Программа 5. Разработка контейнерного класса

Условие задачи

1. Подсчитать, сколько раз в произвольном вводимом из входного потока тексте встречается каждое слово (слова в тексте разделяются пробельными символами).

Примечание: для реализации контейнера использовать STL.

2. Задача полностью повторяет предыдущую (задачу 4): есть ассоциативный массив *Mas*, индекс элемента которого – текстовая строка.

Тогда алгоритм решения задачи:

для каждого слова из текста

++Mas[слово];

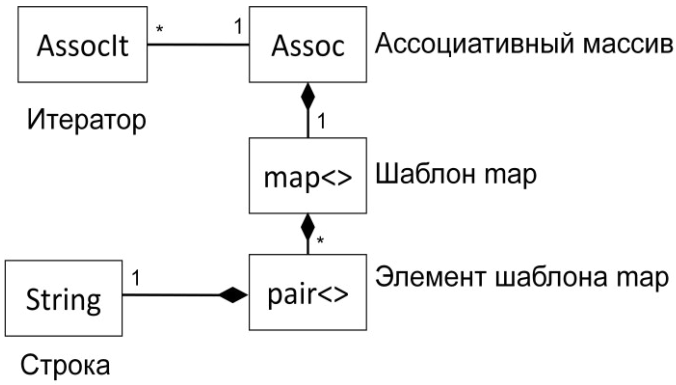
3. Проектирование класса Ассоциативный массив (*Assoc*) рекомендуется начать также с определения состояния этого класса и разработки диаграммы классов. В данном случае класс *Assoc* должен быть реализован с помощью библиотеки STL.

4. Соответственно, для класса *Assoc* определяется и реализуется необходимый интерфейс – состояние и методы класса (состояние изменится, а методы, в основном, повторяют соответствующие методы предыдущего варианта задачи).

5. Для контейнерного класса обязательно должен быть разработан итератор.

Особенности реализации

Так как в условии задачи не определяется конкретный контейнерный класс из STL, следует выбрать наиболее подходящий. Поскольку доступ к элементам контейнера осуществляется по ключу, а сам элемент представляет собой пару значений (ключ – значение), целесообразно использовать контейнер *map*<>; к тому же, данный контейнер не допускает размещение двух элементов с одинаковым значением ключа, что соответствует условиям данной задачи. Диаграмма классов может быть представлена следующим образом:



Если для доступа к элементам класса по индексу используется перегрузка оператора индексирования, следует реализовать два перегруженных оператора – для l-value и для r-value.

Если с помощью итератора предполагается выполнять для экземпляра класса только операции чтения, необходимо реализовать константный итератор. Если же с помощью итератора предполагается и читать состояние экземпляра класса, и напрямую изменять его – тогда должны быть реализованы два итератора: константный и изменяемый.

Текст программы для данного варианта представлен с помощью следующих файлов:

- файл *Assoc_STL.h* – для определения класса,
- файл *Assoc_STL.cpp* – для реализации методов класса,
- файл *Assoc_STL_appl.cpp* – пример отладочной программы, в которой используется и итератор контейнерного класса; следует отметить, что данный файл практически не изменился по сравнению с аналогичным файлом в предыдущем примере программы (*Prog4*) – изменились только имя файла-заголовка и имя пространства имен.

