;

end;

end;

procedure str\_delete\_char;//\*\*//

begin

if p < length(cmd) then

begin

delete(cmd, p + 1, 1);

str\_update(0, 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, 0, '');

end;

end;

procedure str\_shift\_left; //\*\*//

begin

if p > 0 then

begin

dec(p);

str\_update(-1, 0, '');

end;

end;

procedure str\_shift\_right;

begin

if p < length(cmd) then

begin

inc(p);

str\_update(1, 0, '');

end;

end;

procedure history\_prev\_cmd;

begin

if hr.deep < hr.size - 1 then

begin

hr.current := history\_shift(hr.current, hr.size, -1);

inc(hr.deep);

str\_goto\_begin;

cmd := hr.cmds[hr.current];

str\_update(0, 0, '');

str\_goto\_end;

end;

end;

procedure history\_next\_cmd;

begin

if hr.deep > 0 then

begin

hr.current := history\_shift(hr.current, hr.size, 1);

dec(hr.deep);

str\_goto\_begin;

cmd := hr.cmds[hr.current];

str\_update(0, 0, '');

str\_goto\_end;

end;

end;

//Автодополнение

procedure str\_autocomplete;

var

i, offset, cnt, k : integer;

cp: string;

begin

cp := cmd;

str\_skip\_whitesp(cp);

offset := 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;

if cnt = 1 then

begin

str\_goto\_begin;

cmd := copy(cmd, 1, offset) + cmdlist[k];

str\_update(0, 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('Undefined 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('Undefined arg: ', tokens[i]);

err := i;

exit;

end;

end;

if arg\_num[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(st), 0, '');

end;

procedure hotkeys;

var

code : integer;

begin

code := ord(readkey);

case code of

75 : //left

str\_shift\_left;

77 : //right

str\_shift\_right;

72 : //up

history\_prev\_cmd;

80 : //down

history\_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

history\_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;

'pushend' :

list\_push\_end(list, vals[0]);

'pushstart' :

list\_push\_start(list, vals[0]);

'removestart':

list\_remove\_start(list);

'removeend':

list\_remove\_end(list);

'print' :

list\_print(list);

end;

str\_clear;

end;

procedure start();//\*\*//

var

ch: char;

code: integer;

begin

clrscr;

history\_init;

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: str\_backsp\_char; //backspace

9: str\_autocomplete; //tab (autocomplete)

0: hotkeys

else

str\_add\_char(ch);

end;

end;

list\_delete(list);

end;

end.

**Модуль LinkedList**

unit LinkedList;

interface

type

TList = record

value: integer;

next: ^TList;

end;

TUList = record

first: ^TList;

end;

procedure list\_create(var list: TUList);

procedure list\_delete(var list: TUList);

function list\_empty(list: TUList): Boolean;

procedure list\_push\_start(var list: TUList; value: integer);

procedure list\_push\_end(var list: TUList; value: integer);

procedure list\_print(list: TUList);

procedure list\_remove\_start(var list: TUList);

procedure list\_remove\_end(var list: TUList);

procedure list\_help;

implementation

procedure list\_create(var list: TUList);

begin

list.first := nil;

end;

procedure list\_delete(var list: TUList);

var

entList: ^TList;

begin

while (list.first <> nil) do

begin

entList := list.first^.next;

dispose(list.first);

list.first := entList;

end;

end;

function list\_empty(list: TUList) : 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\_start(var list: TUList; value: integer);

var entList: ^TList;

begin

new(entList);

entList^.value := value;

entList^.next := list.first;

list.first := entList;

end;

procedure list\_push\_end(var list: TUList; value: integer);

type

TLe = ^TList;

var

entList: ^TList;

head: ^TLe;

begin

new(entList);

entList^.value := value;

entList^.next := nil;

head := @list.first;

while head^ <> nil do

begin

head := @(head^)^.next;

end;

head^ := entList;

end;

procedure list\_remove\_start(var list: TUList);

type

TLe = ^TList;

var

entList: ^TList;

head: ^TLe;

begin

head := @list.first;

if (head^ = nil) then

writeln('No such values in list')

else

begin

entList := (head^)^.next;

dispose(head^);

head^ := entList;

end;

end;

procedure list\_remove\_end(var list: TUList);

type

TLe = ^TList;

var

head: ^TLe;

begin

head := @list.first;

if (head^ = nil) then

writeln('No such values in list')

else

begin

while (head^)^.next <> nil do

begin

head := @(head^)^.next;

end;

dispose(head^);

head^ := nil;

end;

end;

procedure list\_print(list: TUList);

var

entList : ^TList;

begin

entList := list.first;

while (entList <> nil) do

begin

write(entList^.value, ' ');

entList := entList^.next;

end;

writeln;

end;

procedure list\_help;

begin

writeln('help - Show help message');

writeln('pushstart <value> - Add element at the start');

writeln('pushend <value> - Add element at the end');

writeln('removestart- Remove element from start');

writeln('removeend- Remove element from end');

writeln('empty - Test whether list is empty');

writeln('print - Print all elements stored in list');

writeln('clear - Clear list');

writeln('exit - Exit the pragramm');

end;

end.

**Вывод:** в ходе выполнения лабораторной работы были изучены структуры и принципы организации программных модулей, закреплены навыки работы с динамической память, получены базовые навыки работы в режиме командной строки. Освоены структуры данных, на примере структуры «Дек».