МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева»

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Алгоритмы и структуры данных

Хеширование

Руководитель _____ В. В. Тынченко
подпись, дата инициалы, фамилия

Обучающийся БПИ22-02, 221219040 _____ К.В. Трифонов
номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2023 г.

Цель работы:

Изучить метод хеширования на примере организации таблицы идентификаторов на стадии лексического анализа исходного текстапрограммы.

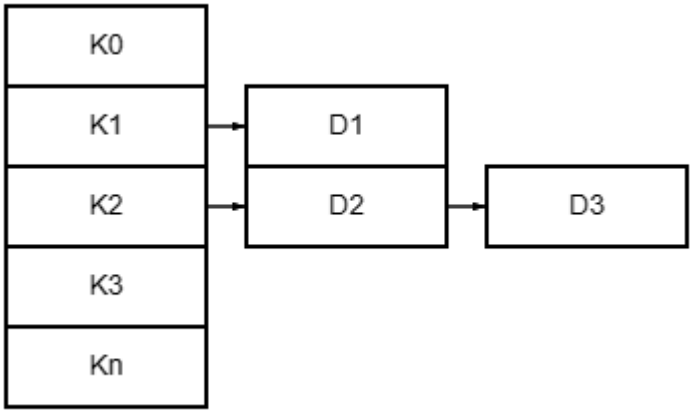
Постановка задачи.

Для выполнения лабораторной работы требуется написать программу, которая получает на входе набор идентификаторов, организует таблицу по заданному методу и позволяет осуществить многократный поиск идентификатора в этой таблице. Список идентификаторов считать заданным в виде текстового файла. Длина идентификаторов ограничена 32 символами. Варианты заданий представлены в конце раздела.

Вариант №21.

21.	Мультипликативный метод	Метод цепочек

Схема организации хеш-таблицы:



Решение коллизий методом цепочек подразумевает использование коллекций для хранения элементов, которым соответствует одно значение хеш-функции. Каждая ячейка такой хеш-таблицы является коллекцией элементов, которые соответствуют одному ключу.

Для поиска элемента в хеш-таблице с решением коллизий методом цепочек нужно получить значение хеш-функции по данному элементу, затем обратиться к коллекции по значению хеш-функции и проверить все её элементы, пока не будет найден искомый, если искомый не найден в коллекции, то делается вывод, что данный элемент в этой хеш-таблице не находится.

Код программы:

Описание класса хеш-таблицы *hashtab*:

```
class hashtab {
private:
    size_t table_size;
    vector<vector<string>> table;

//случайное число в диапазоне от 0 до 1 для работы мультипликативной хеш-функции
    const double hash = 0.618033;
```

Класс включает в себя 3 поля: размер таблицы (задаётся конструктором с параметрами), вектор, состоящий из векторов, которые включают в себя элементы, которые соответствуют одному ключу и константу золотого сечения.

```
public:
    //Конструкторы
    hashtab() : table(table_size) {}
    hashtab(size_t size) : table_size(size), table(table_size) {}
```

```
    //Хеш-функция
    int HashFunction(string key){
        double f = hash;
        f = f / 10;
        int hash = 0;
        // умножить ключ на случайное число из диапазона 0...1
        for (char ch : key){
            hash += ch;
        }
        f = hash * f;
        // взять дробную часть
        f = f - int(f);
        // возвратить число в диапазоне 0...n-1
        return table_size*f;
    }
```

Мультипликативный метод хеширования основан на простой математической операции умножения. В этом методе используется константа f, которая является дробью вида $(\sqrt{5}-1)/2$, которая приближенно равна 0.6180339887. Это число является обратным числом золотого сечения φ, это значение было выбрано, чтобы обеспечить лучшую равномерность распределения хеш-значений.

