Шаблонный класс map

Обсудим теперь шаблонный класс, имеющийся в стандартной библиотеке для организации словаря, или, как его еще называют, ассоциативного массива: это класс map (соответствующий заголовочный файл так и называется map).

Смысл задачи, решаемой этим классом, состоит в том, чтобы иметь "почти" массив. Обычный массив хранит набор элементов, доступных по номеру. Ассоциативный массив (словарь) хранит также набор элементов, но доступны они уже по значению почти любого типа, например, строке, как в обычном, бумажном, словаре (откуда и происходит одно из названий). То значение, которое играет в ассоциативном массиве роль индекса, называется ключом (термин взят из теории баз данных; продолжая это заимствование, то значение, которое соответствует ключу в ассоциативном массиве, будем называть данными).

Тип ключа может быть почти любым. Единственное ограничение состоит в том, чтобы ключи можно было сравнивать на больше-меньше, причем это отношение порядка должно быть линейным, т. е. из двух разных элементов типа ключей всегда один должен быть меньше другого. При этом на удивление смысл этого отношения порядка совершенно не важен, лишь бы этот порядок был линейным и просто вычислялся, т. е. для класса ключей должны быть перегружены операции сравнения, и их вычисление должно работать быстро. Для строк, это, очевидно, так, но возможны и другие типы ключей.

Например, в качестве ключей могут выступать объекты шаблонного класса вектор. Для них в стандартной библиотеке определено линейное отношение порядка, как для слов в словаре, только вместо букв используются элементы произвольного типа – шаблонного параметра (если, конечно, их можно сравнивать).

Причем для векторов в математическом смысле такой порядок является достаточно искусственным, не имеющим почти никакого смысла, но он линеен. А вот разумный с математической точки зрения порядок (покоординатный, т. е. один вектор не превосходит другого, если у них одинаковая размерность и такое неравенство справедливо для всех координат), уже линейным не является (нельзя сравнить, например, векторы (1,2) и (2,1)) и, следовательно, не годится для того, чтобы из векторов можно было сделать ключи ассоциативного массива.

Иногда ассоциативные массивы используются и для целых ключей. Это имеет смысл, если максимальное значение ключа может быть много больше, чем число одновременно используемых ключей. В этом случае обычный массив непригоден, потому что он должен хранить элементы для любых индексов от 0 до максимального значения индекса, и если из этого набора используются одновременно только небольшая часть элементов, это приводит к нерациональному расходу памяти.

Говоря математическим языком, шаблонный класс map описывает отображения, более точно, хранимые отображения. Бывают еще и вычисляемые отображения, они в языке C++ представлены обычными функциями. Разница между теми и

другими состоит в том, каким образом находится значение такого отображения для конкретного аргумента. Если оно находится при помощи вычислений по какой-либо формуле, или в большей общности, при помощи алгоритма, то такое отображение – вычисляемое. Если же мы просто запоминаем значения отображения для каждого конкретного значения аргумента, то такое отображение – хранимое (принцип его хранения похож на таблицу, где для каждого значения параметра просто указано соответствующее ему значение).

Синтаксис объявления ассоциативного массива очень прост:

```
map<тип_ключа, тип_данных> имя_переменной;
```

Например, определить отображение m из целых чисел в целые можно, написав

```
map<int, int> m;
```

Такая декларация создает пустое отображение, т. е. такое, которое не определено нигде (без элементов).

Можно наделить это отображение элементами при определении (после принятия стандарта 11 года):

```
map<тип_ключа, тип_данных> имя_переменной = {{ключ1, данные1}, ..., {ключN, данныеN}};
```

Пример.

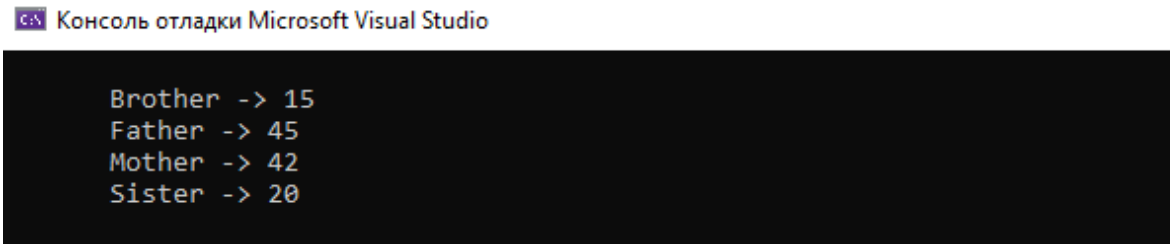
```
map <string, int> M = { {"Father", 45}, {"Mother", 42}, {"Sister", 20}, {"Brother", 15} };
```

Печать отображения
(можно использовать любой из трех способов)

Как и для последовательностей, можно перебрать все элементы отображения при помощи соответствующего итератора. Итератор отображения указывает на пары из ключа (поле first) и соответствующего ему элемента данных (поле second). Поэтому напечатать отображение M: string ->int можно следующим образом:

1-й способ:

```
map<string, int> M =
{{"Father", 45}, {"Mother", 42}, {"Sister", 20},{"Brother", 15}};
map <string, int>::iterator i;
for (i = M.begin(); i != M.end(); i++)
     cout << i->first << " -> " << i->second << endl;
```



Замечание.