Исходный текст

```
// файл Assoc_STL.h
#ifndef _ASSOC_STL_H_
#define _ASSOC_STL_H_
#include <iostream>
#include <map>
#include <string>
```

```
namespace Prog5{
    // Перегруженный оператор вывода для структуры std::pair
    // Так как в структуре все поля - в public области (по умолчанию),
    // объявлять оператор другом структуры нет необходимости
    std::ostream & operator <<(std::ostream &, const std::pair<std::string, int> &);

    // Класс Ассоциативный массив
    class Assoc
    {
    private:
        std::map<const std::string, int> arr;    // массив элементов; контейнер из библиотеки STL
    public:
        // Необходимые конструкторы
        Assoc(){}
        Assoc(const Assoc &);
        // Деструктор
        // Можно ничего не делать, так как будет вызван деструктор для std::map<>
        ~Assoc(){};
        // Оператор присваивания
        Assoc & operator =(const Assoc &);
        // Операторы индексирования
        int & operator [](const std::string &);      // для l-value
        int operator [](const std::string &) const;  // для r-value
        // Вывод в поток
        friend std::ostream & operator <<(std::ostream &, const Assoc &);
        // Объявления для итератора
        friend class ConstAssocIt;
        typedef ConstAssocIt ConstIterator;
        // Методы итератора
        ConstIterator begin();
        ConstIterator end();
    private:
        // поиск элемента по индексу - строке
        ConstIterator find(const std::string &);
    };

    // Класс-итератор для ассоциативного массива
    class ConstAssocIt{
    private:
        std::map<const std::string, int>::iterator cur;  // текущий элемент массива
    public:
        // Необходимые конструкторы
        ConstAssocIt(){}
        ConstAssocIt(std::map<const std::string, int>::iterator it) :cur(it){}
        // Операторы сравнения
        int operator !=(const ConstAssocIt &) const;
        int operator ==(const ConstAssocIt &) const;
        // Доступ к элементу массива по указателю
        // оператор * возвращает значение самой структуры
        std::pair<const std::string, int> & operator *();
        // оператор -> должен возвращать значение указателя на структуру
        std::pair<const std::string, int> * operator ->();
        // Перемещение итератора на следующую позицию в массиве
        ConstAssocIt & operator ++();
        ConstAssocIt operator ++(int);
    };
}
#endif
```

```
// файл Assoc_STL.cpp
#include "Assoc_STL.h"

namespace Prog5 {
    // Перегруженный оператор вывода в поток для структуры
    std::ostream & operator <<(std::ostream &os, const std::pair<std::string, int> &p)
    {
        return os << p.first << " - " << p.second;
    }
```

```
// Копирующий конструктор;
Assoc::Assoc(const Assoc &a)
{
    std::map<const std::string, int>::const_iterator p;
    for (p = a.arr.begin(); p != a.arr.end(); ++p)
        arr.insert(make_pair(p->first, p->second));
    // метод insert() требует только один операнд; поэтому используется специальная функция
    // make_pair() из библиотеки STL, которая из двух заданных аргументами значений
    // создает один экземпляр типа std::pair<>, соответствующего типу элемента контейнера
    // std::map<>
}

// Перегруженный оператор присваивания
Assoc & Assoc::operator =(const Assoc &a)
{
    std::map<const std::string, int>::const_iterator p;
    if (this != &a){ // проверка ситуации a = a
        arr.clear(); // освобождение занятой map памяти
        for (p = a.arr.begin(); p != a.arr.end(); ++p)   // копирование данных
            arr.insert(make_pair(p->first, p->second));      // см. комментарий выше
    }
    return *this;
}

// Поиск элемента массива по значению индекса (строке)
ConstAssocIt Assoc::find(const std::string &s)
{
    std::map<const std::string, int>::iterator p = arr.find(s);
    return ConstAssocIt(p);
}

// Перегруженный оператор [] для операнда l-value
// (выражение ob[] может появиться слева от присваивания)
int & Assoc::operator[] (const std::string &s)
{
    // есть ли в массиве элемент с заданным значением индекса
    std::map<const std::string, int>::iterator p = arr.find(s);
    if (p == arr.end()){
        // такого элемента нет; нужно добавить новый элемент
        std::pair<std::map<const std::string, int>::iterator, bool> pp =
            arr.insert(make_pair(s, 0));
        // метод insert() возвращает значение типа std::pair<iterator, bool>; первый элемент
        // пары – значение итератора, соответствующее вновь вставленному элементу;
        // второй элемент – информация о том, успешно (true) или не успешно (false) выполнилась
        // операция вставки

        if (!pp.second)
            throw std::exception("can't insert new item into map");
