Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования «Томский государственный университет систем
управления и радиоэлектроники»

Факультет систем управления (ФСУ)

Кафедра автоматизированных систем управления (АСУ)

Списки

отчет по лабораторной работе №3 по дисциплине
«Структуры и алгоритмы обработки данных в ЭВМ»

Обучающийся гр. 431-3
_____ Сергиевский Д.В.
«_18_» _октября____ 2022 г.

Проверил: доцент каф. АСУ, д.т.н.
_____Горитов А. Н.
«_18_» _октября____ 2022 г.

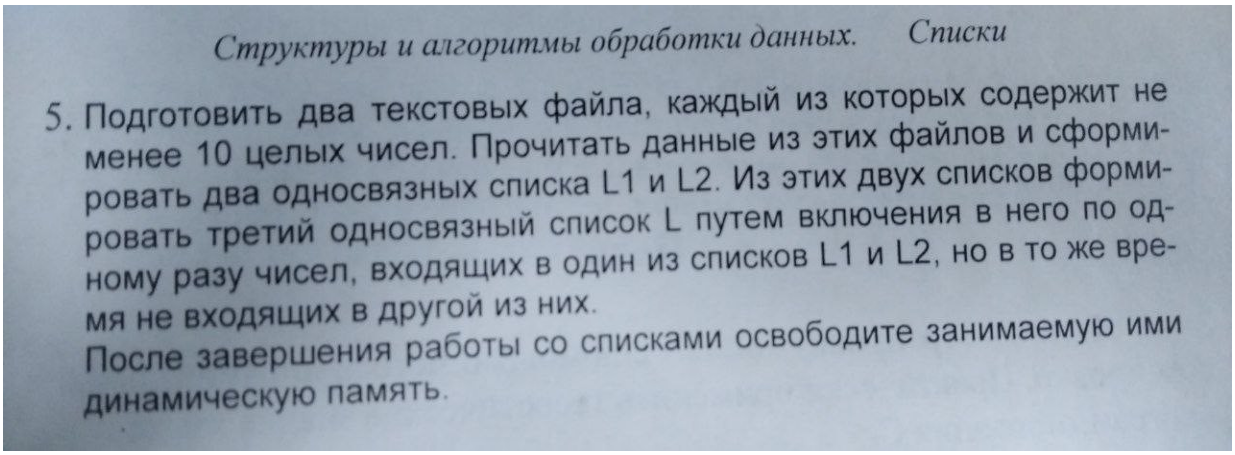
Томск 2022

Содержание

Введение

В рамках данной лабораторной работы необходимо решить небольшую задачу с применением АТД Список для закрепления теоретического материала.

Задание на лабораторную работу представлено на Рисунке 1.



Структуры и алгоритмы обработки данных. Списки

5. Подготовить два текстовых файла, каждый из которых содержит не менее 10 целых чисел. Прочитать данные из этих файлов и сформировать два односвязных списка L1 и L2. Из этих двух списков формировать третий односвязный список L путем включения в него по одному разу чисел, входящих в один из списков L1 и L2, но в то же время не входящих в другой из них.
После завершения работы со списками освободите занимаемую ими динамическую память.

Рисунок 1 - задание

1 Ход работы

1.1 Алгоритм решения

Решение данной задачи осуществляется в четыре этапа.

1. Инициализация списков L1, L2 значениями из файлов.
2. Обработка списка L1. Из списка последовательно считываются элементы и проверяется их вхождение в список L2. В случае вхождения значения во второй список, происходит удаление элементов списков, содержащих оное. Иначе значение заносится в результирующий список L.
3. Обработка списка L2. Все элементы списка считываются и заносятся в результирующий список L.
4. Вывод списка L.

В случае неверных входных данных программа завершается с выводом пояснительного сообщения.

1.2 Реализация решения

Для решения данной задачи потребовалось написать небольшую программу, реализующую вышеописанный алгоритм, а также АТД «односвязный список» на основе динамического распределения памяти в качестве вспомогательного компонента.

В качестве реализации односвязного списка был выбран односвязный список с фиктивным элементом в целях упрощения кода.

Листинг программы приведен в Приложении А.

1.3 Пример решения

Пример входных данных представлены на Рисунке 1.1.

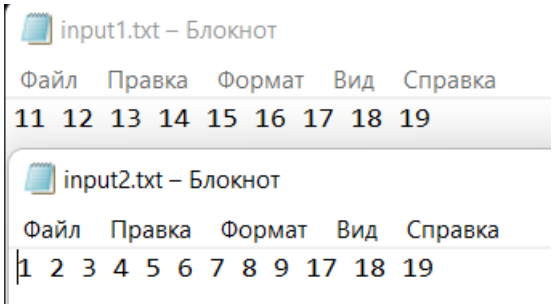


Рисунок 1.1 - пример входных данных

В результате выполнения программы в списке L окажутся элементы списков L1, L2, содержащиеся только в одном из них. То есть симметричная разность множеств их элементов.

