Тема 9. Библиотека STL

В язык C++ была включена так называемая библиотека стандартных шаблонов (Standard Template Library, STL). Библиотека содержит универсальные шаблонные классы и функции, которые реализуют широкий спектр алгоритмов и структур данных. Библиотека основана на механизме шаблонов template и тем самым позволяет работать с самыми разными объектами и типами данных.

Библиотека вводит следующие понятия, которые реализуются в виде программного кода и составляют компоненты этой библиотеки:

контейнеры (container)
итераторы (iterator)
алгоритмы (algorithm)
функциональные объекты или функторы (functor).

Контейнеры.

Контейнер — это объект, используемый для хранения других (однотипных) объектов, которые могут быть как базовыми типами, так и экземплярами разноообразных классов. Общими операциями с контейнером являются добавление, удаление, поиск и доступ к элементам.

В библиотеке STL реализованы следующие контейнеры:

последовательные:

array
vector
vector<bool> — битовое множество;
deque — дек;
list — линейный двунаправленный список;
forward_list — линейный однонаправленный список;

ассоциативные:

map — ассоциативный контейнер, построенный по принципу key:value, в котором каждому ключу key соответствует значение value;

multimap — ассоциативный контейнер, в котором одному значению (key) может соответствовать несколько значений;

set — множество,

multiset — множество, в котором один и тот же элемент может встречаться несколько раз;

адапторы:

stack — стек;
queue — обычная очередь;
priority_queue — очередь с приоритетами;

хеш-реализации:

unordered_set
unordered_multiset
unordered_map
unordered_muptimap

Практически все эти структуры рассматривились нами на лекциях. Поэтому не будем останавливаться на их допустимых операциях, поскольку о них можно прочитать в любой справочной литературе. Основная идея работы с контейнерами состоит в использовании итераторов для доступа к элементам. Так, функции поиска элементов возвращают итераторы, по которым далее можно получить доступ к элементам.

Простой пример map.cpp.

```
#include <iostream>
#include <map>
```

```
using namespace std;

int main()
{
   multimap<int, char> m;
   m.insert(std::pair<int, char>(1, 'a'));
   m.insert(std::pair<int, char>(2, 'b'));
   m.insert(std::pair<int, char>(2, 'c'));
   m.insert(std::pair<int, char>(2, 'd'));
   m.insert(std::pair<int, char>(1, 'e'));
   m.insert(std::pair<int, char>(3, 'f'));

   for (auto i = m.find(2); i != m.end(); ++i) {
       cout << "find " << i->first << " -> " << i->second << endl;
   }

   auto r = m.equal_range(2);
   for (auto i = r.first; i != r.second; i++) {
       cout << "range find " << i->first << " -> " << i->second << endl;
   }
   return 0;
}
```

Итераторы.

Итератор — это абстракция, формализующая последовательный доступ к элементам контейнера через “позицию” элемента в контейнере. Итераторы для разных контейнеров могут различаться по своей функциональности, но все они так или иначе реализуют операции доступа к значению (* или ->), перемещения (например, ++ или −−) и сравнения с другими итераторами (в простейшем случае на равно или не равно). Итераторы могут быть константными и неконстантными.

О переопределении оператора ->.

Этот оператор используется для доступа к отдельным полям класса через указатель на этот класс. Можно, конечно, сначала разименовать указатель, а потом уже добираться к членам класса через операцию "точка но "стрелочка"смотрится нагляднее и короче.

Поскольку итератор является некоторым аналогим указателя на элемент контейнера, то логично, чтобы оператор -> возвращал указатель на этот элемент. Вот упрощенный пример, как это может выглядеть.

```
struct Vec {
   int x, y;
   Vec(int xx, int yy) : x(xx), y(yy) {}
};

struct  Ptr
{
   Vec * p;
   Ptr(Vec *pp) : p(pp) {}
   Vec * operator->() { return p; }
   Vec & operator*() { return *p; }
};

int main()
```

```
{
    Vec v(10,20);
    Ptr p(&v);
    cout << "x=" << (*p).x << " y=" << (*p).y << endl;
    cout << "x=" << p->x << " y=" << p->y << endl;
    cout << "x=" << (p.operator->())->x << " y=" << p.operator->()->y << endl;
    return 0;
}
```

Из этого примера видно, что запись `p->x` вычисляется по правилу `p.operator->()->x`.

В C++ различают 5 видов итераторов:

RandIter — итератор произвольного доступа, записывает и извлекает значение из контейнера. Обеспечивает произвольный доступ к элементам контейнера;

BiIter — двунаправленный итератор, записывает и извлекает значения. Перемещается вперед и назад;

ForIter — прямой итератор — перемещается только вперед. Записывает и извлекает значения;

InIter — итератор ввода, только извлекает значения. Перемещается только вперед;

OutIter — итератор вывода, только записывает значения. Перемещается только вперед.

