Министерство образования РФ

Пермский государственный технический университет

Кафедра ИТАС

ЛАБОРАТОРНАЯ РАБОТА № 8

ПО ИНФОРМАТИКЕ ЗА I СЕМЕСТР

Вариант 9

Выполнил студент:

Главатских Максим Николаевич

Группа РИС-20-1бз

Шифр 20-ЭТФ-659

Кафедра ИТАС:

Полякова Ольга Андреевна

ПЕРМЬ, 2020

# Лабораторная работа №8 "Динамические структуры данных”

**Вариант №9**

**Цель:**

1) Получить практические навыки работы с однонаправленными списками;

2) получить практические навыки работы с двунаправленными списками;

3) получить практические навыки работы с деревьями.

**Задача:**

1. Сформировать однонаправленный список, тип информационного поля указан в варианте.
2. Распечатать полученный список.
3. Выполнить обработку списка в соответствии с заданием.
4. Распечатать полученный список.
5. Удалить список из памяти.
6. Сформировать двунаправленный список, тип информационного поля указан в варианте.
7. Распечатать полученный список.
8. Выполнить обработку списка в соответствии с заданием.
9. Распечатать полученный список.
10. Удалить список из памяти.
11. Сформировать идеально сбалансированное бинарное дерево, тип информационного поля указан в варианте.
12. Распечатать полученное дерево.
13. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.
14. Преобразовать идеально сбалансированное дерево в дерево поиска.
15. Распечатать полученное дерево.

**Вариант задания:**

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | Тип информационного поля int.  Удалить из списка все элементы с четными информационными полями. | Тип информационного поля char\*.  Добавить в список элементы с номерами 1, 3, 5 и т. д. | Тип информационного поля double.  Найти максимальный элемент в дереве. |

**Текст программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <iomanip>

#include <cstring>

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

using namespace std;

char\* fileName;

struct Item

{

Item(int k, const char\* ch, Item\* n = NULL)

:key(k), info(new char[strlen(ch) + 1]), next(n), length(strlen(ch) + 1)

{

strcpy(info, ch); //копирование строчки с информацией в структуру

}

int key;//ключ

char\* info;//информация

Item\* next;//следующий элемент списка

int length;//длина информации

int shift;//смещение в файле (байт)

};

//проверяет принадлежит ли ключ списку

bool Belong(Item\*& list, int key)

{

Item\* temp(list);

while (temp && temp->key != key)//пока есть элементы в списке и ключ текущего элемнта не равен ключу что мы ищем

temp = temp->next;

return (temp ? true : false);//возвращаем true если temp!=NULL, те если ключ найден в списке

}

//добавляет в список элемент

void Add(Item \* &list, int key, const char\* val)

{

if (Belong(list, key))//проверяем нет ли у нас элемента уже с таким ключом

{

cerr << "Element with key = " << key << " already exist\n";

return;

}

if (!list)//если список пуст то добавим в голову списка элемент

list = new Item(key, val, NULL);

else//иначе найдём хвост и добавим после него элемент

{

Item\* temp(list);

while (temp->next)

temp = temp->next;

temp->next = new Item(key, val, NULL);

}

}

//удаление элемента с заданным ключом

bool Delete(Item \* &list, int key)

{

Item\* temp(list);

while (temp && temp->key != key)//ищем элемент с таким ключом как нас просят

temp = temp->next;

if (temp)//если елемент с таким ключом существует

{

//если елемент что нужно удалять это голова списка

if (temp == list)

{

Item\* toDel(list);

list = list->next;

delete toDel;

return true;

}

//иначе элемент где-то в середине или в конце

else

{

Item\* toDel(temp);

Item\* newElem(list);

//найдём элемент перед тем что собрались удалять

while (newElem->next != temp)

newElem = newElem->next;

//уберём удаляемый элемент из списка

newElem->next = temp->next;

delete toDel;//удалим элемент

return true;

}

}

else//иначе элемента нет. Уведомляем пользователя об ошибке

{

cerr << "Can't delete element with key = " << key << endl;

return false;

}

}

//поиск элемента по ключу

Item\* Find(Item \* &list, int key)

{

Item\* result;//результат. Этот указатель мы и вернём пользователю

Item\* temp(list);

while (temp && temp->key != key)//цикл происходит пока не будет найдено совпадение ключа (или пока не кончится список)

temp = temp->next;

if (temp)//если список не кончился - мы нашли ключ, скопируем этот элемент и вернём его как результа

return (result = new Item(temp->key, temp->info, NULL));

else//иначе список закончился быстрее чем мы нашли нужный элемент. Уведомим пользователя об ошибке и вернём указатель на NULL

cerr << "Can't find element with key = " << key << endl;

return (result = NULL);

