**Bài 11: Map - Khái niệm cơ bản - Sử dụng thư viện chuẩn STL cho C/C++**

**Đăng bởi: Admin | Lượt xem: 10930 | Chuyên mục: C/C++**

**Định nghĩa về Map :**

Các lớp vector, list thuộc cấu trúc Sequence Containers (cấu trúc tuần tự), riêng với lớp map thuộc một cấu trúc khác đó là Associative Containers (cấu trúc liên kết), là kiểu dữ liệu cho phép quản lý một cặp key/value - khóa và giá trị, nghĩa là muốn xác định được nội dung value thì phải biết được vị trí key mà map đang quản lý.

Có lẽ trước khi biết đến map trong C++, bạn đã sử dụng qua rất nhiều các cấu trúc dữ liệu quen thuộc khác như cấu trúc mảng (array), danh sách liên kết (linked list), cấu trúc stack, queue, ..v.v. thì map cũng là một cấu trúc dữ liệu tương tự nằm trong thư viện chuẩn std của C++.

Map cũng chính như cái tên của nó, giúp bạn mường tượng được rằng nó là một tập hợp của nhiều phần tử như cấu trúc mảng, nhưng mỗi phần tử trong nó không đơn thuần chỉ là một số hoặc một giá trị cụ thể nào đó (như trong cấu trúc mảng), mà mỗi phần tử chính là một cặp bao gồm giá trị bắt đầu gọi là key (tạm dịch là chìa khóa) và giá trị đích value (tạm dịch là giá trị ẩn), chúng ta cần giá trị bắt đầu để dẫn đường tới thứ ẩn đằng sau nó.

Lấy cuốn từ điển làm ví dụ, chúng ta có một từ không hiểu, gọi từ đó là key, dùng từ đó để tra trong từ điển, nó sẽ dẫn tới một đoạn văn giải thích nào đó, thì đoạn văn đó gọi là value. Ví dụ, "nước": "một loại chất lỏng không mùi không vị".

Trong lập trình, khi bạn muốn dùng một định danh nào đó thay thế cho giá trị ẩn thật sự đằng sau nó, ví dụ dùng từ "một" để thay thế cho số "1", thì bạn nên nghĩ ngay tới việc sử dụng cấu trúc map.

Diagram

Description automatically generated with low confidence

Một điểm đặc biệt nữa là mỗi key trong map là duy nhất và không thể bị trùng lắp, trong khi giá trị ẩn của key đó thì có thể bị trùng lắp với giá trị ẩn của key khác, đây cũng là điều dễ hiểu bởi vì nếu chúng ta có hai điểm bắt đầu, chúng ta sẽ phân vân không biết chọn điểm bắt đầu nào, đó là một trong những tính chất quan trọng của map.

Nếu bạn muốn lưu danh sách các phần tử không bị trùng lắp hoặc đếm số lần xuất hiện của mỗi từ trong câu thì sao, map là sự lựa chọn hoàn hảo.

**Cách sử dụng Map trong stl :**

Bạn có thể tìm thấy cấu trúc map trong thư viện <map> của C++.

Sử dụng khá đơn giản, khai báo một cấu trúc map như sau:

map<kiểu dữ liệu của key, kiểu dữ liệu của value> someMap;

Trong đó kiểu dữ liệu của key thường là string, kiểu dữ liệu của value thường là giá trị số như int, float, ..v.v. someMap là tên bạn muốn đặt cho map.

Ví dụ:

#**include** <iostream>

#**include** <map>

**using** **namespace** std;

**int** **main**() {

map<string, **int**> A; // Khởi tạo một map A

// Thêm vào map A một số phần tử.

A["One"] = 1;

A["Two"] = 2;

A["Three"] = 3;

// Duyệt và xuất ra từng cặp giá trị key: value của mỗi phần tử trong map A

map<string, **int**>::iterator it;

**for** (it = A.begin(); it != A.end(); it++) {

cout << it->first.c\_str() << ": " << it->second << endl;

}

// Thêm vào một phần tử giá trị key trùng lặp, chuyện gì sẽ xảy ra?

A["One"] = 10;

// In kết quả map A ra màn hình sau khi thêm vào 1 key trùng lặp,

// tự chạy code xem kết quả của chính bạn.

