МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ОБНИНСКИЙ ИНСТИТУТ АТОМНОЙ ЭНЕРГЕТИКИ — филиал
федерального государственного автономного образовательного учреждения
высшего профессионального образования
«Национальный исследовательский ядерный университет «МИФИ»
(ИАТЭ НИЯУ МИФИ)

Факультет кибернетики
Кафедра компьютерных систем, сетей и технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
Линейный список

По курсу «Объектно-ориентированное программирование»

Студенты Жеребилова А.В.
группы ..... Максимов Р.А.
ВТ-С-Б12 Балицкий Д.

Руководитель
доцент ..... Тельнов В.П.
кафедры КССТ

Обнинск 2014

1 Постановка задачи

Разработайте в MS Visual Studio программное решение на языке Си, которое реализует динамическую структуру данных (контейнер) типа «Линейный список». Каждый элемент контейнера содержит строки символов произвольной длины.

В программном решении следует реализовать следующие операции над контейнером:

- создание и уничтожение контейнера;
- добавление и извлечение элементов контейнера;
- обход всех элементов контейнера (итератор);
- удаление из контейнера дублирующих элементов;
- вычисление количества элементов в контейнере;
- реверс контейнера (первый элемент контейнера становится последним, второй элемент становится предпоследним и т.д.);
- объединение, пересечение и вычитание контейнеров;
- сохранение контейнера в дисковом файле и восстановление контейнера из файла.

Ограничения. Реализуйте простейший проект типа «приложение командной строки» (т.е. без оконного интерфейса). Средства C++ (объекты, классы, шаблоны классов) использовать не следует. Готовые контейнерные классы из библиотеки STL также использовать не следует. Разработайте контейнер самостоятельно на языке Си.

Рекомендации. Начните работу с изучения wiki:
https://github.com/djbelyak/OOPLab-List/wiki

Найдите и изучите в рекомендованной литературе и в документации MS Visual Studio описания и примеры реализаций данной структуры данных. Обдумайте и обсудите с преподавателем алгоритмы, состав функций, интерфейс и общую структуру программы. Возникающие затруднения пытайтесь преодолеть самостоятельно, потом обращайтесь за помощью.

Письменный отчет по работе должен содержать следующие разделы:

1. Постановку задачи.
2. Описание контейнера как динамической структуры данных, в том числе:
   - рисунки, на которых изображена структура данных и поясняются основные алгоритмы;
   - описание алгоритмов, которые используются при работе с контейнером;
   - область применения данной структуры данных, её преимущества и недостатки.

3. Листинг разработанного авторского кода на языке Си. Код должен быть надлежащим образом структурирован и снабжен комментариями.

Для успешной сдачи лабораторной работы необходимо представить письменный отчет, продемонстрировать на практике работоспособность программного решения и ответить на вопросы преподавателя.

2 Описание контейнера

Линейный односвязный список. Линейный односвязный список – это динамическая упорядоченная структура данных, состоящая из элементов одного типа. Каждый элемент содержит основное поле-указатель на объект и поле с указателем на следующий элемент. Элементы не всегда расположены в памяти последовательно.

Преимуществом линейного односвязного списка как динамической структуры данных является простота добавления нового элемента, удаления первого элемента. Многие операции можно свести к работе с указателями, при этом сами объекты остаются на месте. Это позволяет сократить время обработки, так как объекты могут быть большими сложными. Также для односвязного списка характерна простота устройства.

Недостатком односвязного списка по сравнению с двусвязным является затруднённый доступ к предыдущему элементу, из-за чего усложняются операции извлечения элемента из произвольного места и сортировки. По сравнению с хеш-таблицей операция поиска в большом списке идёт медленнее. Сортировка в списке уступает по времени особой концепции сортировки двоичного дерева.

Линейный односвязный список рационально использовать там, где требуется хранить не слишком большое количество однотипных данных, если при этом предполагается частое удаление и добавление новых объектов (например, небольшой список номеров телефонов). На его основе создаётся структура данных «Стек».