Предыдущий цикл можно записать, используя ключевое слово auto:

```
for (auto i = M.begin(); i != M.end(); i++)
     cout << i->first << " -> " << i->second << endl;
```

Тип переменной `i` имеет длинную запись: `map<string, int>::iterator`. Написав тип `auto`, мы поручаем компилятору самостоятельно определить тип переменной `i`. Благодаря присваиванию `i = M.begin()`, компилятор поймет, что `M` имеет тип map, а метод `begin()` возвращает итератор, поэтому тип переменой `i` будет `map<string, int>::iterator`.

2-й способ:

После принятия стандарта C++ 11 года, то же самое можно сделать и при помощи более короткого цикла:

```
for (auto x : M)
     cout << x.first << " -> " << x.second << endl;
```

3-й способ:

Начиная со стандарта C++ 17 года, приведенную выше запись можно еще сократить:

```
for (auto [f,s] : M)
     cout <<  f  << " -> " <<  s  << endl;
```

где f и s – имена переменных (могут быть любыми идентификаторами).

Замечание.

Чтобы этот пример работал, нужно включить в свойствах проекта соответствие стандарту C++17. Для этого надо щелкнуть на имени проекта правой кнопкой мыши, выбрать пункт "свойства", затем C/C++ и, наконец, "язык".

На появившейся странице нужно выбрать строчку "стандарт языка C++" и в правой ее части выбрать стандарт 17 года.

Отметим, что шаблонный класс map устроен так, что перебор его элементов при помощи итераторов или коротких циклов всегда дает упорядоченный набор ключей.

Операция индексации, методы count, erase и clear.

Изменить отображение можно при помощи операции индексации. Например, можно указать, во что отображается конкретный элемент, написав что-то вроде `m[3]=7` или `M["Brother"] = 16.` При этом неважно, было ли отображение раньше определено на этом элементе, или нет.

Посмотреть, во что отображается некоторый элемент, можно также при помощи операции индексации, указав в квадратных скобках интересующий нас ключ.

Выяснить, определено ли отображение на некотором элементе, можно при помощи метода count (как и для множеств).

Убрать элемент из области определения отображения можно методом erase.

У класса map имеется также полезный метод clear, удаляющий все элементы отображения (как и у всех остальных контейнеров стандартной библиотеки).

Задачи.

Задачи 1–5 (общая сумма 10 баллов, каждое задание оценивается в два балла) или задачу про синонимы (из белого пояса по C++)

1. Написать функцию, вводящую с клавиатуры набор целых чисел (в любом порядке, конец ввода – Ctrl Z на пустой строчке) и выводящую числа из этого набора в порядке возрастания и без повторений, с указанием того, сколько каждое число раз встретилось в этом наборе.

2. Написать функцию, вводящую с клавиатуры отображение из целых чисел в целые числа (вводятся число пар и пары из двух целых чисел, первое из которых– ключ, а второе – данные), и возвращающую это отображение как результат типа map<int, int>.

3. Написать функцию, вводящую отображение (с помощью решения предыдущей задачи), и затем набор чисел, и выводящее для каждого введенного числа его образ при введенном отображении, если введенное отображение определено на этом числе, и само введенное число без изменений, если не определено.

4. Написать шаблонную функцию, вычисляющую сумму двух отображений (можно предполагать, что их области определения совпадают, и типы значений одинаковы).

5. Написать шаблонную функцию, вычисляющую композицию двух отображений.

Комментарий к задачам № 4 и № 5.

В задаче № 4 у нас два шаблонных параметра. Заголовок функции записывается так:

```
 template <class T1, class T2>
map <T1, T2> func4 (map <T1, T2> m1, map <T1, T2> m2)
```

Если шаблонных параметров три, то три шаблонных параметра, тогда

```
template <class T1, class T2, class T3>
map<T1, T3> f5(map<T1, T2> m1, map<T2, T3> m2)
```

```
template <class T1, class T2, class T3>
map<T1, T3> func5 (map<T1, T2> m1, map<T2, T3> m2)
{

}
```