УДК 681.3

О. Т. Марьянович

ГИБКАЯ СОРТИРОВКА ТАБЛИЦ С ИСПОЛЬЗОВАНИЕМ СПИСКОВ ТИПОВ

Описывается применение идиомы обобщенного программирования к работе с табличной информацией в памяти компьютера. Демонстрируется возможность использования шаблонных конструкций языка C++ для порождения таблиц произвольной структуры с возможностью их сортировки на основе заданных критериев.

Введение

В практике программирования нередко возникают задачи, подобные тем, которые решает язык SQL, однако в силу различных ограничений (одним из них является требование переносимости на разные платформы) реализацию таких задач приходится осуществлять в памяти компьютера без использования сторонних библиотек. Речь идет о задачах анализа информации, организованной в виде связанных таблиц. При этом источником информации может быть не реляционная база данных, а нечто иное, например данные могут поступать по сети на основе определенного протокола.

Характерными примерами рассматриваемых задач является поиск количества записей по определенному критерию, связывание записей из нескольких таблиц, агрегирование информации из различных таблиц и т.п. Для эффективного решения таких задач ключевым моментом является возможность сортировки таблиц на основе различных критериев.

Реализация подобных задач на основе традиционного C++ программирования сводится, по сути, к следующему:

а) для каждой таблицы вводится класс (TableRow), члены данных которого представляют поля строки таблицы;

б) каждая таблица представляется специальным классом (Table), содержащим член данных, хранящий коллекцию объектов класса TableRow;

в) предоставляется возможность сортировать записи таблицы – реализуется метод sort. Существует множество подходов к реализации алгоритмов сортировки, одним из который является создание массива указателей на строки таблицы (индекса) и сортировки данного массива с использованием стандартной функции qsort. При этом для каждого из необходимых критериев сортировки нужно реализовать функцию compare сравнения строк таблицы в соответствии с данным критерием;

г) в классе Table реализуются различные прикладные методы, обеспечивающие решение задач анализа данных.

Выполнение пунктов (а–в) не представляет сложности, однако, если таблиц много, возникает желание избавиться от необходимости ручного тиражирования однотипного кода, возложив данную задачу на компилятор C++.

Данная заметка посвящена описанию реализации такого подхода с использованием шаблонных конструкций языка C++ и технических приемов, изложенных в [1].

Использование шаблонных таблиц с сортировкой

Прежде чем перейти к деталям реализации, покажем, каким образом можно использовать шаблонные таблицы с сортировкой в программах на C++.

Основным классом является шаблонный класс Table, объявленный как template <class TList, class TIndexList = Loki::NullType> class Table.

Первый аргумент шаблона - список типов (класс Typelist библиотеки Loki, см. [1]), задающий структуру таблицы (каждый тип из списка соответствует типу данных колонки таблицы).

Второй аргумент - список типов, каждый элемент которого в свою очередь является списком типов. Такие вложенные списки типов содержат определения критериев сортировки, которые могут быть при-

менены к данной таблице. Критерий сортировки представляет собой последовательность пар колонка/порядок сортировки в форме шаблонного класса SortSpec, объявленного как template <int COL, int ORDER = 0> struct SortSpec {enum {column = COL }; enum {order = ORDER };}.

Данный класс содержит номер колонки, по которой следует выполнять сортировку, и порядок сортировки (возрастающий/убывающий).

Конструктор таблицы содержит аргумент, задающий ее размер (количество строк).

Доступ к ячейкам таблицы осуществляется при помощи методов template <unsigned int COL, typename VAL> inline void SetCell(unsigned int row_, const VAL val) и template <unsigned int COL, typename VAL> inline VAL GetCell(unsigned int row_) const.