Пример вывода программы представлен на Рисунке 1.2.

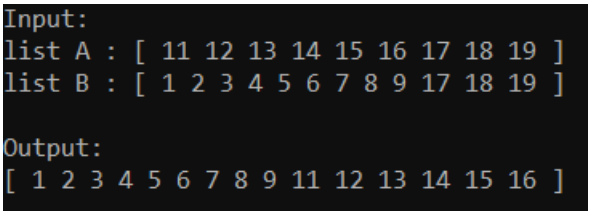


Рисунок 1.2 - пример вывода

Вывод

В результате данной лабораторной работы были подкреплены теоретические знания по теме «АТД Список» реализацией соответствующей структуры и её применением для решения небольшой задачи.

Приложение А

Файл s3_sad_lab3.cpp. Точка входа в программу и решение задачи.

```
// s3_sad_lab3.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.
//


#include <iostream>
#include <fstream>
#include <string>
using namespace std;


using ind = size_t;
using val_type = double;


static const ind nullind = 0;


class List
{
private:
    val_type value;
    List* next;


    List(const val_type value, List* next) : value(value), next(next) {}
```

```
    List* get_item(ind index)
    {
        List* item = this;
        for (ind i = 0; i < index && item; ++i)
            item = item->next;

        if (item == nullptr)
            throw "index is out of range";

        return item;
    }

public:
    List() : value(0), next(nullptr) {}

    ~List()
    {
        while (next != nullptr)
            remove(1);
    }

    void insert_after(ind index, const val_type value)
    {
        List* item = get_item(index);
        item->next = new List(value, item->next);
```

```
    if (next == nullptr)
        throw "new returned nullptr";
}

val_type remove(ind index)
{
    if (index == nullind)
        throw "index is out of range";

    List* item = get_item(index-1);

    if (item->next == nullptr)
        throw "index is out of range";

    val_type val = item->next->value;

    delete exchange(item->next, exchange(item->next->next, nullptr));

    return val;
}

val_type get(ind index)
{
    if (index == nullind)
```

```
            throw "index is out of range";


        return get_item(index)->value;
    }


    ind find(val_type value) const
    {
        ind i = 1;
        for (List* item = this->next; item != nullptr; item = item->next, ++i)
            if (item->value == value)
                return i;
        return nullind;
    }


    bool is_empty() const
    {
        return next == nullptr;
    }
};


void init_list(List& list, const string filename, bool with_cout_values = false, string name = "list")
{
    fstream input(filename, fstream::in);
    if (!input.is_open())
    {
```

```
        throw filename + " didn`t open";
    }

    cout << name << " : " << "[ ";

    val_type buffer;
    while(input >> buffer)
    {
        list.insert_after(0, buffer);
        cout << buffer << " ";
    }

    cout << "]" << endl;
}

void do_task(const string filenameA, const string filenameB)
{
    List listA, listB;

    cout << "Input: " << endl;

    init_list(listA, filenameA, true, "list A");
    init_list(listB, filenameB, true, "list B");

    cout << endl;
```

```
List res;
val_type val;
ind index = nullind;

while (!listA.is_empty())
{
   val = listA.remove(1);
   index = listB.find(val);

   if (!index)
   {
      res.insert_after(0, val);
      continue;
   }

   for (; index; index = listB.find(val))
      listB.remove(index);

   for (index = listA.find(val); index; index = listA.find(val))
      listA.remove(index);
}

while (!listB.is_empty())
```

```
    {
        res.insert_after(0, listB.remove(1));
    }

    cout << "Output: " << endl;
    cout << "[ ";

    while(!res.is_empty())
        cout << res.remove(1) << " ";

    cout << "]" << endl;
}

int main()
{
    try
    {
        do_task("input1.txt", "input2.txt");
    }
    catch (const char message)
    {
        cout << "error: " << message;
        getchar();
    }
```

```
}


// Запуск программы: CTRL+F5 или меню "Отладка" > "Запуск без отладки"

// Отладка программы: F5 или меню "Отладка" > "Запустить отладку"


// Советы по началу работы

//   1. В окне обозревателя решений можно добавлять файлы и управлять ими.

//   2. В окне Team Explorer можно подключиться к системе управления версиями.

//   3. В окне "Выходные данные" можно просматривать выходные данные сборки и другие сообщения.

//   4. В окне "Список ошибок" можно просматривать ошибки.

//     5. Последовательно выберите пункты меню "Проект" > "Добавить новый элемент", чтобы создать файлы кода, или "Проект" > "Добавить существующий элемент", чтобы добавить в проект существующие файлы кода.

//   6. Чтобы снова открыть этот проект позже, выберите пункты меню "Файл" > "Открыть" > "Проект" и выберите SLN-файл.
```