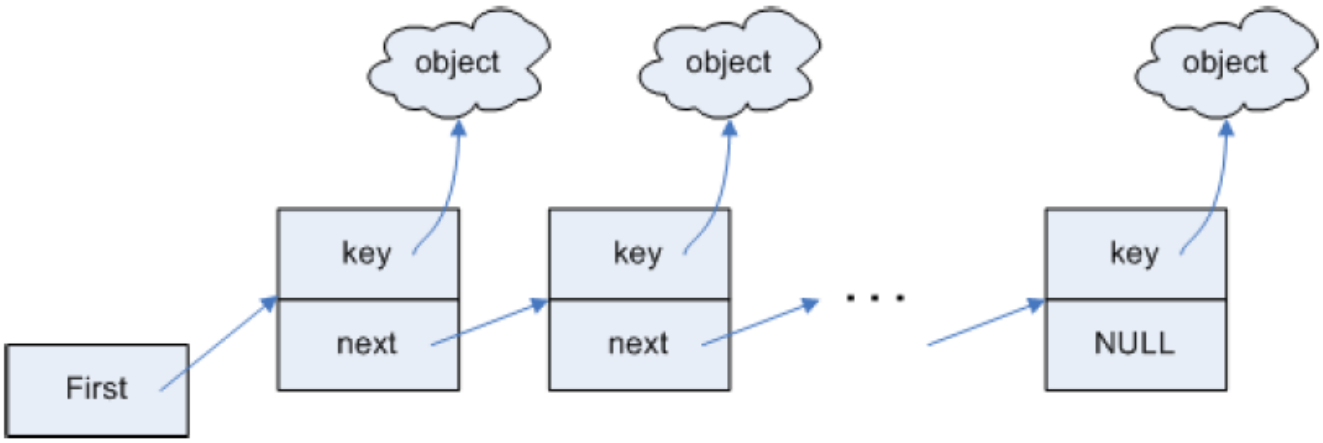

Рисунок 1 – Структура линейного односвязного списка.

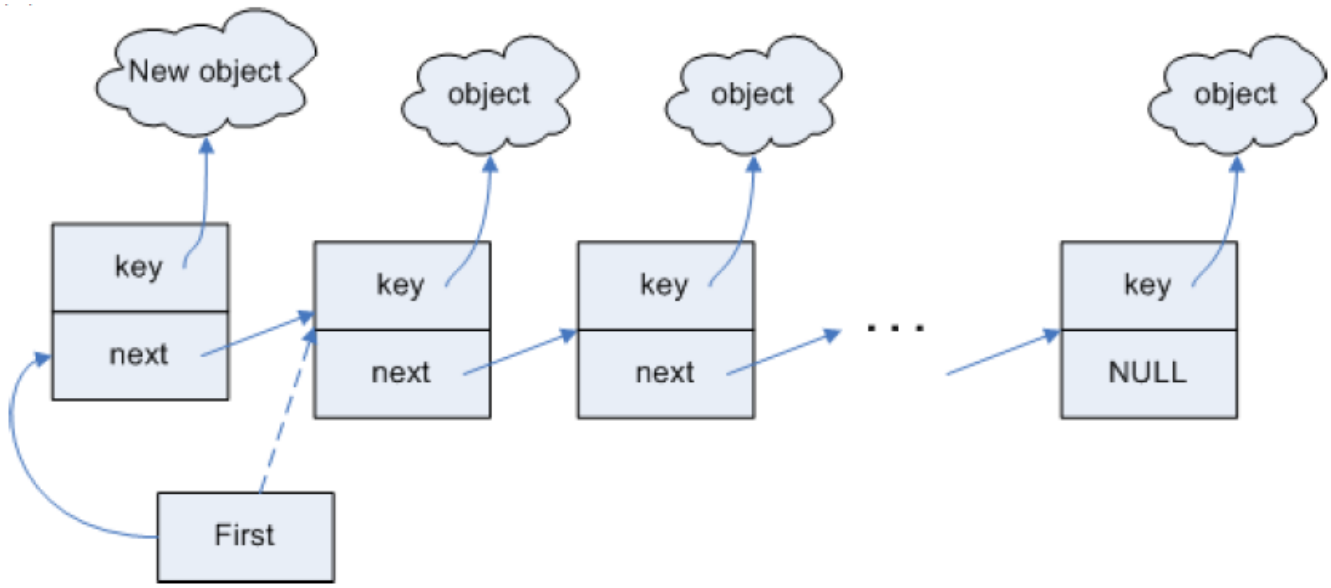

Рисунок 2 – Структура линейного односвязного списка.