Кроме того, имеются методы template <int INDEX> void sort(), inline void unsort(), inline void dump() и template < class TList > void dump(), позволяющие выполнять сортировку по заданному критерию, отменять сортировку (возвращаться к естественному порядку строк таблицы) и выводить таблицу на стандартное устройство построчно в соответствии с ее текущей сортировкой. При этом есть возможность избирательного вывода колонок посредством шаблонного варианта функции dump, где в качестве параметра передается список колонок, которые следует выводить.

В качестве типов колонок таблицы могут служить элементарные C++ типы, а также классы, удовлетворяющие несложным требованиям, среди которых реализация операторов присваивания и сравнения, а также функции dump для вывода объекта на стандартное устройство. В частности, в таком качестве для хранения текстовых строк используется специально разработанный класс template <unsigned int INIT_BUF_SIZE = 16 > class StringField.

Поясним сказанное следующим примером. Пусть необходимо работать со списком людей, содержащим следующую информацию: фамилия, год рождения, место рождения, заработная плата. При этом требуется выводить данный список, отсортированный по любой из колонок в возрастающем порядке, и фамилии всегда должны следовать в алфавитном порядке. Кроме того, нужно иметь возможность выводить таблицу, отсортированную по последней колонке в убывающем порядке.

Для этого достаточно объявить таблицу следующим образом:

```
enum { name, year, place, wage };
typedef SortSpec<name, asc>  ByName;
typedef SortSpec<year, asc>  ByYear;
typedef SortSpec<place, asc> ByPlace;
typedef SortSpec<wage, asc> ByWage;
typedef SortSpec<wage, desc> By WageDesc;
typedef Table<
    TYPELIST_4(StringField<>,
       int, StringField<>, int),
    TYPELIST_5(
    TYPELIST_1( ByName ),
    TYPELIST_2( ByYear, ByName ),
    TYPELIST_2( ByPlace, ByName ),
    TYPELIST_2(ByWage, ByName ),
    TYPELIST_2( ByWageDesc, ByName ))
    > PeopleData.
```

Перечислимый тип введен для удобства ссылки на порядковый номер столбца таблицы.

Первый параметр шаблона инструктирует компилятор, что колонками таблицы являются строка, целое число, строка и целое число, следующие в указанном порядке.

Второй параметр шаблона говорит о том, что таблица должна поддерживать сортировку по пяти различным критериям. При этом список типов, соответствующий, скажем, пятому критерию сортировки, говорит о том, что сортировать следует вначале по четвертой колонке в убывающем порядке, а затем – по первой колонке в возрастающем порядке.

Пример использования таким образом объявленной таблицы приводится ниже.

Вначале объявляются типы, определяющие колонки таблицы:

```
typedef Loki::Int2Type<name> Name;
typedef Loki::Int2Type<year> Year;
typedef Loki::Int2Type<place> Place;
typedef Loki::Int2Type<wage> Wage;.
```

Затем объявляется макрос, упрощающий заполнение таблицы:

```
#define SET_ROW(ROW, NAME, YEAR, PLACE, WAGE) \
table.SetCell<0, const char* >(ROW, NAME);\
```

table.SetCell<1, int >(ROW, YEAR); \
table.SetCell<2, const char*>(ROW, PLACE);\
table.SetCell<3, int >(ROW, WAGE);

В главной функции вначале создается таблица из четырех записей: PeopleData table(4).

Данная таблица заполняется информацией:
SET_ROW(0, "Smith", 1960, "USA", 1000);
SET_ROW(1, "Armstrong", 1975, "USA", 1200);
SET_ROW(2, "Lee", 1956, "China", 800);
SET_ROW(3, "Petroff", 1982, "Russia", 300);.

Теперь, чтобы вывести таблицу в первоначальном порядке следования записей, достаточно выполнить table.dump(), чтобы вывести только имя и место рождения, достаточно выполнить table.dump<TYPELIST_2(Name, Place)>(), чтобы вывести целиком таблицу, отсортированную по году/имени, достаточно выполнить table.sort<1>(); table.dump(), чтобы вывести таблицу, отсортированную по убыванию заработной платы, включив в нее только заработную плату и имя, достаточно выполнить table.sort<4>(); table.dump<TYPELIST_2(Wage, Name)>(), и т.д.

