Министерство образование Республики Беларусь

Учреждение образования «Полоцкий государственный университет»

Информационно-технологический факультет

Кафедра технологий программирования

Лабораторная работа №6

По дисциплина «Основы алгоритмизации и программирования»

Выполнил: Крутилин С.Д.

Группа 19-ИТ-1.

Проверил: Адамовский Е.

Новополоцк

2019 г.

**Введение в абстрактные структуры**

**Вариант 2**

**Цель работы**: изучить методы организации списочных структур в динамической памяти. Реализовать алгоритмы добавления, удаления и сортировки списков.

**Задание 1.** Функции для создания нового списка и его удаления.

**Описание инициализации:** выделяем память под фиктивный элемент; делаем проверку на выделение и при необходимости завершаем работу; указателю ptr присваиваем указатель на фиктивный элемент; присваиваем указателю элемента указатель на NULL.

/\* Инициализация списка \*/

void initListOne(ListOne\* L) {

L->start = (element\*)malloc(sizeof(element));

if (!L->start) {

listOneError = listOneNoMem;

return;

}

L->ptr = L->start;

L->ptr->link = NULL;

}

**Описание удаления:** установка рабочего указателя в начало; создание временной переменной; до тех пор, пока список не пуст удаляем по одному элементу из списка; в конце удаляем фиктивный элемент.

/\* Очистка списка \*/

void doneListOne(ListOne\* L) {

beginPtrListOne(L);

listOneBaseType temp;

while (!isEmptyListOne(L))

getListOne(L, &temp);

free((void\*)L->ptr);

}

**Задание 2.** Функции добавления элемента в начало и после указанного элемента.

**Описание:** Задание было реализовано в виде меню в функции main с помощью функций movePtrListOne(), который осуществлял сдвиг рабочего указателя в нужную позицию; затем вставлялся элемент с помощью функции putListOne2() (создание нового элемента, новый элемент указывает на новый, старый элемент новый); возвращение рабочего указателя в начало. Весь необходимый код можно посмотреть ниже.

**Задание 3.** Функции удаления элемента.

**Описание:** принцип удаления элемента такой же, как и в добавлении: переходим к нужному, удаляем, возвращаем указатель на начало.

**Задание 4.** Функции печати всех элементов списка.

**Описание:** с помощью дополнительного независимого элемента перебираем каждый элемент списка и на каждом шаге выполняем печать данных, затем удаляем элемент.

/\* Печать всех элементов списка с начала до конца \*/

void printListOne(ListOne\* L) {

// дополнительный элемент для независимого

// прохода от начала до конца списка

element\* tmp;

// ссылка на первый элемент

tmp = L->start->link;

// пока не конец списка

while (tmp != NULL) {

cout << " " << tmp->data;

// печать текущего элемента

tmp = tmp->link;

// независимый переход на следующий элемент

}

cout << endl;

free(tmp); // удаление дополнительного элемента

}

#include <iostream>

#include <cstdlib>

#include <clocale>

#include <Windows.h>

using namespace std;

// переменная ошибок

int listOneError;

// описание исключительных ситуаций

const int listOneOK = 0;

const int listOneEmpty = 1;

const int listOneNoMem = 2;

const int listOneEnd = 3;

/\* Базовый тип списка \*/

typedef char listOneBaseType[20];

/\* Структура элемента списка \*/

struct element {

listOneBaseType data; // данные

element\* link; // указатель на следующий элемент списка

};

/\* Дескриптор списка \*/

struct ListOne {

element\* start; // указатель на фиктивный элемент

element\* ptr; // указатель на рабочий элемент

};

/\* Сдвиг рабочего указателя \*/

void movePtrListOne(ListOne\* L) {

// если рабочий указатель указывает на последний элемент

if (!L->ptr->link) {

listOneError = listOneEnd;

return;

}

// иначе смещение указателя

L->ptr = L->ptr->link;

}

/\* Установка рабочего указателя в начало списка \*/

void beginPtrListOne(ListOne\* L) {

L->ptr = L->start;

}

/\* Инициализация списка \*/

void initListOne(ListOne\* L) {

// выделение памяти под фиктивный элемент

L->start = (element\*)malloc(sizeof(element));

// если памяти не хватило

if (!L->start) {

listOneError = listOneNoMem;

return;

}

// ptr указывает на то, что и start - на фиктивный

L->ptr = L->start;

// нет ссылки на следующий элемент

L->ptr->link = NULL;

}

/\* Включение элемента в список \*/

void putListOne(ListOne\* L, listOneBaseType E) {

// выделение памяти под новый элемент

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

// если памяти не хватило

if (!L->start) {

listOneError = listOneNoMem;

return;

}

// внесение информации в новый элемент

strcpy\_s(temp->data, E);

