**МИНОБРНАУКИ РОССИИ**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №4

Дисциплина: Алгоритмы и структуры данных

по теме «Структуры данных «линейные списки» »

Выполнил: ст. группы ВТ-22  
Макаров Даниил Сергеевич

Проверил: Синюк В.Г.

Белгород 2018

Л а б о р а т о р н а я р а б о т а № 5

**Структуры данных «линейные списки» (Pascal/С)**

**Цель работы:** изучить СД типа «линейный список», научиться их программно реализовывать и использовать.

З а д а н и е

1. Для СД типа «линейный список» определить:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.

1.1.2. Набор допустимых операций.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения.

1.2.2. Объем памяти, занимаемый экземпляром СД.

1.2.3. Формат внутреннего представления СД и способ его интерпретации.

1.2.4. Характеристику допустимых значений.

1.2.5. Тип доступа к элементам.

1.3. Логический уровень представления СД.

Способ описания СД и экземпляра СД на языке программирования.

2. Реализовать СД типа «линейный список» в соответствии с вариантом индивидуального задания (см. табл.14) в виде модуля.

3. Разработать программу для решения задачи в соответствии с вариантом индивидуального задания (см. табл.14) с использованием модуля, полученного в результате выполнения пункта 2 задания.

**Задание 1**

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости: динамическая, последовательная, линейная структура данных

1.1.2. Набор допустимых операций: инициализация, включение элемента, исключение элемента, чтение текущего элемента, переход в начало списка, переход в конец списка, переход к следующему элементу, переход к i-му элементу, определение длинны списка, уничтожение списка.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения: последовательная или связная схема хранения.

1.2.2. Объем памяти, занимаемый экземпляром СД: Зависит от типа линейного списка и количества элементов в списке.

1.2.3. Формат внутреннего представления СД и способ его интерпретации: Дескриптор хранится в статической памяти. Элементы списка хранятся в динамической памяти.

1.2.4. Характеристику допустимых значений: CAR(ЛС) = CAR(BaseType)0 + CAR(BaseType)1 +… + CAR(BaseType)max, где CAR(BaseType) — кардинальное число элемента ЛС типа BaseType, max — максимальное количество элементов в ЛС (не всегда определено, т.к. может зависеть от объема свободной динамической памяти).

1.2.5. Тип доступа к элементам: Для ПЛС - прямой, для ОЛС – последовательный, для ДЛС – последовательный.

1.3. Логический уровень представления СД.

Способ описания СД и экземпляра СД на языке программирования:

ПЛС:

typedef void \*t\_base;

typedef unsigned ptrel;

typedef struct List { t\_base memlist[sizeList]; ptrel ptr; unsigned N; unsigned size; } t\_List;

ОЛС:

typedef int t\_base;

typedef struct element \*ptrel;

typedef struct element { t\_base data; ptrel next; } t\_element;

typedef struct List { ptrel start, ptr; unsigned n; } t\_list;

ДЛС:

typedef void \*t\_base;

typedef struct element \*ptrel;

typedef struct element { t\_base data; ptrel next; ptrel last; } t\_element;

typedef struct List {ptrel start; ptrel end; ptrel ptr; unsigned N; } t\_List;

**Задание 2**

Вариант 7

Задача

Многочлен P(x) = anxn + an-1xn- 1+ ... + a1x + a0 с целыми коэффициентами можно представить в виде списка, причем если ai = 0, то соответствующее звено не включать в список. Определить процедуру, которая стоит многочлен p — сумму многочленов q и r;

Формат

7. ПЛС. Массив, на основе которого реализуется ПЛС, находится в динамической памяти (базовый тип элемента определяется задачей). Память под массив выделяется при инициализации ПЛС и количество элементов сохраняется в дескрипторе.

#ifndef LIST\_H\_

#define LIST\_H\_

extern const char list\_ok;

extern const char list\_mem\_error;

extern const char list\_over;

extern const char list\_under;

char list\_error;

typedef struct polynom\_node{

int a;

int exp;

}node;

typedef node list\_base\_type;

typedef struct Element{

list\_base\_type data;

struct Element\* next;

}element;

typedef struct List {

element\* pos\_start;

element\* pos\_current;

unsigned list\_lenght;

unsigned list\_size;

}list;

element\* element\_gen(list\_base\_type data);

list\* list\_init(unsigned size);

void list\_put(list \*l, list\_base\_type e);

list\_base\_type list\_get(list \*l);

list\_base\_type list\_read(list \*l);

int list\_full(list \*l);

int list\_empty(list \*l);

int list\_end(list \*l);

unsigned list\_count(list \*l);

void pos\_move\_to\_first(list \*l);

void pos\_move\_to\_last(list \*l);

void list\_next(list \*l);

void list\_move\_to(list \*l, unsigned int index);

void list\_done(list \*l);

void list\_copy(list \*list\_1,list \*list\_2);