Использование случайного числа f для хеш-функции делает невозможным быстрое обращение к элементам по ключу, т.к. такая функция бы выдавала разные ключи для одного и того же элемента в разные моменты времени.

```
    //Вставка ключа / возвращает число коллизий
    int insert(const std::string& key){
        int index = HashFunction(key);
        table[index].push_back(key);
        return table[index].size();
    }
    //Поиск элемента / возвращает число сравнений
    int search(const std::string& key){
        int index = HashFunction(key);
        int count = 1;
        int comparisons = 0;
        vector<string>& chain = table[index];
        for (string value : chain) {
            comparisons++;
            if (value == key) {
                return comparisons; // Элемент найден
            }
        }
        return 0; // Элемент не найден
    }
```

Метод поиска в хеш-таблице с использованием метода цепочек включает вычисление хеша для ключа, определение соответствующей ячейки в таблице, и поиск элемента в цепочке, связанной с этой ячейкой. Если в ячейке происходит коллизия, элементы с одинаковым хешем хранятся в цепочке, и поиск осуществляется внутри этой цепочки. Результатом поиска является найденное значение или вывод о том, что элемент отсутствует в хеш-таблице.

```
    //Удаление ключа / возвращает число сравнений
    int remove(const std::string& key){
        int index = HashFunction(key);
        int count = 1;
        int comparisons = 0;
        for (auto i = table[index].begin(); i!=table[index].end(); i++){
            comparisons++;
            if (*i == key){
                table[index].erase(i);
                cout<<"Ключ удалён.\n";
                return comparisons;
            }
        }
        cout<<"Ключ не найден.\n";
        return comparisons;
    }
```

Удаление ключа из хеш-таблицы с методом цепочек включает в себя вычисление хеша для ключа, нахождение соответствующей ячейки и удаление элемента из цепочки, связанной с этой ячейкой. После удаления элемента, хеш-таблица остается валидной, и последующий поиск данного ключа возвращает " Ключ не найден.".

```
    //Вывод в поток
    ostream& print(std::ostream& out) const{
        size_t c = 0;
        size_t size = table_size;
        for (int i = 0; i < table_size; ++i){
            out << i << " : ";
            if (table[i].size() == 0){size--;}
            else{
                for(int j = 0; j < table[i].size(); j++){
                    c++;
                    out << table[i][j];
                    if (j + 1 != table[i].size()){
                        out<<" -> ";
                    }
                }
            }
            out<<"\n";
        }
        if(size!=0){
            out<<"Среднее число сравнений: "<<static_cast<double>(c) / size<<
"\n";
            }
        return out;
    }
```

Подсчет среднего числа сравнений производится путём деления количества всех элементов в таблице на количество занятых в ней ключей.

```
    //Очистка таблицы
    void clear(){
        for (int i = 0; i < table_size; ++i){
            table[i].clear();
        }
    }
};
```

Очистка таблицы методом stl clear для vector.

```
ostream& operator << (ostream & out,const hashtab& p){
    p.print(out);
    return out;
}
```

Содержимое файла *main.cpp*:

```
while (f){
        cout << "Главное меню\n"
            << "1) Вставить элемент\n"
            << "2) Вставить из файла\n"
            << "3) Удалить ключ\n"
            << "4) Найти ключ\n"
            << "5) Вывод в консоль\n"
            << "6) Вывод в файл\n"
            << "0) <<< Выход\n";
```

Меню тестирования программы

```
case 1:{
    string key;
    cout << "Введите элемент: ";
    cin >> key;
    if(p.search(key)){
        cout << "Этот элемент уже находится в таблице.\n";
    }
    size_t collisions = p.insert(key);
    cout<<"Всего коллизий: "<< collisions<<"\n";
    break;
}
```

Ключ вводится пользователем, если элемент уже присутствует в таблице, то пользователю выведется соответствующее сообщение. Если элемент отсутствует, то произойдёт вставка элемента и последующим выводом на экран числа коллизий (длина вектора элементов, соответствующих одному ключу).

```
case 2:{
    p.clear();
    string key_insert_file;
    ifstream fin("input.txt");
    if (!fin){
        cout << "Ошибка открытия файла.\n";
    }
    else{
        while (getline(fin, key_insert_file)){
            if (!p.search(key_insert_file)){
                p.insert(key_insert_file);
            }
        }
        fin.close();
        cout << "Чтение завершено.\n";
    }
    break;
}
```

