МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

### федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

(ГУАП)

КАФЕДРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ И СЕТЕЙ

Преподаватель

канд. техн. наук, доцент Л.Н. Бариков

Отчёт

по лабораторной работе №14

по дисциплине ОСНОВЫ ПРОГРАММИРОВАНИЯ

на тему: «Линейные списки»

Работу выполнил

студент гр. 4141 В.С. Сыворотнев

Санкт-Петербург

2022

***Цель лабораторной работы:*** *изучение способов создания и принципов использования односвязных линейных списков; изучение стандартных средств языка C/C++ для работы с динамической памятью; совершенствование навыков модульного программирования на языке C/C++ при решении задач обработки линейных списков; изучение способов разработки многофайловых проектов.*

***Задание на программирование:*** *используя технологию процедурного программирования разработать программу обработки односвязных линейных списков с числом элементов в списке* *не менее десяти в соответствии с индивидуальным заданием.*

***Вариант № 31:***

*Заменить в списке L последнее вхождение элемента со значением E1 на E2.*

***Порядок выполнения работы:***

1. Получить у преподавателя индивидуальное задание.

2. Провести структуризацию задачи. С этой целью выделить подзадачи, которые будут реализованы в виде отдельных функций. При этом запрещается совмещать в одной функции решение нескольких подзадач.

3. Составить описание процесса решения задачи.

4. Построить схему алгоритма решения задачи (функции *main*()) с использованием функций создания, просмотра, обработки списка, удаления списка из динамической памяти.

5. Обосновать перечень и типы параметров всех функций.

6. Построить схему алгоритма функции обработки списка.

7. Создать многофайловый проект на языке *C*/*C*++.

8. Проверить и продемонстрировать преподавателю работу программы на полном наборе тестов. Обеспечить одновременный показ на экране исходного и результирующего списков.

9. Выходные данные должны выводиться на экран с пояснениями. Операторы вывода результатов работы должны находиться либо в функции *main*(), либо в специальной функции вывода (например, преобразованного списка), вызов которой осуществляется из функции *main*().

10. Оформить отчет о лабораторной работе в составе: постановка задачи, описание процесса решения, схемы алгоритмов функции *main*(0 и функции обработки списка, текст всех модулей проекта, контрольные примеры (скриншоты).

11. Текст программы в отчете не должен представлять из себя скриншот.

12. Скриншоты тестов должны легко читаться. Все их неинформативные части должны быть удалены.

***Описание процесса решения***

В разрабатываемый проект будут входить три файла:

- заголовочный файл *modSp*.*h*, содержащий определения новых типов и объявления функций;

- исходный файл *modSp*.*cpp*, содержащий реализацию набора функций для обработки списка;

- основной файл *spis*.*cpp* с программой (функция *main*()).

При решении задачи, а значит в тексте проекта, кроме функции *main*() необходимо реализовать следующие функции:

- функцию инициализации списка;

- функцию добавления нового элемента в конец списка;

- функцию обработки списка в соответствии с заданием;

- функцию просмотра содержимого списка;

- функцию удаления списка из динамической памяти.

Конкретный тип значений информационной части элементов списка в задании не указан, поэтому для получения легко модернизируемой программы с целью изменения типа обрабатываемых данных определяем новый тип с использованием *typedef*:

- тип значений информационной части элементов списка (*telem*).

Определяем новый тип - структуру *list*, которая будет описывать элементы списка. У неё два поля: информационное поле типа *telem* и поле адреса - указатель на следующий элемент списка (*list*\*).

Эти определения новых типов помещаем в заголовочный файл *modSp*.*h*. Сюда же будут помещены объявления функций, необходимых для решения задачи.

Функцию инициализации списка называем *initList*(*list*\*\*,*list*\*\*). Тип возвращаемого функцией значения - *void*. В задании не сказано, как должен формироваться список: подключением новых элементов к голове или к хвосту списка. Выбираем способ формирования с подключением новых элементов к концу списка. Поэтому функция инициализации будет иметь два параметра типа “указатель на указатель на тип *list*”. Первый хранит адрес адреса головного элемента списка, а второй – адрес адреса концевого элемента. Под инициализацией списка будем понимать присваивание адресам головного и концевого элементов списка значения *nullptr* (т.е. список пуст).

Функцию добавления нового элемента в конец списка называем *addElementToTheEnd*(*telem*,*list*\*\*,*list*\*\*). Тип возвращаемого функцией значения - *void*. У неё три параметра: значение информационной части нового элемента списка типа *telem* и два параметра типа “указатель на указатель на тип *list*” (адреса головного и концевого элементов списка будут меняться при подключении нового элемента и передаются через указатели).

Функцию обработки списка называем *processing*(*list*\*, char, char). В соответствии с условиями задачи адрес головного и концевого элементов при обработке измениться не могут. Тип возвращаемого функцией значения - *void*. У неё 3 параметра, первый типа “указатель на тип *list*”, второй типа Char и третий тоже. В функцию передаётся значение адреса головного элемента списка.