3 Листинг исходного кода

Listing 1 – container.h

```
#include <tchar.h>
#include <stdio.h>
#include <locale.h>
#include <iostream>
#include <conio.h>


const int MAX_LEN_STR = 50;
const int MAX_LEN_FILENAME = 24;
const int DIR_VIEW = 1;
const int BACK_VIEW = -1;

struct List
{
        char* data;
        List* next;
};

void AddList(List*&, char*);
int DeleteList(List*&, List*&);
void DeleteList(List*& Point_First);
void DeleteDblLists(List*);
```

```
void Add_Items();
void Extracting_Elements();
void ReversContain();
int SaveToFile(List*, FILE*);
int ExtractingFile(List*&, FILE*);
void Menu1();
void Menu2(List*&);
void Menu3(List*&, List*&);
List* LinkLists(List*, List*);
List* SubtrLists(List*, List*);
List* CrossLists(List*, List*);
int NumberOfElements(List*);
void BypassAllElements(List*, int);
```

Listing 2 – container.cpp

```
#include "container.h"


void DeleteList(List*& Point_First)
{
        while (Point_First != 0)
                DeleteList(Point_First, Point_First);
}

void AddList(List*& Point_First, char* Str_buf)
{
        List* P = new List;
        P->data = new char[strlen(Str_buf)];
        strcpy(P->data, Str_buf);
        P->next = Point_First;
        Point_First = P;
}

int DeleteList(List*& Point_First, List*& Point_Current)
{
        if (Point_First == 0) return 1;
        if (Point_Current == 0) return 2;
        List* P;

        if (Point_Current == Point_First)
        {
```

```
                P = Point_Current;
                Point_First = Point_First->next;
                Point_Current = Point_First;
                delete P;
                return 0;
        }

        P = Point_First;
        while (P->next != Point_Current)
        {
                P = P->next;
                if (P == 0) return 2;
        }
        P->next = Point_Current->next;
        P = Point_Current;
        Point_Current = Point_Current->next;
        delete P;
        return 0;
}

void DeleteDblLists(List* Point_First)
{
        List* P2;
        for(; Point_First && Point_First->next; Point_First = Point_First->nex
                while(!FindList(Point_First->next, P2, Point_First->data))
                        DeleteList(Point_First->next, P2);
}

void Add_Items()
{
        do AddList(gets(buffer));
        while(strlen(buffer)>1);
}

void Extracting_Elements()
{
        List* Point_First;
        while(Point_First)
        {
                puts(Point_First->data);
```

```
                Point_First=Point_First->next;
        }
}

List* LinkLists(List* Point_First1, List* Point_First2)
{
        List* pCFirst1 = CopyList(Point_First1);
        List* pCFirst2 = CopyList(Point_First2);
        List* pCLast1 = pCFirst1;
        if (pCLast1 != 0)
        {
                while (pCLast1->next != 0)
                        pCLast1 = pCLast1->next;
                pCLast1->next = pCFirst2;
        }
        else
                pCFirst1 = pCFirst2;
        return pCFirst1;
}

List* SubtrLists(List* Point_First1, List* Point_First2)
{
        List* Point_NewFirst = 0;
        List* P2;
        for (; Point_First1; Point_First1 = Point_First1->next)
                if (FindList(Point_First2, P2, Point_First1->data) != 0)
                        AddList(Point_NewFirst, Point_First1->data);
        return Point_NewFirst;
}

List* CrossLists(List* Point_First1, List* Point_First2)
{
        List* Point_NewFirst = 0;
        List* P2;
        for (; Point_First1; Point_First1 = Point_First1->next)
                if (FindList(Point_First2, P2, Point_First1->data) == 0)
                        AddList(Point_NewFirst, Point_First1->data);
        return Point_NewFirst;
}
```

```
void ReversContain()
{
        List* Point_First;
        char* Point_Adress[]={NULL};
        int i=0,j,k;
        while(Point_First)
        {
                Point_Adress[i]=Point_First->data;
                Point_First=Point_First->next;
        }
        for(j=i;j<=0;j--)
        {
                Point_First->data=Point_Adress[j];
                Point_First=Point_First->next;
        }
}

int SaveToFile(List* Point_First, FILE* F)
{
        if (F == 0) return 1;
        while (Point_First != 0)
        {
                fputs(Point_First->data, F);
                fputc('\n', F);
                Point_First = Point_First->next;
        }
        fputc('\0', F);
        return 0;
}

int ExtractingFile(List*& Point_First, FILE* F)
{
        if (F == 0)
                return 1;
        char strBuf[MAX_LEN_STR];
        int i;

        Point_First = new List;
        fgets(strBuf, MAX_LEN_STR, F);
        strBuf[strlen(strBuf)-1] = '\0';
```

```
        Point_First->data = new char [strlen(strBuf)];
        strcpy(Point_First->data, strBuf);
        List* P = Point_First;
        P->next = 0;

        for(;;)
        {
                if (*(fgets(strBuf, MAX_LEN_STR, F)) == '\0') return 0;
                strBuf[strlen(strBuf)-1] = '\0';
                P->next = new List;
                P = P->next;
                P->data = new char [strlen(strBuf)];
                strcpy(P->data, strBuf);
                P->next = 0;
        }
        return 0;
}

int NumberOfElements(List* Point_First)
{
        int i = 0;
        for (; Point_First; Point_First = Point_First->next)
                i++;
        return i;
}

void BypassAllElements(List* Point_First, int Direction)
{
        if (Direction == BACK_VIEW)
        {
                if (Point_First != 0)
                {
                        BypassAllElements(Point_First->next, BACK_VIEW);
                }
        }
        else
        {
                while (Point_First != 0)
                {
                        Point_First = Point_First->next;
```

```
                }
        }
}
```