Открытие файла с последующим чтением с проверкой на существование элементов в таблице.

```
case 3:{
    string key_delete;
    cout << "Введите ключ: ";
    cin >> key_delete;
    p.remove(key_delete);
    break;
}
```
Удаление элемента

```
case 4:{
    string key;
    cout << "Введите ключ: ";
    cin >> key;
    size_t comparisons = p.search(key);
    if (comparisons){
        cout << "Ключ найден.\n"
        <<"Всего сравнений: "<<comparisons<<"\n";
        }
    else{cout << "Ключ НЕ найден.\n";}
    break;
}
```
Пользователю предоставляется возможность ввести ключ, после чего производится поиск этого ключа в хеш-таблице. Если ключ найден, программа сообщает об успехе и выводит количество сравнений, выполненных в процессе поиска. В противном случае выводится сообщение о том, что ключ не найден.

```
            case 5:{
                cout << p;
                break;
            }
```
Вывод в консоль через перегрузку оператора <<.

```
case 6:{
                ofstream fout("output.txt");
                if (!fout){
                    cout << "Ошибка открытия файла.\n";
                }
                else{
                    p.print(fout);
                    fout.close();
                    cout << "Запись завершена.\n";
                }
                break;
            }
```
Вывод в файл output.txt.

Тестирование

Заполнение хеш-таблицы элементами World, apple, paple:

```
Введите ключ: World
Всего коллизий: 1
Главное меню
1) Вставить элемент
2) Вставить из файла
3) Удалить ключ
4) Найти ключ
5) Вывод в консоль
6) Вывод в файл
0) <<< Выход
1
Введите ключ: apple
Всего коллизий: 1
Главное меню
1) Вставить элемент
2) Вставить из файла
3) Удалить ключ
4) Найти ключ
5) Вывод в консоль
6) Вывод в файл
0) <<< Выход
1
Введите ключ: paple
Всего коллизий: 2
```

Вывод в консоль хеш-таблицы:

```
5
0 :
1 :
2 : World
3 :
4 :
5 :
6 :
7 :
8 :
9 :
10 :
11 :
12 :
13 :
14 :
15 : apple -> paple
16 :
17 :
18 :
19 :
Среднее число сравнений: 1.5
```

Элементы apple и paple были расположены друг за другом, т.к. им соответствует одно значение хеш-функции (их char-сумма равна).

Чтение файла состоящего из 100 хаотичных слов (неосмысленных), часть из них является копией другого слова из этого списка с перестановленными буквами, чтобы вызвать множественные коллизии.

Часть файла input.txt:

Результат работы программы:

```
2
Чтение завершено.
Главное меню
1) Вставить элемент
2) Вставить из файла
3) Удалить ключ
4) Найти ключ
5) Вывод в консоль
6) Вывод в файл
0) <<< Выход
5
0 : Ldore -> Eldor -> Elrod -> Ledro -> Rodel -> Droel -> Rleod -> Ldreo -> Lrdoe -> Lored -> Odler -> Loder
-> Rdleo -> Orlde -> Olred -> Dreol
1 :
2 : Wrold
3 : Rwelo -> Wroel -> Wloer -> Lwore -> Rlewo -> Rwleo -> Wleor -> Rleow -> Orlwe -> Wlroe -> Lreow -> Rwoel
4 :
5 :
6 : Ldowe -> Dowel -> Wodle -> Wloed -> Wdloe -> Edwol -> Dleow -> Dwelo -> Owdel
7 :
8 : LrweoD
9 : Word -> Wrod -> Rodw -> RwoD -> Drow -> Wdor -> OrwD
10 : Lwder -> Rldwe -> Lwred -> Dlwre
11 :
12 :
13 :
14 : OrweD -> Doerw -> RwoeD -> Edwro -> Rewod -> Dorwe -> Erwod -> Dewor -> Odrwe -> Drweo -> Rewdo -> Wdreo
 -> Wrdoe -> Rweod -> Wrode
15 :
16 :
17 :
18 :
19 :
Среднее число сравнений: 10.875
```

