

第四次上机

题目描述

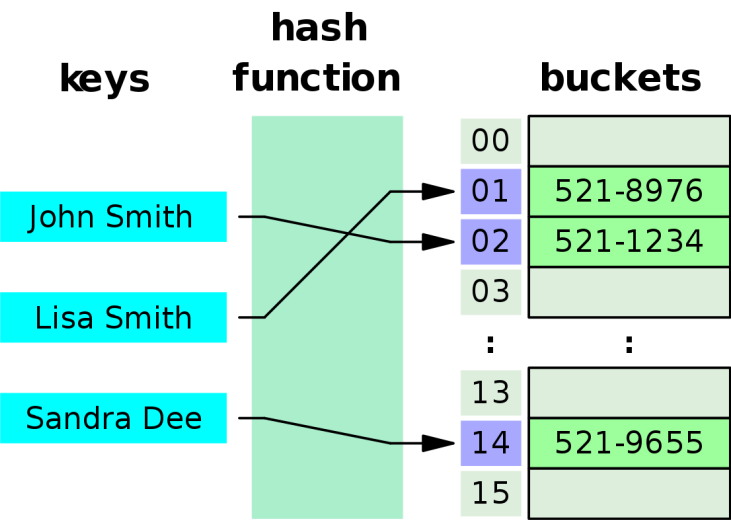
本次上机考察template的使用，实现一个hashmap。

需要注意的是本次实验的难度相比之前会大一些，但是我们的测试样例设置了基础分，也就是说只要你的文件能够编译，实现几个简单的功能，就能拿到及格的分。也就说，不必等到全部写完再提交。

hash map：这里我们用若干buckets来做hash，每个bucket均为nodes串联的链表。

通过计算公式`hash(key)%buckets.size()`来将key value对散列到bucket中。

注意这里的`hash(key)`得到的值可能远远大于bucket的个数，因此需要对其取余。



函数接口	实现
void clear()	清空hashmap
pair insert(const value_type& value)	插入key value对
bool erase(const K& key)	移除key
node_pair find_node(const K& key)	寻找node的前驱和node
void rehash(size_t new_bucket_count)	重新散列
==, !=, =	重载

注意：以下内容是为了方便理解和查看而列，函数和成员的具体描述请阅读注释，注释十分详尽。

template：K,M意思是key,value对，为了避免困惑用M表示value。H为hash函数。

```
template <typename K, typename M, typename H = std::hash<K>>
```

****别名：****使用**using**关键字和**typedef**关键字起别名，如**value_type**就可以用**std::pair<const K,M>**替代。

```
//这里value_type表示map里面的key value对
using value_type = std::pair<const K, M>;

//node_pair用来表示链表中相连的两个node，因为要链表往往要插入，需要pre、cur节点
//有了node_pair之后表示会容易很多
//hashmap.hpp文件中的find_node的返回值类型为node_pair，该函数的作用就是寻找到
pre,cur对
using node_pair = std::pair<typename HashMap::node*, typename
HashMap::node*>;
```

****std::pair：****你可以使用**make_pair(T1,T2)**或者**{T1,T2}**来创建。可以通过**pair.first**，**pair.second**分别获取对应的元素

命名空间：经常看到的**::**前半部分往往是命名空间，后半部分往往是类型。

成员：

```
private:
    //buckets链表节点
    struct node{
        value_type value;
        node* next;
    }
    //如node_pair处所述
    node_pair find_node(const K& key) const;
    //hashmap中节点/key的个数
    size_t _size;
    //usage: _hash_function(key);
    H _hash_function;
    //每个_buckets_array[i]均为一条链表
    std::vector<node*> _buckets_array;
```

Tips:

- 这几个函数容易实现：**size()**,**empty()**,**load_factor()**,**bucker_count()**
- **laod_factor()**的注释中Return value: **size_t - number of buckets**意思是返回值类型为**size_t**并不是说返回值是**size - num**。
- 先实现**find_node()**
- **insert()**,**contains()**,**erase()** 都可以调用**find_node()**，本质上都是对链表的操作。
- 对**[]**的重载：如果**key**不存在，需要你创建一个**key value**对，当然因为此时**value**并不知道，所以用**M()**代替就好。这么做的原因是实现**map[key]=value**的操作，**[]**重载之后，如果**key**不存在，会先创建**key**的空间，返回对**value**的引用，此时可以修改**map[key]**。
- **rehash()**的意思是此时**buckets**的个数发生了变化，需要重新散列，因此，去申请新的**vector<node*>**的空间，将**node**根据**key**进行散列即可。

- ==需要逐个比较

编译：

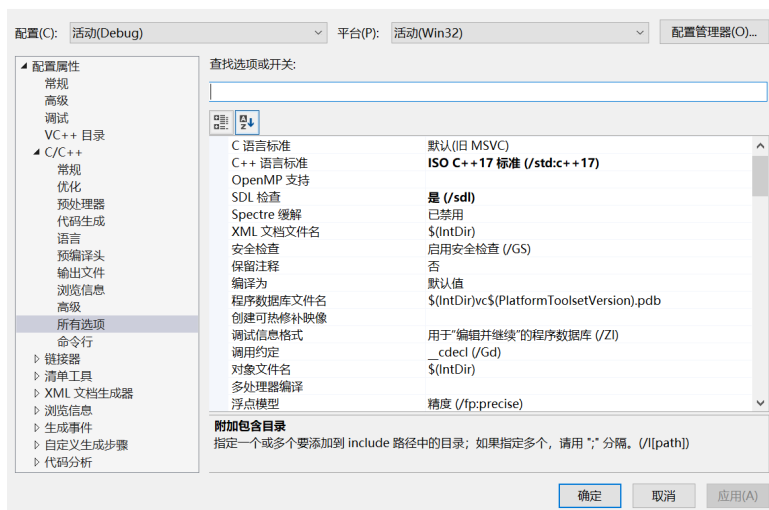
通常在写`template`时我们把（interface + implementation）放在.h文件中，但是这个文件注释太长了，所以我们把它分开。同时也是为了防止直接调用`hashmap`库。

我们强烈建议在Linux环境下编译运行。

但如果你一定要使用IDE（包括不限于VS, VScode, Clion）来编译运行，我们都建议将`hashmap.cpp`里的内容复制粘贴到`hashmap.hpp`中，否则可能会出现各种奇怪的问题。

提交的时候只用提交`hashmap.cpp`。

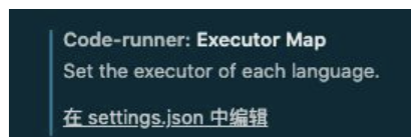
- 如果你使用的是linux系统，请使用 `g++ -std=c++17 test.cpp`进行编译。
- 如果你使用vs，请在右侧解决方案栏中右击`projectxx`，选择属性，选择C/C++一栏，选择所有选项，选择C++语言标准，选择C++17。



- 如果你使用VS，按以下操作，这里用C++17

c++11运行环境配置的解决方法：

- 点击Code Runner的齿轮⚙（或者直接右键它），打开扩展设置
- 找到 Code-runner: **Executor Map** 点击 在settings.json中编辑



- 在 "code-runner.executorMap" 中找到 "cpp" 的这一行
- 在 \$fileNameWithoutExt 的后面添加 `-std=c++11` 保存后即可食用

```
"cpp": "cd $dir && g++ $fileName -o $fileNameWithoutExt -std=c++11 && $dir$fileNameWithoutExt",
```

提交要求

- 提交源码文件：`hashmap.cpp` ,直接打包成zip格式的压缩包。不要添加其他任何目录
- 文件的编码格式只支持utf-8。
- 请严格按照给定的接口进行编码,否则无法调用测试用例。
- 提交的源码文件中不需要包含main函数,否则无法通过编译。