Первый итератор работает, например, с массивами vector. Двунаправленный итератор работает с большинством остальных контейнеров. Последние два итератора предназначены для классов потокового ввода/вывода, которые реализуют последовательный доступ к данным.

В C++ также имеется понятие интервала (range), которое определяется как два итератора, задающих позицию первого элемента и позицию после последнего элемента.

Также есть специальные итераторы (адаптеры). Например, так называемый итератор вставки. Он позволяет более эффективно организовать добавление элементов, сокращая количество промежуточных объектов.

Функторы.

Абстракции контейнера и итератора как объектов, позволяющих работать с любыми типами данных, далее дополняется понятием функтора, как некоторой абстракции для операций, которые, возможно, будут выполняться с этими типами.

Функтор — это класс, в котором реализована функция operator(). Таким ообразом, класс может выступать как некоторая функция, которая теперь может не только вычислять значение, но и сохранять свое внутренее состояние.

Библиотека STL определяет некоторое количество функторов, которые делятся на две категории:

бинарные — содержат два аргумента;

унарные — содержат один аргумент.

STL содержит следующие бинарные функторы:

plus — суммирует (+) два аргумента;

minus — вычитает (–) аргументы;

multplies — умножает (*) два аргумента;

divides — делит (/) аргументы;

modulus — возвращает результат операции % для двух аргументов;

equal_to — сравнивает аргументы на равенство (==);

not_equal_to — сравнивает аргументы на неравенство (!=);

greater — определяет, больше ли первый аргумент чем второй аргумент (>);

greater_equal — определяет, первый аргумент есть больше или равен второму аргументу (>=);

less — определяет, меньше ли первый аргумент чем второй аргумент (<);
less_equal — определяет, первый аргумент меньше или равен второму аргументу (<=);
logical_and — применяет к аргументам логическое «И» (AND);
logical_or — применяет к двум аргументам логическое «ИЛИ» (OR).
Также определены два унарных функтора:
logical_not — применяет к аргументу логическое «НЕТ» (NOT);
negate — изменяет знак своего аргумента на противоположный.
Такие функторы используются в стандартных алгоритмах (см. далее). Они определены для базовых типов, а пользователь может определить их для своих типов и тем самым получить возможность применять стандартные алгоритмы к своим наборам данных.

Алгоритмы.

Алгоритмы позволяют обрабатывать данные из контейнеров. Например, инициализировать, сортировать, преобразовывать, реализовывать различные виды поиска и т.п.

Все алгоритмы являются шаблонными функциями. Они могут быть применены к любому типу контейнера. В STL есть, например, такие алгоритмы:
adjacent_find — осуществляет поиск пары соседних элементов, которые совпадают между собой;
binary_search — выполняет бинарный поиск в упорядоченной последовательности;
copy — копирует одну последовательность в другую;
copy_backward — делает то же, что и copy, только результирующая последовательность записывается в обратном порядке;
count — подсчитывает количество вхождений заданного элемента в последовательности;
count_if — вычисляет количество вхождений элемента в последовательности, соответствующей заданному условию;
equal — определяет, совпадают ли элементы двух диапазонов;
equal_range — возвращает диапазон, который допускает вставку элементов без нарушения порядка;
fill, fill_n — заполняют диапазон нужными значениями;
find — осуществляет поиск элемента, возвращает позицию (итератор) первого вхождения элемента в последовательности;
(и еще несколько вариантов поиска ...)
for_each — применяет указанную функцию к заданному диапазону элементов;
generate, generate_n — присваивают элементам диапазона значения, возвращаемые функцией-генератором;
include — определяет, содержит ли одна последовательность другую;
inplace_merge — объединяет два диапазона;
make_heap — создает пирамиду на основе заданной последовательности;
max — возвращает максимум из двух значений;
max_element — возвращает позицию (итератор) на максимальный элемент последовательности;
min, min_element
merge — объединяет две упорядоченные последовательности. Результат записывается в третью последовательность;
next_permutation — формирует следующую перестановку элементов последовательности;
remove — удаляет элементы, которые имеют определенное значение;
remove_if — удаляет элементы по заданному предикату;
replace — заменяет элементы заданного диапазона из одного значения на другое;

reverse — изменяет порядок элементов последовательности на противоположный в пределах заданного диапазона;

rotate — выполняет циклический сдвиг влево элементов последовательности в заданных пределах;

search — выполняет поиск подпоследовательности в последовательности;

set_difference, set_intersection, set_symmetric_difference, set_union — операции как с множествами;

sort, stable_sort — сортировки;

swap — меняет местами значения, заданные ссылками;

transform — для заданной последовательности применяет функцию к каждому элементу;

unique — удаляет дубликаты из указанной последовательности;

и кое-какие другие ...

Алгоритмы обычно требуют указания диапазона элементов контейнера (в виде итераторов) и требуемой функции (в виде функтора).

Простые примеры

— сортировка sort.cpp

— удаление remove.cpp

Аллокаторы.