**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ОБНИНСКИЙ ИНСТИТУТ АТОМНОЙ ЭНЕРГЕТИКИ** – **филиал**

федерального государственного автономного образовательного учреждения высшего профессионального образования

**«Национальный исследовательский ядерный университет «МИФИ»»**

Факультет кибернетики

Кафедра компьютерных систем, сетей и технологий

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1**

**Линейный список**

По курсу «Объектно-ориентированное программирование»

|  |  |  |
| --- | --- | --- |
| Студенты  группы  ВТ-С-Б12 | …………………………………………………….. | Жеребилова А.В.  Максимов Р.А.  Балицкий Д. |
| Руководитель  доцент  кафедры КССТ | …………………………………………………….. | Тельнов В.П. |

Обнинск 2014

1

**Постановка задачи**

Разработать в MS Visual Studio программное решение на языке Си, которое реализует динамическую структуру данных (контейнер) типа «линейный односвязный список». Каждый элемент контейнера содержит строки символов произвольной длины. В программном решении следует реализовать следующие операции над контейнером:

* Создание и уничтожение списка
* Поиск, добавление и извлечение элементов контейнера
* Обход всех элементов контейнера в прямом и обратном направлениях (итератор)
* Удаление из контейнера дублирующих элементов
* Вычисление количества элементов в контейнере
* Сортировка элементов контейнера
* Объединение, пересечение и вычитание контейнеров
* Сохранение контейнера в дисковом файле и восстановление контейнера из файла

**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.

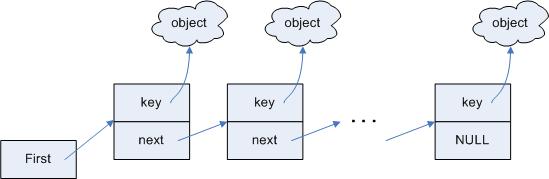
Это позволяет сократить время обработки, так как объекты могут быть большими сложными. Также для односвязного списка характерна простота устройства.

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».

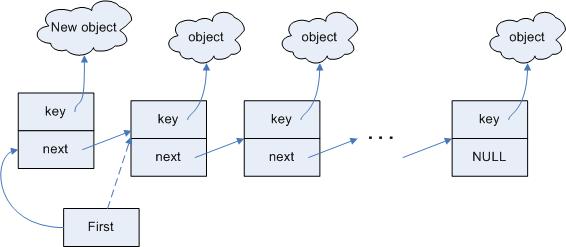
**Рисунки, поясняющие работу со списком**

*Структура линейного односвязного списка*



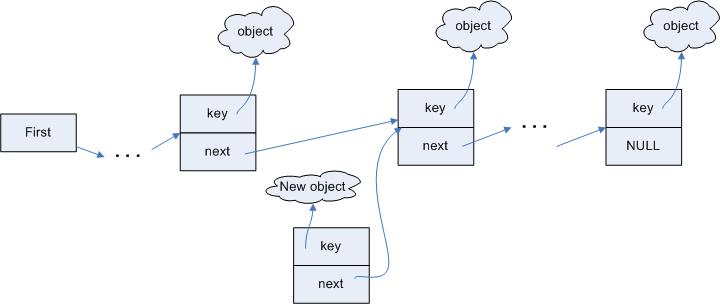
2

*Добавление элемента в начало списка*

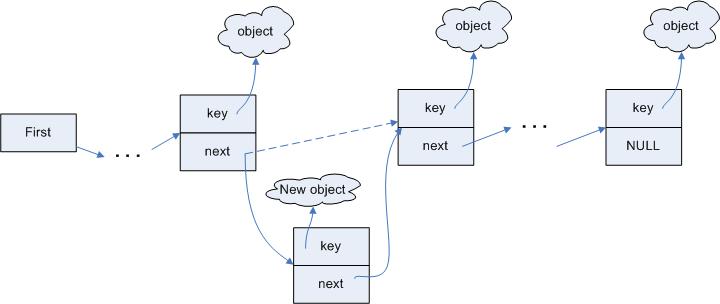


*Добавление элемента в произвольное место в списка*

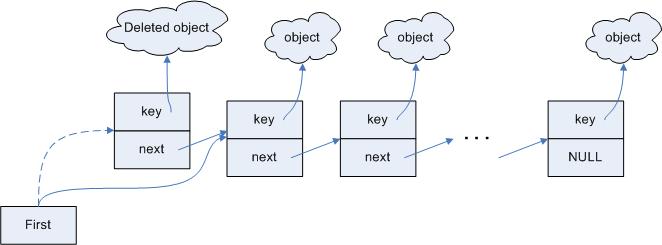
*1.*



*2.*

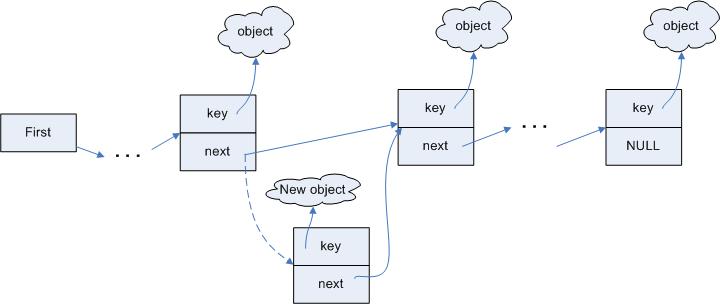


*Удаление элемента из начала списка*



3

*Удаление элемента из произвольного места в списке*



**Основные алгоритмы работы с односвязным линейным списком**

1. *Проход по списку в прямом направлении (итеративный).*

Указатель на элемент списка изначально указывает на начало. Двигаемся по списку, присваивая элементу значение следующего за ним, пока этот указатель не получит значение NULL. На каждом шаге совершаем полезные действия над элементом (вывод на экран, подсчёт количества элементов, запись в файл).

