C++

Урок 25

Повтороение

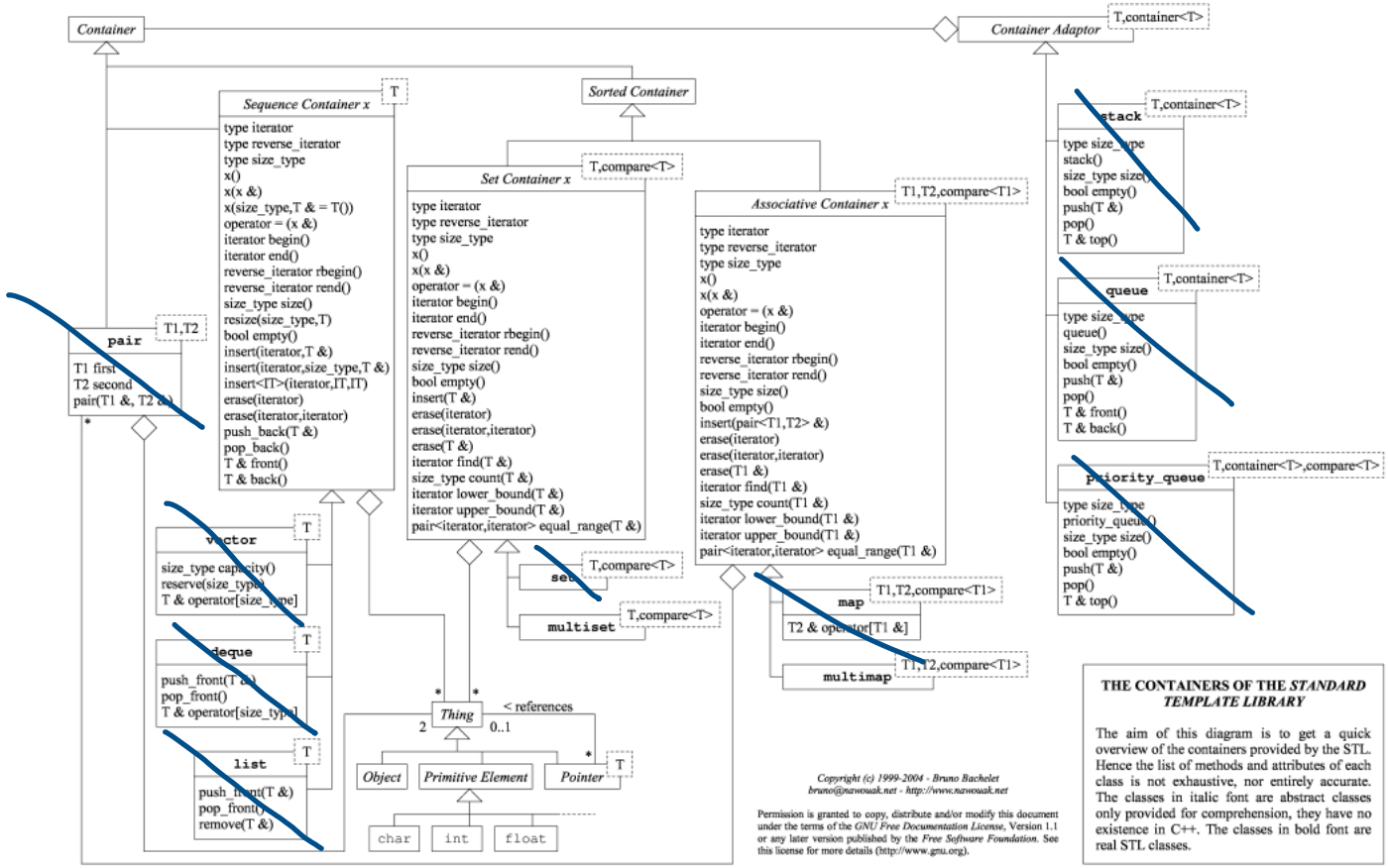
Как считать данные из файла?

Как считать построчно?

Контейнеры 4 часть

STL

Контейнеры есть стандартные структуры данных, такие как список (**list**), вектор (**vector**), словарь (**map**) и многие другие.



multimap

multimap

Класс **multimap** стандартной библиотеки C++ используется для хранения и извлечения данных из коллекции, в которой каждый элемент является **парой** со значением данных и ключом сортировки.

Класс multimap реализует ассоциативный массив, в котором **одному ключ**у могут соответствовать **несколько значений**.