Listing 3 – main.cpp

```
#include "container.h"

int main()
{
        setlocale(LC_CTYPE, "rus");
        Menu1();
        return(0);
}

void Menu1()
{
        setlocale(LC_CTYPE, "rus");
        const char* MENU_1 =
"1 - New list\n2 - Open list\nEsc - Exit\n";

        char data;
        char FileName[MAX_LEN_FILENAME];
        List* pL1;
        FILE* F;

        for(;;)
        {
                system("cls");
                puts(MENU_1);
                pL1 = 0;
                data = getch();
                switch (data)
                {
                case '1':
                        {
                                Menu2(pL1);
                                break;
                        }
                case '2':
                        {
                                system("cls");
```

```
                                printf("Enter the file name: \n");
                                gets(FileName);
                                F = fopen(FileName,"r");
                                if (!F)
                                {
                                        printf("Error opening file");
                                        getch();
                                        break;
                                }
                                ExtractingFile(pL1, F);
                                fclose(F);
                                Menu2(pL1);
                                break;
                        }
                case 27:
                        {
                                return;
                        }
                default:
                        {
                                system("cls");
                                printf("Error!\nRetype!");
                                getch();
                                break;
                        }
                }
        }
}

void Menu2(List*& pL1)
{
        setlocale(LC_CTYPE, "rus");
        const char* MENU_2 =
"______\n\n \
1 - Save\t\t 8 - Change the direction of the display\n \
2 - Add\t\t 9 - Merge\n \
3 - Extract\t\ 
4 - Find\t\t 10 - Cross\n \
5 - Counting\t 0 - Deduct\n \
6 - Sorting \t\t Esc - Copy\
```

```
7␣-␣Remove␣duplicate\n";

        List* pL2;
        List* pL3;
        List* Point_Current;
        char data, FileName[MAX_LEN_FILENAME], strBuf[MAX_LEN_STR];
        int Direction = DIR_VIEW;
        FILE* F;

        for(;;)
        {
                system("cls");
                BypassAllElements(pL1, Direction);
                puts(MENU_2);
                data = getch();
                switch (data)
                {
                case '1':
                        {
                                system("cls");
                                printf("Enter␣the␣file␣name:␣\n");
                                gets(FileName);
                                if (!(F = fopen(FileName, "w")))
                                {
                                        printf("Error␣opening␣file\n\nRetype!'
                                        getch();
                                        break;
                                }
                                SaveToFile(pL1, F);
                                fclose(F);
                                break;
                        }
                case '2':
                        {
                                system("cls");
                                printf("Enter␣string:␣\n");
                                Add_Items();
                                break;
                        }
                case '3':
```

```
                system("cls");
                    Extracting_Elements();
                    break;

        case '4':
                {
                    system("cls");
                    printf("Enter string: \n");
                    gets(strBuf);
                    switch (FindList(pL1, Point_Current, strBuf))
                    {
                    case 1:
                            {
                                system("cls");
                                printf("List is empty\n\nPress
                                getch();
                                break;