**for** (it = A.begin(); it != A.end(); it++) {

cout << it->first.c\_str() << ": " << it->second << endl;

}

}

**Thành phần Map :**

Nếu như lớp map quản lý cặp đối tượng key/value, vậy đối tượng nào sinh ra được cặp đối tượng đó. Để sinh ra cặp đối tượng key/value cần sử dụng lớp pair, lớp pair nằm trong thư viện utility. Trước khi tìm hiểu về lớp map cần biết về lớp pair.

**1. Lớp pair :**

*pair*cho phép gộp 2 đối tượng thành một cặp, 2 đối tượng có thẻ cùng kiểu hoặc khác kiểu, với thuộc tính *first*là key và *second*là value.

Cú pháp khai báo:*pair<valueType1, valueType2> variableName*;

ví dụ :

pair<string, string> dictionary;

*pair*có các constructor như:

* default constructor
* copy constructor
* initialization constructor

Ví dụ:

// default constructor

pair<**int**, **char**\*> defaultPair;

// initialization constructor

pair<**int**, **char**\*> initPair(0, "a");

// copy constructor

pair<**int**, **char**\*> copyPair(pair1);

Lớp *pair*có các thuộc tính *first*và *second*cho phép lấy dữ liệu :

pair<**string**, **string**> **word**("eat", "an");

cout << word.first << " " << word.second;

**2 . Lớp map :**

Lớp *map*nằm trong thư viện *map*vì vậy muốn sử dụng trước tiên phải *#include <map>.*

Cú pháp khai báo: *map<valueType1, valueType2> variableName;*

Ví dụ:

map<string, string> dictionary;

dictionary["eat"] = "an";

dictionary["sleep"] = "ngu";

Biến *dictionary*được khai báo với cặp dữ liệu là*<string, string>*, vì vậy:

* **key**của dictionary phải là kiểu*string - "eat" "sleep"...*
* ***value****của dictionary phải là kiểu string - "an" "ngu"...*

Lớp *map*cũng giống như những lớp*vector, list* đều được định nghĩa các hàm thành viên hỗ trợ cho việc truy xuất, lấy kích thước, các *constructor*đã được*override, …*

**Bài 12: Map - Bài tập cơ bản - Sử dụng thư viện chuẩn STL cho C/C++**

**Đăng bởi: Admin | Lượt xem: 3782 | Chuyên mục: C/C++**

**Các phương thức thường dùng trong map**

**1 . Constructor :**

Cũng giống với những lớp *vector*, *list*đều có:

* default constructor
* ranger constructor
* copy constructor.

Các hàm thành viên này hoạt động hoàn toàn giống với lớp *vector*, *list*.

Ví dụ :

// default constructor

map<**char**, **int**> character0;

character0['a'] = 97;

character0['b'] = 98;

character0['c'] = 99;

character0['d'] = 100;

// range constructor

map<**char**, **int**> **character1**(character0.begin(), character0.**end**());

// copy constructor

map<**char**, **int**> **character2**(character1);

Ngoài ra còn có *operator*= cho phép các lớp map có thể sao chép nội dung cấu trúc dữ liệu với nhau.

**map**<**int**, char\*> **map**0, map1;

**map**0[0] = "a";

**map**0[1] = "b";

map1 = **map**0;

**2 . Duyệt map :**

Lớp *map*quản lý value bằng key, nếu trường hợp quên *map*thì làm sao có thể truy xuất để lấy dữ liệu *value*. Để giải quyết vấn đề trên trong lớp *map*định nghĩa thao tác *iterators* cho phép truy xuất đến phần tử trong map để lấy dữ liệu *value*và *key*cần thiết. Có các *iterators*đã được cung cấp như: *begin*(), *end*(), *rbegin*(), *rend*(), ... chúng hoạt động như nhau nhưng chỉ khác phần tử truy cập ở vị trí nào trong lớp map.

Ví dụ :

map<**int**, **char**\*> texes;

texts[0] = "ab";

texts[1] = "bc";

texts[0] = "cd";

map<**int**, **char**\*>::iterator i;

**for** (i = texts.begin(); i != texts.end(); i++)

cout << \*i << endl;

// output (0, "cd"), (1, "bc")

Kết quả tại sao không phải là:*(0, "ab") (1, "bc") hay (0, "ab") (1, "bc") (0, "cd"), ...* đó là do trong lớp map quy định key chỉ được tồn tại duy nhất (không được trùng), vì vậy khi khai báo ở dòng *texts[0] = "cd"*; sẽ làm thay đổi dữ liệu của*texts[0]*từ "ab" thành "cd".

size() cho phép lấy kích thước của map.

