**Последовательные контейнеры**

Рассмотрим один из трех последовательных контейнеров – списки. Контейнер STL список (list) представляет собой дважды связанный (двусвязанный) список, в котором каждый элемент хранит указатель на соседа слева и справа. Списки не поддерживают произвольный доступ к элементам.

Функции push\_front() и push\_back() вcтавляют данные соответственно в начало и конец списка.

Функции pop\_front() и pop\_back() удаляют данные соответственно из начала и конца списка.

Рассмотрим пример вставки, чтения и извлечения данных.

**Пример**: // демонстрация методов push\_front(), front(), pop\_front()

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<int> ilist;

ilist.push\_back(30); // вставка элементов в конец

ilist.push\_back(40);

ilist.push\_front(20); // вставка элементов в начало

ilist.push\_front(10);

int size = ilist.size(); // число элементов

for (int j = 0; j < size; j++)

{

cout << ilist.front() << ' '; // читать данные из начала

ilist.pop\_front(); // извлечение данных из начала

}

cout << endl;

system("pause");

return 0;

}

Результаты выполнения программы:

10 20 30 40

**Итераторы**

Состав операций, который поддерживает итератор, зависит от внутренней структуры контейнера.

Существует три основных вида итераторов:

1. *Прямой итератор* (движение поэлементно вперед (++)).
2. *Двунаправленный* (движение вперед и назад (++, --)).
3. *Произвольного доступа* (движение вперед и назад с *перескоком,* можно прибавлять и вычитать смещения, обрабатывать разность и сравнивать итераторы).

Итераторы контейнерных классов list, set, multiset, map и multimap являются двунаправленными.

Итераторы контейнерных классов vector и deque, а также итераторы строк являются итераторами произвольного доступа.

Есть еще два специализированных вида итераторов:

1. *Итератор ввода* – указывает на устройство ввода (***cin*** или просто входной файл) и считывает последовательно элементы данных в контейнер.

*ostream\_iterator<тип данных>(cout*, “текст”) [текст – обычно пробел].

1. *Итератор вывода* – указывает на устройство вывода (***cout*** или выходной файл) и выводит элементы из контейнера.

Значения прямых, двунаправленных итераторов и произвольного доступа могут быть сохранены. Значения итераторов ввода и вывода сохраняться не могут.

|  |  |
| --- | --- |
| **Итератор** | **Вид доступа** |
| Произвольного доступа | Обеспечивает произвольный доступ к элементам. |
| Двунаправленный | Перемещается вперед и назад. |
| Прямой | Перемещается только вперед. |
| Ввода (для считывания значений) | Перемещается только вперед. |
| Вывода (для записи) | Перемещается только вперед. |

Итераторы, обладающие более широкими возможностями, можно использовать вместо более “слабых итераторов”. Например, вместо итератора ввода можно применять прямой итератор.

Итераторы действуют как указатели. Их можно увеличивать, уменьшать и разыменовывать, применяя оператор "\*".

В STL существуют *обратные итераторы* (reverse iterator). Они могут быть либо двунаправленными, либо итераторами произвольного доступа. Если обратный итератор ссылается на конец последовательности, его увеличение приведет к перемещению на предыдущий элемент, то есть эти итераторы работают в противоположном направлении. Все контейнеры поддерживают создание обратных итераторов функциями rbegin() и rend().

**Соответствие итераторов и контейнеров**

Каждый контейнер сам определяет типы своих итераторов, поэтому для работы с итераторами контейнеров специальные заголовочные файлы не нужны. При определении итератора надо только указать, для какого контейнера его надо использовать. Однако существует несколько специальных итераторов (например, обратных), которые определяются в заголовочном файле <iterator> .

Пример:

list<int> iList; // определили список

list<int>::iterator iter; // итератор для целочисленного списка

***Метод copy()***

1. Выводит данные.
2. Добавляет данные из контейнера A в контейнер B.
3. Копирование диапазона ячеек и вставка в нужную позицию.

***Метод insert()***

Помещает данные в список

***Метод erase()***

Удаляет данные из интервала

***Метод remove()***

Удаление элемента

***Метод remove\_if()***

Удаление элементов по условию

Пример: // Использование итератора вставки (insert) и вывода (ostream\_iterator)

#include <iostream>

#include <list>

#include<iterator>

using namespace std;

bool f\_remove(const int& val) {

return val > 15;

}

int main()

{

setlocale(LC\_ALL, "RUS");

list<int> ilist = { 5, 7, 9, 12, 13, 45, 12, 27, 13 };

list<int>::iterator it;

it = ilist.begin();

advance(it,3); // сдвигаем итератор на нужную позицию (+3)

ilist.insert(it, 101);

copy(ilist.begin(), ilist.end(), ostream\_iterator<int>(cout, " "));

cout << endl;

ilist.remove(12); // удаление числа 12

copy(ilist.begin(), ilist.end(), ostream\_iterator<int>(cout, " "));

cout << endl;

ilist.remove\_if(f\_remove); // удаление с условием

copy(ilist.begin(), ilist.end(), ostream\_iterator<int>(cout, " "));

cout << endl;

//ilist.erase(ilist.begin(),--ilist.end());

it = ilist.end();

advance(it, -4);

ilist.erase(ilist.begin(),it);

copy(ilist.begin(), ilist.end(), ostream\_iterator<int>(cout, " "));

cout << endl;

system("pause");

return 0;

}

***Задание:*** Вывести список, начиная со 2-го значения а виде вектора-столбца.

**Пример**: // Итератор для заполнения контейнера данными

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<int> iList(5); // пустой список для хранения 5

// значений типа int

list<int>::iterator it; // итератор

int data = 0;

// заполнение списка данными

for (it = iList.begin(); it != iList.end(); it++)

\*it = data += 2;

// вывод списка

for (it = iList.begin(); it != iList.end(); it++)

cout << \*it << ' ';

cout << endl;

system("pause");

return 0;

}

Результаты выполнения программы:

2 4 6 8 10

**Алгоритмы и итераторы**

Алгоритмы используют итераторы в качестве параметров, а иногда в качестве возвращаемых значений. Рассмотрим пример применения алгоритма find() к списку.

**Пример**: // find() возвращает итератор списка

#include <iostream>

#include <algorithm>

#include <list>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUS");

list<int> theList(5); // пустой список для 5 значений int

list<int>::iterator iter; // итератор

int data = 0;

// заполнение списка данными

for (iter = theList.begin(); iter != theList.end(); iter++)

\*iter = data += 2; //2, 4, 6, 8, 10

// поиск числа 8

iter = find(theList.begin(), theList.end(), 8);

if (iter != theList.end())

cout << "\nНайдено число 8.\n";

else

cout << "\nЧисло 8 не найдено.\n";

system("pause");

return 0;

}

Результаты выполнения программы:

Найдено число 8