В последующих разделах приводятся краткие пояснения реализации данной функциональности.

Хранение данных в таблице

Основой для хранения данных в таблице служит кортеж, реализованный в классе Loki::Tuple. Указанный класс служит базовым для класса Row, объекты которого используются для хранения строк таблицы: template <class TList> class Row : public Loki::Tuple<TList>.

Список типов, передаваемый в шаблонном параметре, определяет типы данных, хранимых в строке таблицы.

Необходимость в таком классе вместо непосредственного использования Loki::Tuple вызвана потребностью вывода элементов кортежа, что не реализовано в Loki::Tuple.

Класс Table хранит строки в динамически выделяемом массиве Row<TList>* _rows.

Конкретизация класса Row определяется первым аргументом шаблона класса Table.

Доступ к отдельным ячейкам таблицы осуществляется через методы класса Loki::Tuple, обернутые в методы класса Table, дополнительно задающие номер строки, в которой расположена рассматриваемая ячейка.

Порядок сортировки в соответствии с различными критериями, задаваемыми вторым шаблонным аргументом класса Table, хранится в массиве индексов. Каждый индекс представлен экземпляром класса TableIndex. Индекс - это целочисленный массив, элементы которого указывают на строки таблицы, соответствующие позициям индекса.

Массив индексов объявлен следующим образом:
TableIndex_indexes[Loki::TL::Length <TIndexList>:: value].

Таблица хранит номер текущего активного индекса, и при доступе к строке на основе ее порядкового номера такая строка выбирается с использованием ссылки, хранимой в указанной позиции активного индекса,

Активный индекс устанавливается при помощи шаблонной функции template <int INDEX> void sort() с целочисленным параметром, определяющим критерий сортировки (номер индекса в массиве индексов). При этом индекс может перестраиваться, если он устарел, или использоваться как есть, если после очередной перестройки индекса не было изменений, делающих его неправильным.

Для построения индекса используется специальная адаптация функции быстрой сортировки qsort – функция qsortex. Смысл адаптации сводится к введению в функцию compare дополнительного аргумента, позволяющего передавать указатель this, тем самым давая возможность ее использования для сортировки нестатических данных класса. Сигнатура функции qsort модифицирована соответствующим образом, при этом никаких изменений в алгоритм сортировки не введено.

Вывод таблицы с избирательным выбором колонок

Вывод таблицы осуществляется функцией template < class T > void dump().

Шаблонным аргументом данной функции является список типов, элементами которого являются представленные в виде типов целые числа, соответствующие колонкам, которые следует выводить.

Нешаблонный вариант функции, осуществляющий вывод всех колонок, всего лишь вызывает шаблонную функцию dump, конкретизированную списком типов, полученным на основе первого шаблонного параметра класса Table: inline void dump() { dump <IndexList<TList>::Result >(); }.

Класс IndexList используется для преобразования произвольного списка типов в список вида

TYPELIST_N(Loki::Int2Type<0>,
Loki::Int2Type<1>, … Loki::Int2Type<N>),

где *N* – число элементов в списке TList.

Функция dump проходит по строкам таблицы и для каждой строки вызывает метод dump класса Row. В свою очередь функция Row::dump использует для вывода шаблонный класс Dumper: template <class TListTuple, class TListCols> class Dumper.

Этот класс реализует единственную статическую функцию DoDump, аргументом которой является указатель на строку (объект класса Row). Шаблонными аргументами класса являются: список типов, определяющий структуру строки, и список выводимых колонок в форме Int2Type-типов. При этом используется стандартный прием рекурсивного порождения кода на основе частичной специализации шаблонов, описанный в [1]. По сути, для каждого элемента из списка TListCols строится и вызывается код, осуществляющий вывод соответствующего элемента строки. При этом используется шаблонная функция DumpHelper, реализующая вывод элемента в зависимости от его типа.