}

//Удаление всего списка, необходимо чтобы можно было освободить память

void DeleteList(Item \* &list)

{

Item\* toDel(list);

while (list)//пока есть что удалять - удаляем элементы

{

list = list->next;

delete toDel;

toDel = list;

}

}

//вывод в поток списка

ostream& operator<<(ostream & os, const Item \* print)

{

const Item\* temp(print);

if (!temp)//если элементов в списке нет (list = NULL) то не будем ничего пытаться печатать

os << "List is empty.";

else//иначе распечатаем построчно ключи и информационную часть элементов

while (temp)

os << temp->key << " -> " << temp->info << '\n', temp = temp->next;

return (os << endl);//вернём адрес потока чтобы пользователь мог туда ещё что-то засунуть (позволяет делать так cout<<list1<<list2<<list3)

}

void WriteList(Item \* &list)

{

FILE\* file = fopen(fileName, "w");

Item\* temp = list;

int Shift = 0;

int\* count = new int(0);

while (temp)

temp = temp->next, count++;

Shift += sizeof(Item) \* (\*count);

temp = list;

while (temp)

{

temp->shift = Shift;

Shift += temp->length;

temp = temp->next;

}

temp = list;

fwrite(count, 1, sizeof(int), file);

while (temp)

{

fwrite(temp, 1, sizeof(Item), file);

temp = temp->next;

}

temp = list;

while (temp)

{

fwrite(temp->info, temp->length, sizeof(char), file);

temp = temp->next;

}

fclose(file);

}

Item\* ReadList()

{

int\* count = new int(-1);

Item \* lst = NULL;

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

FILE \* file = fopen(fileName, "r");

fread(count, 1, sizeof(int), file);

for (int i = 0; i < (\*count); i++)

{

fread(temp, 1, sizeof(Item), file);

temp->info = new char[temp->length];

fseek(file, temp->shift, SEEK\_SET);

fread(temp->info, temp->length, sizeof(char), file);

Add(lst, temp->key, temp->info);

fseek(file, sizeof(int) + (i + 1) \* sizeof(Item), SEEK\_SET);

}

return lst;

}

int main()

{

fileName = new char[strlen("out.txt") + 1];

strcpy(fileName, "out.txt");

Item\* lst = NULL;

Add(lst, 2, "value2");

Add(lst, 3, "zzzzzz4");

//Add(lst,3,"value3");

cout << lst;

WriteList(lst);

Item\* lst2 = ReadList();

cout << lst2;

}

/\*

int main()

{

int choice=1;//выбор пользователя. Сюда считывается пункт меню что выбрал пользователь

int key;//ключ который ввёл пользователь при добавлении/удалении/поиске

char str[256];//строка для информации что будет вводить пользователь при добавлении/удалении/поиске

Item\* forFind=NULL;//результат поиска. Пока просто обнулим его

Item\* lst=NULL;//сам список. Пока что он пуст

while(choice)

{

system("cls");

cout<<"Main Menu\n"

"1 - Add new element\n"

"2 - Find element\n"

"3 - Delete element\n"

"4 - Print list\n"

"5 - Exit\n";

cin>>choice;

cout<<endl;

if(cin.fail() || (choice<1 || choice>5))//проверка на ошибочный выбор пользователем пункта меню

{

cin.clear();//очистка состояния потока

choice=1;

continue;

}

else

switch (choice)

{

case 1:

cout<<"Enter key:\n";

cin>>key;

if(cin.fail())

{

cin.clear();

cout<<"Try again.\n";

\_flushall();//очистка потока от мусора что навводил пользователь

break;

}

cin.ignore();

cout<<"Enter info:\n";

cin.getline(str,256);

Add(lst,key,str);

break;

case 2:

cout<<"Enter key:\n";

cin>>key;

if(cin.fail())

{

cin.clear();

cout<<"Try again.\n";

\_flushall();

break;

}

cin.ignore();

forFind=Find(lst,key);

if(forFind)

cout<<forFind;

break;

case 3:

cout<<"Enter key:\n";

cin>>key;

if(cin.fail())

{

cin.clear();

cout<<"Try again.\n";

\_flushall();

break;

}

cin.ignore();

if(Delete(lst,key))

cout<<"Done\n";

break;

case 4:

cout<<lst;

break;

case 5:

break;

default:

cout<<"Wrong selection.\n";

}

if(choice==5)

break;

cout<<"Press any key to continue...\n";

getch();

}

DeleteList(lst);

return 0;

}

\*/