Функцию просмотра содержимого списка называем *viewList*(*list*\*). Тип возвращаемого функцией значения - *void*. У неё один параметр типа “указатель на тип *list*”. В функцию передаётся значение адреса головного элемента списка. Под просмотром содержимого списка здесь понимается вывод пользователю на экран значений информационных частей всех элементов списка.

Функцию удаления списка из динамической памяти называем *deleteList*(*list*\*\*,*list*\*\*). Тип возвращаемого функцией значения - *void*. У неё два параметра типа “указатель на указатель на тип *list*” (адреса головного и концевого элементов списка будут меняться при удалении списка из динамической памяти), при этом инициализация списка сохраняется.

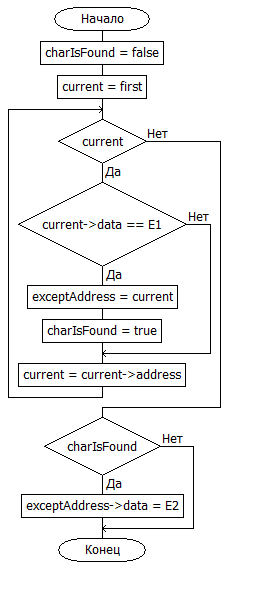
На этом формирование заголовочного файла *modSp*.*h* заканчивается.

**В исходный файл *modSp*.*cpp*** помещаем реализацию набора функций для работы со списком.

Остановимся подробнее на разработке функции *processing*() (реализации процесса решения задачи). Реализация остальных функций для работы со списком была рассмотрена в разделе 3.6.

Согласно заданию на разработку необходимо в созданном списке заменить заданный символ E1, если такой имеется в списке, на заданный символ E2. Для этого в цикле while(current), пока в списке есть элементы, мы рассматриваем каждый из них и сравниваем со значением E1. Если значения совпадают, запоминаем адрес. Повторяем эту операцию до тех пор, пока не пройдем весь список, причем по завершении цикла в переменной будет находится адрес крайнего элемента, т.к. мы запоминаем значение при совпадение каждый раз заново. По окончании цикла смотрим на флаг, если адрес запомнен, то меняем значение крайнего элемента на E2.

***Схема алгоритма функции обработки (processing)***



**В основной файл *spis*.*cpp*** помещаем функцию *main*(). Исполнение программы (функции *main*()) начинается с объявления переменных. Это указатели на первый и последний элементы списка *first* и *last* , переменная *listElements* типа *telem,* а также переменные типа char для хранения значений E1 и E2.

Инициализируем исходный список. Для этого в функцию *main*() помещаем вызов функции *initList*(), параметрами которой являются адреса указателей на первый и последний элементы списка (&*first*, &*last*).

Приступаем к созданию линейного списка. Поскольку линейный список – это динамическая структура, пользователю выводится подсказка с указанием признака окончания ввода. Если у элементов списка символьная информационная часть, то для примера оговариваем, что признаком окончания ввода значений является символ точка ‘.’. Организуем цикл *while*(*listElements* != '.').

В тело этого цикла помещаем вызов функции добавления нового элемента в список *add*\_*spis*(), передавая ей в качестве параметров значение очередного считанного символа *ch* и адреса указателей на первый и последний элементы списка (&*first*, &*end*).

После выхода из цикла проверяем факт создания списка, сравнивая значение указателя на первый элемент списка со значением *NULL*. Если список не создан, выполнение программы заканчивается (*return* 0).

Если список создан, выводим значения его элементов, вызывая функцию *view\_spis*(), параметром которой является значение указателя на первый элемент созданного списка *first*.

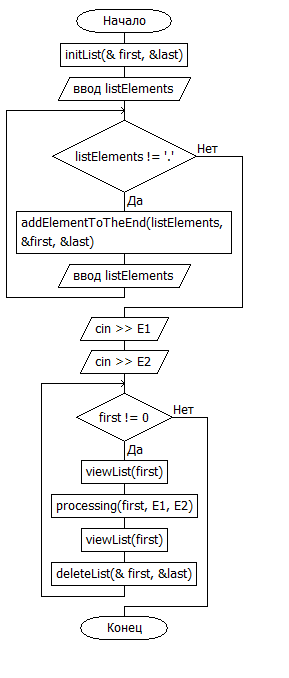
Проверка создания списка выполнена. Приступаем к обработке созданного списка. Вызываем функцию *obrabotka*(), передавая ей в качестве параметра значение указателя на первый элемент списка *first*.

После обработки списка выводим значения его элементов, вызывая функцию *view\_spis*(), параметром которой является значение указателя на первый элемент обработанного списка *first*.

Программа завершается. Удаляем список из динамической памяти, вызывая функцию *udal\_spis*(), передавая ей в качестве параметров значения адресов указателей на первый и последний элементы списка (&*first*, &*end*).

Анализируем результаты работы проекта и делаем выводы.