                            }
                    case 2:
                            {
                                system("cls");
                                printf("No such element\n\nPr
                                getch();
                                break;
                            }
                    default:
                            Menu3(pL1, Point_Current);
                    }
                    break;
                }
        case '5':
                {
                    system("cls");
                    printf("%i List\n\nPress any key", CountLists
                    getch();
                    break;
                }
        case '6':
                {
                    SortList(pL1);
```

```
                    break;
                }
        case '7':
                {
                    DeleteDblLists(pL1);
                    break;
                }
        case '8':
                {
                    Direction = -(Direction);
                    break;
                }
        case '9':
                {
                    system("cls");
                    printf("Enter the file name: \n");
                    gets(FileName);
                    F = fopen(FileName, "r");
                    if (!F)
                    {
                            printf("Error\n\nPress any key");
                            getch();
                            break;
                    }
                    ExtractingFile(pL2, F);
                    fclose(F);
                    pL3 = LinkLists(pL1, pL2);
                    DeleteList(pL2);
                    Menu2(pL3);
                    break;
                }
        case '10':
                {
                    system("cls");
                    printf("Enter the file name: \n");
                    gets(FileName);
                    F = fopen(FileName, "r");
                    if (!F)
                    {
                            printf("Error\n\nPress any key");
```

```
                        getch();
                        break;
                }
                ExtractingFile(pL2, F);
                fclose(F);
                pL3 = CrossLists(pL1, pL2);
                DeleteList(pL2);
                Menu2(pL3);
                break;
        }
case '0':
        {
                system("cls");
                printf("Enter the file name: \n");
                gets(FileName);
                F = fopen(FileName, "r");
                if (!F)
                {
                        printf("Error\n\nPress any key");
                        getch();
                        break;
                }
                ExtractingFile(pL2, F);
                fclose(F);
                pL3 = SubtrLists(pL1, pL2);
                DeleteList(pL2);
                Menu2(pL3);
                break;
        }
case 27:
        {
                DeleteList(pL1);
                return;
        }
default:
        {
                system("cls");
                printf("Wrong command\n\nPress any key");
                getch();
                break;
```

```
                        }
                }
        }
}

void Menu3(List*& pL1, List*& Point_Current)
{
        setlocale(LC_CTYPE, "rus");
        const char* MENU_3 =
"______\n\n \
1 - Remove the list\n \
Esc - Back\n";

        char data;

        for(;;)
        {
                system("cls");
                puts(Point_Current->data);
                puts(MENU_3);
                data = getch();
                switch (data)
                {
                case '1':
                        {
                                DeleteList(pL1, Point_Current);
                                return;
                        }
                case 27:
                        {
                                return;
                        }
                default:
                        {
                                system("cls");
                                printf("Invalid instruction\n\nPress any key"
                                getch();
                                break;
                        }
                }
```

```
        }
    }
```