Класс содержит такие же методы, что и класс **map**.

multimap

- Ассоциативный контейнер, который является контейнером переменного размера, поддерживающим эффективное получение значений элементов на основе значения соответствующего ключа.
- Реверсивный, поскольку он предоставляет двунаправленные итераторы для получения доступа к его элементам.
- Сортированный, поскольку его элементы упорядочены по значениям ключей в контейнере в соответствии с заданной функцией сравнения.
- Несколько, так как его элементы не должны иметь уникальный ключ, поэтому одно значение ключа может иметь множество значений данных элемента, связанных с ним.
- Является контейнером ассоциативной пары, поскольку его значения данных элементов отличаются от его значений ключей.
- Шаблон класса, так как функциональность, которая предоставляется, является универсальной и поэтому независимо от конкретного типа данных, содержащихся в виде элементов или ключей. Типы данных, используемые для элементов и ключей, вместо этого определяются как параметры в шаблоне класса вместе с функцией и распределителем сравнения.

```cpp
// CPP Program to demonstrate the implementation of multimap
#include <iostream>
#include <iterator>
#include <map>
using namespace std;

// Driver Code
int main()
{
    multimap<int, int> gquiz1; // empty multimap container

    // insert elements in random order
    gquiz1.insert(pair<int, int>(1, 40));
    gquiz1.insert(pair<int, int>(2, 30));
    gquiz1.insert(pair<int, int>(3, 60));
    gquiz1.insert(pair<int, int>(6, 50));
    gquiz1.insert(pair<int, int>(6, 10));

    // printing multimap gquiz1
    multimap<int, int>::iterator itr;
    cout << "\nThe multimap gquiz1 is : \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr)
    {
        cout << '\t' << itr->first << '\t' << itr->second
             << '\n';
    }
    cout << endl;

    // adding elements randomly,
    // to check the sorted keys property
    gquiz1.insert(pair<int, int>(4, 50));
    gquiz1.insert(pair<int, int>(5, 10));

    // printing multimap gquiz1 again

    cout << "\nThe multimap gquiz1 after adding extra "
            "elements is : \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr)
    {
        cout << '\t' << itr->first << '\t' << itr->second
             << '\n';
    }
    cout << endl;

    // assigning the elements from gquiz1 to gquiz2
    multimap<int, int> gquiz2(gquiz1.begin(), gquiz1.end());

    // print all elements of the multimap gquiz2
    cout << "\nThe multimap gquiz2 after assign from "
            "gquiz1 is : \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)
    {
        cout << '\t' << itr->first << '\t' << itr->second
             << '\n';
    }
    cout << endl;

    // remove all elements up to
    // key with value 3 in gquiz2
    cout << "\ngquiz2 after removal of elements less than "
            "key=3 : \n";
    cout << "\tKEY\tELEMENT\n";
    gquiz2.erase(gquiz2.begin(), gquiz2.find(3));
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)
    {
        cout << '\t' << itr->first << '\t' << itr->second
             << '\n';
    }

    // remove all elements with key = 4
    int num;
    num = gquiz2.erase(4);
    cout << "\ngquiz2.erase(4) : ";
    cout << num << " removed \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)
    {
        cout << '\t' << itr->first << '\t' << itr->second
             << '\n';
    }

    cout << endl;

    // lower bound and upper bound for multimap gquiz1 key =
    // 5
    cout << "gquiz1.lower_bound(5) : "
         << "\tKEY = ";
    cout << gquiz1.lower_bound(5)->first << '\t';
    cout << "\tELEMENT = " << gquiz1.lower_bound(5)->second
         << endl;
    cout << "gquiz1.upper_bound(5) : "
         << "\tKEY = ";
    cout << gquiz1.upper_bound(5)->first << '\t';
    cout << "\tELEMENT = " << gquiz1.upper_bound(5)->second
         << endl;

    return 0;
}
```

Инициализация + методы + вывод

Iterators

`begin` `cbegin` (C++11)	returns an iterator to the beginning (public member function)
`end` `cend` (C++11)	returns an iterator to the end (public member function)
`rbegin` `crbegin` (C++11)	returns a reverse iterator to the beginning (public member function)
`rend` `crend` (C++11)	returns a reverse iterator to the end (public member function)

Capacity

`empty`	checks whether the container is empty (public member function)
`size`	returns the number of elements (public member function)
`max_size`	returns the maximum possible number of elements (public member function)

Modifiers

`clear`	clears the contents (public member function)
`insert`	inserts elements or nodes (since C++17) (public member function)
`emplace` (C++11)	constructs element in-place (public member function)
`emplace_hint` (C++11)	constructs elements in-place using a hint (public member function)
`erase`	erases elements (public member function)
`swap`	swaps the contents (public member function)
`extract` (C++17)	extracts nodes from the container (public member function)
`merge` (C++17)	splices nodes from another container (public member function)

Lookup

`count`	returns the number of elements matching specific key (public member function)
`find`	finds element with specific key (public member function)
`contains` (C++20)	checks if the container contains element with specific key (public member function)
`equal_range`	returns range of elements matching a specific key (public member function)
`lower_bound`	returns an iterator to the first element *not less* than the given key (public member function)
`upper_bound`	returns an iterator to the first element *greater* than the given key (public member function)

https://en.cppreference.com/w/cpp/container/multimap

Документация multimap

Multiset

Мультимножество

multiset — это контейнер, который также будет содержать элементы в отсортированном порядке при добавлении, но он хранит **повторяющееся** элементы, по сравнению с множеством set.

Часто его называют **мультимножество**.