        p = pp.first;
    }
    return p->second;    // возвращается ссылка на элемент массива
                         // с заданным значением индекса
}

// Перегруженный оператор [] для операнда r-value
// (выражение ob[] может появиться только в выражениях (справа от присваивания))
int Assoc::operator[] (const std::string &s) const
{
    std::map<const std::string, int>::const_iterator p = arr.find(s);
    if (p == arr.end()) // есть ли в массиве элемент с заданным значением индекса
        // если элемента нет - обращаться к элементу массива нельзя
        throw std::exception("illegal index");
    return p->second;
}

// перегруженный оператор вывода в поток
std::ostream & operator<<(std::ostream &s, const Assoc &a)
{
```

```
        s << "===== Assoc =====" << std::endl;
        std::map<const std::string, int>::const_iterator pp;
        for (pp = a.arr.begin(); pp != a.arr.end(); ++pp)
            s << *pp << std::endl;
        return s;
    }

    // Начальная инициализация для итератора
    Assoc::ConstIterator Assoc::begin()
    {
        return ConstAssocIt(arr.begin());
    }

    // Определение последнего значения для итератора
    Assoc::ConstIterator Assoc::end()
    {
        return ConstAssocIt(arr.end());
    }

    // Сравнение значений двух итераторов
    int Assoc::ConstIterator::operator !=(const ConstAssocIt &it) const
    {
        return cur != it.cur;
    }

    int ConstAssocIt::operator ==(const ConstAssocIt &it) const
    {
        return cur == it.cur;
    }

    // Разыменование итератора (доступ по указателю)
    // оператор * возвращает значение самой структуры – т.е. именно то, что возвращает
    // перегруженный оператор * для итератора, определенного для STL-контейнера map<>
    std::pair<const std::string, int> & ConstAssocIt::operator *()
    {
        return *cur;
    }

    // Разыменование итератора (доступ по указателю)
    // оператор -> должен возвращать значение указателя на структуру;
    // поэтому сначала с помощью перегруженного оператора * для итератора для map<>
    // получаем саму структуру, а затем вычисляем ее адрес
    std::pair<const std::string, int> * ConstAssocIt::operator ->()
    {
        return &(*cur);
    }

// Продвижение итератора к следующему элементу массива:
    // префиксный оператор ++
    ConstAssocIt & ConstAssocIt::operator ++()
    {
        ++cur;
        return *this;
    }

    // постфиксный оператор ++
    ConstAssocIt ConstAssocIt::operator ++(int)
    {
        ConstAssocIt res(*this);
        ++cur;
        return res;
    }
}
```

```
// файл Assoc_STL_appl.cpp
#include <iostream>
#include "Assoc_STL.h"

using namespace Prog5;
```

```
int main()
{
    std::string s;
    Assoc a;
    std::cout << "Enter words; press CTRL+z to quit:" << std::endl;
    while (std::cin >> s)
        ++a[s]; // перегруженный [] как l-value
    std::cout << a << std::endl;

    // пример использования итератора:
    // вывод с помощью итератора; используется перегруженный оператор *
    std::cout << "Output by use of class iterator" << std::endl;
    Assoc::ConstIterator it;
    for (it = a.begin(); it != a.end(); ++it)
        std::cout << (*it).first << " - " << (*it).second << std::endl;

    // подсчитать среднюю частоту использования слов с помощью итератора;
    // используется перегруженный оператор ->
    double avg = 0;
    int n = 0;
    for (it = a.begin(); it != a.end(); ++it){
        ++n;
        avg += it->second;
    }
    avg /= n;
    std::cout << "avg frequency = " << avg << std::endl;

    // использование перегруженного [] как r-value
    const Assoc ca = a;
    std::cin.clear();
    std::cout << "Enter the word you want to find; press ctrl+z to quit" << std::endl;
    while (std::cin >> s, std::cin.good()){
        try{
            // перегруженный [] как r-value
            std::cout << "The word \"" << s << "\" was met " << ca[s] << " times" <<  std::endl;
        }
        catch (const std::exception &msg)
        {
            std::cout << msg.what() << std::endl;
        }
    }
    return 0;
}
```

Пример тестирования программы (отличается от предыдущей программы наличием теста для проверки работы перегруженного оператора присваивания)

```
Enter words; press CTRL+z to quit:
qwe rty qwe rty fgh rty  fghd   fgh  hgf  qwe  fgh   rty  ss
^Z
===== Assoc =====
fgh - 3
fghd - 1
hgf - 1
qwe - 3
rty - 4
ss - 1

Output by use of class iterator
fgh - 3
fghd - 1
hgf - 1
qwe - 3
rty - 4
ss - 1
avg frequency = 2.16667
Enter the word you want to find; press ctrl+z to quit
qwe
```

```
The word "qwe" was met 3 times
rtyu
illegal index
^Z
after operator =:
===== Assoc =====
fgh - 3
fghd - 1
hgf - 1
qwe - 3
rty - 4
ss - 1

Для продолжения нажмите любую клавишу . . .
```