Схема алгоритма функции *main*() приведена на рисунке.

***Схема алгоритма функции main()*** 

***Текст программы***

***Заголовочный файл modSp.h***

#include <iostream>

//Определения типов

typedef char telem; //определение типа информационного поля

#include "windows.h"

struct list { //определение типа элемента списка

telem data; //информационное поле

list\* address; //поле адреса

};

//Объявления (прототипы) функций

void initList(list\*\* first); //инициализация списка

void addElementToTheEnd(telem listElements, list\*\* first); //добавление элемента

void processing(list\* first, telem E1, telem E2); //обработка списка

void viewList(list\* current); //просмотр списка

void deleteList(list\*\* first); //освобождение памяти

***Основной файл spis.cpp***

*/\*Основной файл spis.cpp*

*Создать линейный список L.*

*Заменить в списке L последнее вхождение элемента со значением E1 на E2.*

*Многофайловый проект\*/*

#include "modSp.h"

using namespace std;

//основная программа

int main() {

list\* first;

telem listElements;

telem E1;

telem E2;

initList(&first);

SetConsoleOutputCP(CP\_UTF8);

//создание исходного списка

cout << "**\n** Заменить в списке L последнее вхождение элемента со значением E1 на E2." << endl;

cout << "**\n** Введите значение элемента E1: ";

cin >> E1;

cout << "**\n** Введите значение элемента E2: ";

cin >> E2;

cout << " Введите элементы списка сплошной строкой;"

<< " в конце - точка:" << endl;

cin >> listElements;

while (listElements != '.') {

addElementToTheEnd(listElements, &first);

cin >> listElements;

}

//вывод исходного списка

if (first && first->address && first->address->address) { //список создан правильно

cout << " Исходный список:" << endl;

viewList(first);

//обработка списка

processing(first, E1, E2);

cout << "**\n**Результат получен:**\n**";

viewList(first);

//освобождение памяти

deleteList(&first);

}

return 1;

}

}

***Исходный файл modSp.cpp***

*/\*Файл modSp.cpp*

*Исходный файл*

*Реализация набора функций для обработки списка\*/*

#include "modSp.h"

using namespace std;

//Обработка списка

void processing(list\* first, telem E1, telem E2) {

list\* exceptAddress;

exceptAddress = nullptr;

while (first) {

if (first->data == E1) {

exceptAddress = first;

}

first = first->address;

}

if (exceptAddress) {

exceptAddress->data = E2;

}

}

//Просмотр содержимого списка

void viewList(list\* current) {

do {

cout << current->data;

current = current->address;

} while (current);

}

//Добавление нового элемента в конец списка

void addElementToTheEnd(telem listElements, list\*\* first) {

list\* tmp;

list\* newElement = new list; //выделяем память под элемент списка

newElement->data = listElements; //заполняем информационную часть

newElement->address = nullptr; //в ссылочную - NULL

if (\*first) { //если список не пуст

tmp->address = newElement;

} else {

\*first = newElement; //если список пока пуст

}

tmp = newElement;

}

//Инициализация списка

void initList(list\*\* first) {

\*first = nullptr;

}

//Освобождение динамической памяти

void deleteList(list\*\* first) {

list\* current;

while (\*first) {

current = \*first; //запоминаем его адрес

\*first = (\*first)->address; */\*удаляем из списка и переходим*

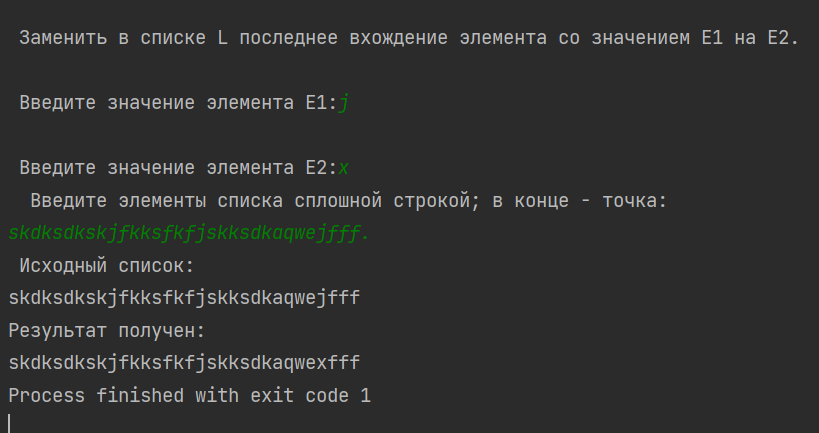
*к следующему элементу\*/*

delete current; //освобождаем память

}

}

***Скриншот результатов выполнения программы***

******

***Вывод:*** *я изучил способы создания и принципы использования односвязных линейных списков; изучил стандартные средства языка C/C++ для работы с динамической памятью; усовершенствовал навыки модульного программирования на языке C/C++ при решении задач обработки линейных списков; изучил способы разработки многофайловых проектов.*