Пример удаления элемента “Wrode” из хеш-таблицы:

```
5
0 : Ldore -> Eldor -> Elrod -> Ledro -> Rodel -> Droel -> Rleod -> Ldreo -> Lrdo
-> Rdleo -> Orlde -> Olred -> Dreol
1 :
2 : Wrold
3 : Rwelo -> Wroel -> Wloer -> Lwore -> Rlewo -> Rwleo -> Wleor -> Rleow -> Orlw
4 :
5 :
6 : Ldowe -> Dowel -> Wodle -> Wloed -> Wdloe -> Edwol -> Dleow -> Dwelo -> Owde
7 :
8 : LrweoD
9 : Word -> Wrod -> Rodw -> RwoD -> Drow -> Wdor -> OrwD
10 : Lwder -> Rldwe -> Lwred -> Dlwre
11 :
12 :
13 :
14 : OrweD -> Doerw -> RwoeD -> Edwro -> Rewod -> Dorwe -> Erwod -> Dewor -> Odr
 -> Wrdoe -> Rweod -> Wrode
15 :
16 :
17 :
18 :
19 :
```

```
5
0 : Ldore -> Eldor -> Elrod -> Ledro -> Rodel -> Droel -> Rleod -> Ldr
-> Rdleo -> Orlde -> Olred -> Dreol
1 :
2 : Wrold
3 : Rwelo -> Wroel -> Wloer -> Lwore -> Rlewo -> Rwleo -> Wleor -> Rle
4 :
5 :
6 : Ldowe -> Dowel -> Wodle -> Wloed -> Wdloe -> Edwol -> Dleow -> Dwe
7 :
8 : LrweoD
9 : Word -> Wrod -> Rodw -> RwoD -> Drow -> Wdor -> OrwD
10 : Lwder -> Rldwe -> Lwred -> Dlwre
11 :
12 :
13 :
14 : OrweD -> Doerw -> RwoeD -> Edwro -> Rewod -> Dorwe -> Erwod -> De
 -> Wrdoe -> Rweod
15 :
16 :
17 :
18 :
19 :
```

Вывод хеш-таблицы в файл output.txt:

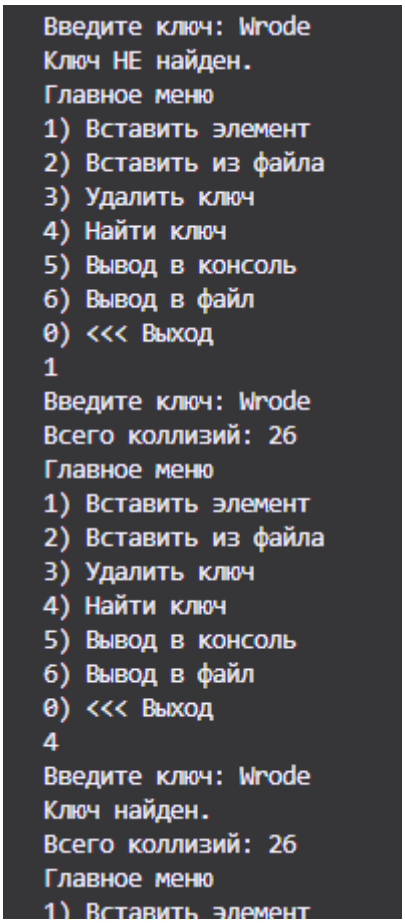
```
Главное меню
1) Вставить элемент
2) Вставить из файла
3) Удалить ключ
4) Найти ключ
5) Вывод в консоль
6) Вывод в файл
0) <<< Выход
6
Запись завершена.
```

Содержимое файла output.txt:

lab5 > output.txt

```
0 : Ldore -> Eldor -> Elrod -> Ledro -> Rodel -> Droel -> Rleod -> Ldreo -> Lrdoe -> Lored ->
1 :
2 : Wrold
3 : Rwelo -> Wroel -> Wloer -> Lwore -> Rlewo -> Rwleo -> Wleor -> Rleow -> Orlwe -> Wlroe ->
4 :
5 :
6 : Ldowe -> Dowel -> Wodle -> Wloed -> Wdloe -> Edwol -> Dleow -> Dwelo -> Owdel
7 :
8 : LrweoD
9 : Word -> Wrod -> Rodw -> RwoD -> Drow -> Wdor -> OrwD
10 : Lwder -> Rldwe -> Lwred -> Dlwre
11 :
12 :
13 :
14 : OrweD -> Doerw -> RwoeD -> Edwro -> Rewod -> Dorwe -> Erwod -> Dewor -> Odrwe -> Drweo ->
15 :
16 :
17 :
18 :
19 :
Среднее число сравнений: 10.875
```