Сортировка таблицы

Для сортировки таблицы вводится массив индексов. Размер массива равен числу элементов шаблонного параметра TIndexList класса Table. Каждый элемент такого массива представлен экземпляром класса TableIndex. В свою очередь класс TableIndex включает целочисленный массив с элементами, указывающими на номер строки таблицы в соответствии с заданной сортировкой.

Для сортировки таблицы используется метод sort с шаблонным аргументом, указывающим на критерий сортировки, который следует к ней применить. При этом строки таблицы не перемещаются, перестраивается только соответствующий индекс. Кроме того, данный индекс запоминается как активный для использования при выводе таблицы и выполнении других зависящих от текущей сортировки действий.

Замечание. Индекс перестраивается только в том случае, если в этом есть необходимость. Детали соответствующей реализации будут рассмотрены в следующем разделе.

Для сортировки применяется алгоритм qsort. Непосредственное использование функции qsort оказывается невозможным в связи с тем, что в функцию сравнения не удается передать указатель на класс, элементы которого сортируются. По этой причине сигнатура данной функции была изменена введением аргумента data. Результирующая функция qsortex была написана на основе исходного текста функции qsort путем внесения указанных выше изменений.

Наибольший интерес в связи с сортировкой представляет функция сравнения. В данном случае она реализована как шаблонная функция compare, которой в качестве шаблонного параметра передается список типов, определяющий критерий сортировки:

```
template <class TList1> inline static int compare (const void * elem1, const void * elem2, const void* data)
{
Row<TList>& row1 = ((Table<TList, TIndexList>*)data)->_rows[*(unsigned nt*)elem1];
Row<TList>& row2 = ((Table<TList, TIndexList>*)data)->_rows[*(unsigned int*)elem2];
return Compare<TList, List1> ::DoCompare (row1, row2);
}.
```

При вызове этой функции из qsortex в качестве elem1 и elem2 передаются указатели на элементы индекса, которые следует сравнить, а в качестве data – указатель на таблицу, строки которой должны использоваться для сравнения.

Сравнение как таковое обеспечивается шаблонным классом Compare: template <class TListTuple, class TListCols> class Compare.

Первым шаблонным аргументом является список типов, определяющий структуру объектов класса Tuple, которые следует сравнить. Второй шаблонный аргумент задает колонки (элементы из Tuple), которые должны участвовать в сравнении в заданном порядке.

С использованием частичной специализации шаблонов класс Compare обеспечивает рекурсивное построение кода, который сравнивает элементы строк, которые представленые в списке TListCols (функция DoCompare). При этом для сравнения отдельных элементов используется шаблонная функция CompareGeneral. В настоящей публикации данная функция реализована только для сравнения целых чисел и объектов классов (в последнем случае классы должны содержать реализацию метода compare).

Идея реализации функции DoCompare заключается в том, что она выполняет сравнение по элементу кортежа, определяемому головой списка сравниваемых элементов. Если они не равны, результат возвращается немедленно. При их равенстве вызывается функция DoCompare класса Compare, конкретизированного хвостом списка типов. Рекурсия продолжается до тех пор, пока в хвосте списка не окажется NullType.

Избирательная перестройка индексов

При выполнении функции sort нет необходимости всякий раз перестраивать индекс. Это необходимо делать лишь в случае, если после очередной перестройки индекса изменились данные в колонках таблицы, входящих в критерий сортировки для данного индекса. Иными словами, это означает, что необходимо добиться того, чтобы функция, меняющая значение в ячейке таблицы, выполняла также пометку нужных индексов как “грязных”. Тогда функция sort может перестраивать индекс только в случае, когда он помечен как “грязный”.

Данная задача решается следующим образом.