1. *Проход по списку в обратном направлении (рекурсивный).*

При помощи рекурсии доходим до конца списка, каждый раз подставляя на место аргумента указатель на следующий элемент. Когда указатель принимает значение NULL, стек вызова функций начинает очищаться в обратном направлении, при этом совершаются полезные действия над элементами (вывод на экран).

1. *Добавление элемента в начало списка.*

Полю next нового элемента присваиваем значение указателя на первый элемент. Указателю на первый элемент присваиваем значение указателя на новый элемент.

1. *Удаление элемента из начала списка.*

Указателю на первый элемент присваиваем значение следующего за ним элемента. Память из-под удалённого элемента можно впоследствии высвободить.

1. *Удаление элемента из произвольного места списка.*

Полю next предшествующего элемента присваиваем указатель на следующий элемент после удаляемого (p->next->next). (Если предшествующий элемент нужно получить проходом в прямом направлении) Так как на удаляемый элемент теперь нет ссылки, то до него нельзя добраться через список. Память из-под него можно высвободить.

1. *Сортировка списка (по алфавиту).*

На первом этапе большая часть списка сортируется по алгоритму выбора. Создаём дополнительный пустой список. Ищем максимальный элемент в исходном списке и переносим его в дополнительный список (так как элемент добавляется в начало, то для сортировки по возрастанию нужно искать именно максимальный элемент). Для корректного удаления максимального элемента из исходного списка нужно получить указатель на предшествующий элемент, а в односвязном списке это затруднено, то оперируем с указателями не на сами элементы, а на следующие за ними (P->next). Поэтому первый элемент не участвует в сортировке выбором.

Далее применяется алгоритм вставки. Проходим по дополнительному списку, пока не встретим элемент, больший первого. Вставляем первый элемент. Возможны ситуации, когда первый элемент является наибольшим или наименьшим. В первом случае указатель, проходящий по списку, принимает значение NULL, и первый элемент вставляется в конец.

4

Во втором случае цикл прохода по списку так и не начинается, первый элемент вставляется в начало. Результатом сортировки является указатель на начало дополнительного списка.

**Структурная схема программы**

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);

**Структура папок проекта**

Laba 1

 Debug

 ipch

* 1. laba 1-53500306
* Laba 1
  1. Debug

**Руководство пользователя**

Для начала работы с программой нужно запустить файл Laba 1.exe. Появится окно консольного приложения с начальным меню. Обычно экран разделён на две области: основная область, отображающая список, элемент или другую информацию, и меню со списком доступных команд. Слева от каждой команды указывается соответствующая клавиша (0 – 9 или Esc). При нажатии клавиши, не соответствующей никакой команде, выводится сообщение Wrong instruction. Начальное меню предлагает создать список (1 - New list) либо загрузить готовый из текстового файла (2 - Open list). (Если возникли проблемы с открытием файла, или указанного файла не существует, будет выведено сообщение Error open file.) При нажатии Esc в начальном меню программа завершает работу.

Меню списка отображает основные команды для работы со списком:

1 – Save list

Сохранение текущего списка на диске в текстовом файле. После выбора этого пункта программа предложит указать имя файла. Для последующей корректной работы с файлом нужно также указать расширение .txt. Имя файла (без учёта расширения) не должно быть длиннее 20 символов.

2 – Add item

Добавление нового элемента в начало списка. Будет предложено ввести строку, которая станет новым элементом списка. Длина строки – не более 50 символов.

3 – Find item

Поиск элемента. Будет предложено ввести строку. Программа выведет первый элемент, содержащий эту строку. Возможные ситуации: No such item – такого элемента нет в списке; Empty list – список пуст.

4 – Count items

Подсчёт количества элементов в списке.

5 – Sort list

Сортировка текущего списка по алфавиту.

5

6 – Delete double items

Удаление повторяющихся элементов из списка.

7 – Change print direction

Смена направления вывода на экран. Можно сменить как с прямого на обратное, так и с обратного на прямое.

8 – Link with other list

9 – Cross with other list

0 – Subtract other list

Объединение, пересечение двух списков и вычитание второго списка из первого (в понимании теории множеств). Будет предложено ввести имя файла, в котором находится второй список. В случае удачного открытия файла отобразится результат выбранной операции – третий список, и его меню. После закрытия третьего списка продолжится работа над первым.

Esc – Close list

Завершение работы со списком, возврат в начальное меню. Все несохранённые данные теряются.

Меню элемента отображает основные команды для работы с элементом (если он был успешно найден при помощи 3 - Find item):

1 – Delete item

Удаление элемента из списка.

Esc – Back

Возврат к списку.

6

**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->next)

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 any key");

getch();

break;

}

case 2:

{

system("cls");

printf("No such element\n\nPress any key");

getch();

break;

}

default:

Menu3(pL1, Point\_Current);

}

break;

}

case '5':

{

system("cls");

printf("%i List\n\nPress any key", CountLists(pL1));

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;

}

}

}

}