// новый элемент указывает на следующий

temp->link = L->ptr->link;

// текущий элемент указывает на новый

L->ptr->link = temp;

}

void putListOne2(ListOne\* L, listOneBaseType E) {

// выделение памяти под новый элемент

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

// если памяти не хватило

if (!L->start) {

listOneError = listOneNoMem;

return;

}

element\* p;

p = L->ptr->link;

L->ptr->link = temp;

// внесение информации в новый элемент

strcpy\_s(temp->data, E);

// новый элемент указывает на следующий

temp->link = L->ptr->link;

// текущий элемент указывает на новый

temp->link = p;

}

/\* Печать всех элементов списка с начала до конца \*/

void printListOne(ListOne\* L) {

// дополнительный элемент для независимого

// прохода от начала до конца списка

element\* tmp;

// ссылка на первый элемент

tmp = L->start->link;

// пока не конец списка

while (tmp != NULL) {

cout << " " << tmp->data;

// печать текущего элемента

tmp = tmp->link;

// независимый переход на следующий элемент

}

cout << endl;

free(tmp); // удаление дополнительного элемента

}

/\* Исключение элемента из списка \*/

void getListOne(ListOne\* L, listOneBaseType\* E) {

// если список пуст

if (!L->ptr->link) {

listOneError = listOneEnd;

return;

}

// временный элемент

element\* pntr = L->ptr->link;

// текущий элемент ссылается на тот, на который ссылается исключаемый

L->ptr->link = pntr->link;

// считывание информации из элемента списка

strcpy\_s(\*E, pntr->data);

// временный элемент ни на что не ссылается

pntr->link = NULL;

// удаление временного элемента

free((void\*)pntr);

}

/\* Предикат: является ли список пустым \*/

int isEmptyListOne(ListOne\* L) {

if (!L->start->link) {

listOneError = listOneEmpty;

return 1;

}

return 0;

}

/\* Очистка списка \*/

void doneListOne(ListOne\* L) {

beginPtrListOne(L);

// установка рабочего указателя в начало

listOneBaseType temp;

// создание временной переменной

while (!isEmptyListOne(L))

// пока список не пуст

getListOne(L, &temp);

// извлечь элемент

free((void\*)L->ptr); // удалить фиктивный элемент

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

ListOne\* List = (ListOne\*)malloc(0);

int n;

int p = 0;

int g = 0;

for (; ;) {

system("cls");

cout << "1 - Создать список" << endl;

cout << "2 - Добавить элемент" << endl;

cout << "3 - Удалить элемент" << endl;

cout << "4 - Вывод" << endl;

cout << "5 - Удалить список" << endl;

cout << "6 - Выход" << endl;

cin >> n;

if (n < 1 || n>6) {

cout << "Повторите ввод" << endl;

cin >> n;

}

switch (n)

{

case 1: {

system("cls");

initListOne(List);

cout << "Список содан!" << endl;

p++;

system("pause");

}break;

case 2: {

system("cls");

if (p > 0) {

int u = 0;

cout << "Ввод в начало или после указанного элемент?\n";

cout << "1 - В начало 2 - После указанного\n";

cin >> u;

while (u > 2 || u < 1) {

cout << "Повторите ввод: ";

cin >> u;

}

g++;

if (u == 1) {

cout << "Ввод: ";

char d[20];

cin >> d;

putListOne(List, d);

}

else {

int t = 0;

cout << "Выберите после какого вставить элемент(" << g << ")" << endl;

cin >> t;

while (t > g || t < 1) {

cout << "Повторите ввод(" << g << ")" << endl;

cin >> t;

}

for (int i = 1; i < t; i++) {

movePtrListOne(List);

}

cout << "Ввод: ";

char d[20];

cin >> d;

putListOne2(List, d);

beginPtrListOne(List);

}

system("pause");

}

else {

break;

}

}break;

case 3: {

system("cls");

if (p > 0) {

char i[20];

int t = 0;

cout << "Выберите какой элемент удалить(" << g << ")" << endl;

cin >> t;

if (t > g || t < 1) {

cout << "Повторите ввод(" << g << ")" << endl;

cin >> t;

}

g--;

for (int i = 1; i < t; i++) {

movePtrListOne(List);

}

getListOne(List, &i);

beginPtrListOne(List);

system("pause");

}

else {

break;

}

}break;

case 4: {

system("cls");

if (p > 0) {

cout << "Список после добавления элементов:" << endl;

printListOne(List); // печать списка

system("pause");

}

else {

break;

}

}break;

case 5: {

system("cls");

if (p > 0) {

doneListOne(List);

p = 0;

g = 0;

system("pause");

}

else {

break;

}

}break;

case 6: {

return 0;

}break;

}

}

}