Создается класс ChangedSortIndexes, назначение которого состоит в том, чтобы по списку критериев сортировки и номеру колонки построить список из номеров индексов в виде Int2Type, которые следует перестроить в связи с изменением значения в указанной колонкt. Класс имеет следующее объявление: template <class TSortIndexList, typename TChangedCol, int Mode = -1, int ColCount = -1> struct ChangedSortIndexes.

При реализации ChangedSortIndexes используются вспомогательный класс HasCol, объявленный как template <class TList, typename T> struct HasCol.

Предполагается, что шаблонный аргумент TList данного класса содержит список из классов SortSpec, а T является номером колонки в виде Int2Type. Константа value данного класса оказывается равной 1, если в списке TList содержится SortIndex с константой column, равной номеру колонки, пердставленной параметром T. В противном случае значение константы оказывается равным 0.

Для того чтобы в классе HasCol можно было воспользоваться классом Loki::IndexOf, список из SortIndex вначале преобразуется в список из Int2Type, для чего используется класс ColNumbers.

Идея реализации класса ChangedSortIndexes состоит в следующем:

а) вводятся технические шаблонные аргументы Mode и ColCount.;

б) строятся три частичные специализации шаблона относительно аргумента Mode. Специализация для Mode = -1 выполняет роль модератора, а именно проверяет, содержит ли список, представленный параметром Head, колонку, заданную TChangedCol, в зависимости от чего применяет частичную специализацию для Mode = 0 или Mode = 1 соответственно. Кроме того, если данная специализация применяется впервые (ColCount = -1), запоминается первоначальная длина списка TSortIndexList, которая в дальнейшем передается последующим специализациям шаблона. Таким образом появляется возможность определить порядковый номер элемента Head относительно первоначального списка как ColCount - Loki::TL::Length<Tail>::value – 1;

в) Частичная специализация для Mode = 0 только применяет частичную спе-

циализацию для Mode = -1 относительно хвоста списка, тем самым обеспечивая рекурсию до исчерпания списка критериев сортировки;

г) Частичная специализация для Mode = 1, в дополнение к в), добавляет нужный элемент Int2Type в результирующий список типов. Таким образом, результирующий список типов содержит только элементы, добавленные данной частичной специализацией.

Получив возможность строить список изменившихся колонок, остается лишь применить данный список к пометке “грязных” индексов. Для этого используется класс RecurseAction, объявленный как template <class TList> struct RecurseAction.

Данный класс обеспечивает выполнение функции Do абстрактного класса Action для каждого элемента списка TList. При этом функции Do передается целое число, соответствующее очередному элементу TList (напомним, что элементами TList являются классы Int2Type).

Для пометки “грязных” индексов из класса Action наследуется класс ClearIndex, замещающий виртуальную функцию Do. Функция Do класса ClearIndex как раз и выполняет пометку соответствующего индекса как “грязного”.

Заключение

Основная цель данной статьи – изложить общие идеи возможности использования списков типов для реализации различной функциональности, относящейся к работе с таблицами, на примере вывода на печать и сортировки строк таблицы. Предложенный подход можно распространить и на другие задачи, в частности фильтрацию, группировку и связывание строк. Текущая реализация работы со строками таблицы носит, главным образом, иллюстративный характер. Как следствие, она весьма ограничена для практического использования, тем не менее, не представляет особой сложности применить более развитые механизмы работы со строками, тем самым расширить область приложений данного подхода.

1. *Александреску А*. Обобщенное программирование и прикладные шаблоны проектирования. - Москва: изд. дом ‘Вильямс’, 2002. –245 с.

Получено 05.12.05

Об авторе

Марьянович Олег Тадеушевич
канд. физ.-мат. наук

Место работы
Институт кибернетики им. В. М. Глушкова НАН Украины. 03680, МСП, Киев-187,
просп. Акад. Глушкова, 40, отд. 145.
Тел. 526 3603, Факс 526 1558.
Email OMarianovich@miratech.ua.