Поиск элемента “Wrode” в хеш-таблице:



Ответы на контрольные вопросы

1. **Что такое таблица символов, и для чего она предназначена?**

Таблица символов — это структура данных, предназначенная для хранения информации об идентификаторах программы, таких как переменные и константы, и связанных с ними атрибутов.

2. **Какая информация может храниться в таблице символов?**

В таблице символов может храниться информация об идентификаторах, включая их типы данных, области видимости, адреса в памяти и другие характеристики.

3. **Какие цели преследуются при организации таблицы символов?**

Основные цели при организации таблицы символов - обеспечить удобное хранение и эффективный поиск информации об идентификаторах, а также поддерживать операции добавления, удаления и модификации элементов.

4. **Какими характеристиками могут обладать константы, переменные?**

Константы и переменные в таблице символов могут обладать характеристиками, такими как тип данных, область видимости (локальная или глобальная), адрес в памяти, значение и другие свойства.

5. **Какие существуют способы организации таблиц символов?**

Существуют различные способы, включая списки, деревья и хеш-таблицы, для организации таблиц символов в зависимости от требований и характеристик языка программирования.

6. **Отсутствует**

7. **Расскажите о древовидной организации таблиц идентификаторов. В чем ее преимущества и недостатки?**

Древовидная организация использует структуру дерева для хранения идентификаторов. Преимущества включают эффективность поиска и удобство поддержки областей видимости. Недостатки - возможно неэффективное использование памяти и более сложная реализация.

8. **В чем суть хеш-адресации?**

Хеш-адресация — это метод, при котором используется хеш-функция для вычисления адреса памяти, в которой должен быть хранится идентификатор.

9. **Что такое хеш-функции и для чего они используются?**

Хеш-функции преобразуют входные данные (например, ключ) в фиксированный размер хеш-значения. Они используются для быстрого и эффективного поиска в хеш-таблицах.

10. **Расскажите о преимуществах и недостатках организации таблицы идентификаторов с помощью хеш-функции.**

Преимущества включают быстрый поиск по хеш-значению. Недостатки включают возможные коллизии и сложности выбора эффективной хеш-функции.

11. **Что такое коллизия? Почему она происходит? Можно ли полностью избежать коллизий?**

Коллизия — это ситуация, когда два различных ключа имеют одинаковое хеш-значение. Она происходит из-за ограниченности диапазона хеш-значений, и полностью избежать коллизий невозможно.

12. **Что такое рехеширование? Какие методы рехеширования существуют?**

Рехеширование — это процесс изменения хеш-функции или пересчета хеш-значений для устранения коллизий. Методы включают линейное, квадратичное и двойное рехеширование.

13. **В чем заключается метод цепочек?**

Метод цепочек использует структуры данных (например, списки) для хранения элементов с одинаковым хешем в одной ячейке хеш-таблицы.

14. **Как могут быть скомбинированы различные методы организации хеш-таблиц?**

Различные методы могут быть скомбинированы, например, использование метода цепочек с рехешированием для обработки коллизий и обеспечения эффективности.

Вывод

Программа успешно реализовывает метод цепочек для разрешения коллизий, где элементы с одинаковым хешем хранятся в структурах данных. Мультипликативный метод хеширования был использован для эффективного распределения ключей в таблице. В результате лабораторной работы получена программа, способная эффективно обрабатывать поиск идентификаторов в хеш-таблице с использованием метода цепочек. Лабораторная работа позволяет лучше понять принципы организации таблиц символов с использованием хеш-функций и метода цепочек при лексическом анализе программ.