Ví dụ:

map<**int**, **char**\*> mymap, copymymap;

mymap[0] = "a";

mymap[1] = "b";

mymap[0] = "c";

cout << mymap.size();

**3. Kiểm tra map có rỗng hay không?**

empty() cho phép ta kiểm tra map có rỗng hay không?

Ví dụ:

map<**int**, **char**\*> mymap, copymymap;

mymap[0] = "a";

mymap[1] = "b";

mymap[0] = "c";

**if** (copymymap.empty()) cout << "copymymap is empty";

**4. Truy xuất theo chỉ số**

[index] hay at(index) cho phép truy xuất trực tiếp đến key của từng phần tử.

map<**int**, **char**\*> mymap, copymymap;

mymap[0] = "a";

mymap[1] = "b";

cout << mymap[1] << " " << mymap.at(0);

**5. Thêm dữ liệu vào map**

insert() cho phép chèn thêm một đối tượng.

Ví dụ:

map<int, char\*> mymap, copymymap;

mymap[0] = "a";

mymap[1] = "b";

mymap[5] = "c";

// chèn vào copymymap cặp đối tượng (10, "c")

copymymap.insert(pair<int, char\*>(10, "c"));

// chèn (-1, "d") vào copymymap từ vị trí bắt đầu của copymymap

copymymap.insert(copymymap.**begin**(), pair<int, char\*>(-1, "d"));

// chèn mymap vào copymymap

copymymap.insert(mymap.**begin**(), mymap.**end**());

// => copymymap = {(-1,"d"),(0,"a"),(1,"b"),(5,"c"),(10,"c")}

Lưu ý việc chèn ở vị trí nào thật ra là vô nghĩa, vì vị trí của nó phụ thuộc vào key trong cặp đối tượng "key/ value", trong ví dụ trên thì "key" là kiểu số nguyên vì vậy vị trí của các cặp đối tượng key/value sẽ được xác định theo các key sắp xếp tăng dần.

**6. Tìm kiếm phần tử**

find() cho phép tìm kiếm theo key của cặp giá trị key/value

Ví dụ:

map<**int**, **char**\*> mymap;

mymap[0] = "a";

mymap[1] = "b";

mymap[5] = "c";

mymap[9] = "e";

map<**int**, **char**\*>::iterator var = mymap.find(5);

// var -> (5,"c")

**7. Xóa bỏ 1 phần tử trong map**

erase() cho phép xóa một phần tử.

Ví dụ:

map<**int**, **char**\*> mymap, copymymap;

mymap[0] = "a";

mymap[1] = "b";

mymap[5] = "c";

mymap[7] = "d";

mymap[9] = "e";

// xóa cặp đối tượng với "key" là 5

mymap.erase(5);

// => mymap = {(0,"a"),(1,"b"),(7,"d"),(9,"e")}

map<**int**, **char**\*>::iterator var = mymap.begin();

// xóa cặp đối tượng mà var đang truy cập

mymap.erase(var); // => mymap = {(1,"b"),(7,"d"),(9,"e")}

var = mymap.find(7); // => var truy cập đến (7,"d")

// xóa từ vị trí var đang truy cập cho đến (9,"e")

mymap.erase(var, mymap.end()); // => mymap = {(1,"b")}

**8. Xóa tất cả phần tử trong map**

clear() cho phép xóa tất cả phần tử.

Ví dụ:

map<**int**, **char**\*> mymap;

mymap[0] = "a";

mymap[1] = "b";

mymap[5] = "c";

mymap[7] = "d";

mymap[9] = "e";

mymap.clear();

// => mymap = {}

**9. Hoán đổi nội dung 2 map**

swap() cho phép hoán đổi nội dung của 2 map.

Ví dụ:

**map**<char,**int**> map1, map2;

map1['x'] = 100;

map1['y'] = 200;

map2['a'] = 11;

map2['b'] = 22;

map2['c'] = 33;

map1.swap(map2);

// => map1 = {("x",11),("b",22),("c",33)}

// => map2 = {("x",100),("y",200)}