#endif

#include <stdio.h>

#include <stdlib.h>

#include "list.h"

const char list\_ok = 0;

const char list\_mem\_error = 1;

const char list\_over = 2;

const char list\_under = 3;

list\* list\_init(unsigned size){

list\_error=list\_ok;

list\* new\_list;

element\* fake;

new\_list=(list\*)malloc(sizeof(list));

if(new\_list==NULL){

list\_error=list\_mem\_error;

exit(list\_error);

}

fake=(element\*)malloc(sizeof(element));

new\_list->pos\_start=fake;

new\_list->pos\_current=fake;

new\_list->list\_lenght=0;

new\_list->list\_size=size;

return new\_list;

}

void list\_put(list \*l, list\_base\_type e){

if(list\_full(l)){

list\_error=list\_over;

exit(list\_error);

}

element\* new\_element=element\_gen(e);

new\_element->data=e;

new\_element->next=l->pos\_current->next;

l->pos\_current->next=new\_element;

l->list\_lenght++;

}

list\_base\_type list\_get(list \*l){

if(list\_empty(l)){

list\_error=list\_under;

exit(list\_error);

}

if(list\_end(l)){

list\_error=list\_under;

exit(list\_error);

}

list\_base\_type temp;

temp=l->pos\_current->next->data;

l->pos\_current->next=l->pos\_current->next->next;

l->list\_lenght--;

return temp;

}

list\_base\_type list\_read(list \*l){

if(list\_empty(l)){

list\_error=list\_under;

exit(list\_error);

}

return l->pos\_current->data;

}

int list\_full(list \*l){

return l->list\_lenght>=l->list\_size;

}

int list\_empty(list \*l){

return l->list\_lenght==0;

}

int list\_end(list \*l){

return l->pos\_current==NULL;

}

unsigned list\_count(list \*l){

return l->list\_lenght;

}

void pos\_move\_to\_first(list \*l){

l->pos\_current=l->pos\_start;

}

void pos\_move\_to\_last(list \*l){

while(!(list\_end(l))){

list\_next(l);

}

}

void list\_next(list \*l){

if(l->pos\_current->next!=NULL){

l->pos\_current=l->pos\_current->next;

}

else{

list\_error=list\_under;

printf("error %d\n",list\_under);

exit(list\_error);

}

}

void list\_move\_to(list \*l, unsigned int index){

if(index>l->list\_lenght){

list\_error=list\_under;

exit(list\_error);

}

pos\_move\_to\_first(l);

for(int i=0;i<index;i++) list\_next(l);

}

void list\_done(list \*l){

free(l);

}

void list\_copy(list \*list\_1,list \*list\_2){

pos\_move\_to\_first(list\_1);

pos\_move\_to\_last(list\_2);

int free\_space=list\_2->list\_size-list\_2->list\_lenght;

for(int i=0;((i<free\_space)&&(i<list\_1->list\_lenght));i++){

list\_2->pos\_current->next=list\_1->pos\_current;

list\_next(list\_1);

list\_next(list\_2);

}

}

element\* element\_gen(list\_base\_type data){

element\* temp;

temp=(element\*)malloc(sizeof(element));

temp->data=data;

temp->next=NULL;

return temp;

}

Исходный код модуля task.h

#include <stdio.h>

#include <stdlib.h>

#include "task.h"

#include "list.h"

#include <string.h>

list\* str\_to\_list(char\* str){

list \*poly;

poly=list\_init(strlen(str)/2);

node temp;

char flag;

pos\_move\_to\_first(poly);

int i=0;

while(str[i]!='\0'){

temp.a=0;

temp.exp=0;

while((str[i]>='0')&&(str[i]<='9')){

temp.a+=str[i]-'0';

temp.a\*=10;

i++;

}

temp.a=temp.a/10;

i+=2;

while((str[i]>='0')&&(str[i]<='9')){

temp.exp+=str[i]-'0';

temp.exp\*=10;

ы

}

temp.exp=temp.exp/10;

if(str[i]=='+') i++;

if(temp.a!=0) {

list\_put(poly,temp);

list\_next(poly);

}

}

return poly;

}

list\* sum\_poly(list\* poly1,list\* poly2){

list\* result=list\_sum(poly1,poly2);

print\_poly(result);

list\_base\_type temp1,temp2;

for(int i=0;i<result->list\_lenght-1;i++){

list\_move\_to(result,i);

temp1=list\_read(result);

for(int j=i+1;j<result->list\_lenght;j++){

list\_move\_to(result,j);

temp2=list\_read(result);

if(temp1.exp==temp2.exp){

list\_move\_to(result,j-1);

list\_get(result);

list\_move\_to(result,i);

result->pos\_current->data.a+=temp2.a;

}

}

}

return result;

}

void print\_poly(list\* polynom){

pos\_move\_to\_first(polynom);

list\_next(polynom);

while(!(list\_end(polynom))){

printf("%dx^%d",polynom->pos\_current->data.a,polynom->pos\_current->data.exp);

if(polynom->pos\_current->next!=NULL){

printf("+");

}

list\_next(polynom);

}

printf("%dx^%d\n",polynom->pos\_current->data.a,polynom->pos\_current->data.exp);

}