注意：本学期 CS101 PA 的所有 C++ 代码都采用 C++20 标准。如果您使用 GCC 或 Clang，您需要在编译时设置 `-std=c++20` 。如果您看到类似这样的报错

```
g++-9: error: unrecognized command line option ‘-std=c++20’; did you mean ‘-std=c++2a’?
```

说明您的编译器版本过低，请安装更高版本的编译器。目前 GCC 13 几乎已经将 C++20 除 modules 外的全部新特性实现完毕，我们推荐使用 GCC 13。

Hash Table

In this problem, you are required to implement a hash table to achieve fast key-value insertion and look-up.

The background of our problem is set to the DNS service. We need to find the IP address corresponding to the given domain name. This is a classical problem solved by a hash table.

NOTE: You can not use STL data structures such as `std::unordered_map`, `std::unordered_set`, `std::map`,`std::set` in this problem. **All of your hash tables should be designed by yourself.**

The files attached to this problem are shown as follows. We provide some tests to help you check your code.

```
.
└── attachments/
    ├── tests/
    |   ├── clear.cpp
    |   ├── erase.cpp
    |   ├── hash_function.cpp
    |   ├── insert.cpp
    |   └── probing.cpp
    ├── requirements.cpp
    ├── hash_table.hpp
    └── hash_types.hpp
```

Hash Function

In this part, you are required to implement a hash function for `class Website`.

`class Website` only has one private member `std::string name`, which represents the website's domain name. So our goal is to design a hash function for a string consisting of **ASCII characters**.

For our problem, you need to implement a specific hash function for string hash.

Suppose the string s has a length of n, we could write string s as $s_0s_1\dots s_{n-1}$. Our hash function $f(s)$ for string s is defined as follows:

$$f(s) = (s_0b^{n-1} + s_1b^{n-2} + \cdots + s_{n-2}b + s_{n-1}) \mod 2^{32}$$

Where s_i is the character's ASCII code in numerical forms and $b = 1009$ is a constant.

```cpp
template<>
struct std::hash<Website>{
    static const uint32_t base = 1009u;// 1009 is a prime number.
    uint32_t operator ()(const Website &S) const noexcept{

    }
};
```

Implement the hash function here. Suppose `y` is an instance of `class Website`, Call `std::hash<Website>{}(y)` to obtain the hash value of `y`. We will check whether you implement the hash function correctly.

(Hint: Integer overflow on a 32-bit unsigned integer is equal to modulus by 2^{32}.)

Hash Table

In this part, you should implement a hash table with key type `class Key` and value type `class T`. Your hash table should support **insert, find, erase and clear** operation.

First, your hash table has a constant size of `Mod=1000037`. An object `y` should be mapped into the position `std::hash<Key>{}(y) % Mod`.

Second, you should deal with hash collisions using **Linear Probing** or **Quadratic Probing**.

Third, implement the erase operation with **Lazy Erasing**.

```cpp
template<class Key = Website, class T = IPv4>     // template for key type class Key and value type class T
class HashTable{
private:
    enum entrystate{
        Unused, Used, Erased
    };
    const size_t Mod = 1000037u;   // 1000037 is a prime number
    std::vector<T> Table;          // main table stores value
    std::vector<Key> Stored_Key;   // used for storing original key
    std::vector<entrystate> State; // keeps the state of every table entry : {Unused, Used or Erased};
    std::vector<size_t> Used_pos;  // hint: used for function clear()

public:
    static const std::size_t npos = -1;
    HashTable() : Table(Mod),Stored_Key(Mod),State(Mod, entrystate::Unused) {}

    const T& operator [] (const Key &x) const
    {
        std::size_t pos = find(x);
        if (pos==npos) throw std::invalid_argument("Key does not exist");
        return Table[pos];
    }

    std::size_t Search(const Key &x) const;
    std::size_t find(const Key &x) const;
    void insert(const Key &x, T value);
    bool erase(const Key &x);
    void clear();
```

```
29
30 };
```

Requirements for member functions:

`std::size_t Search(const Key &x) const`:

If `Key x` already exists in the table, `std::size_t Search(const Key &x) const` should return the position where `Key x` is stored in the table.

If `Key x` does not exist in the table, `std::size_t Search(const Key &x) const` should return a position to insert `Key x` into this position.

`std::size_t find(const Key &x) const`:

If `Key x` already exists in the table, `std::size_t find(const Key &x) const` should return the position where `Key x` is stored in the table.

If `Key x` does not exist in the table, `std::size_t find(const Key &x) const` should return `npos`.

`void insert(const Key &x, const T &value)`:

Insert `Key x` into the table, and store its corresponding value `T value`.
If `Key x` already exists in the table, update its corresponding value to the newly inserted `const T &value`.

You needn't consider the situations where the hash table has no place to insert this `Key x`. We will guarantee the table cannot be full.

`bool erase(const Key &x):`

If `Key x` exists in the table, erase the table entry of `Key x` using **Lazy erasing**. After this operation, return `true`.

If `Key x` does not exist in the table, just return `false` and the table remains unchanged.

`void clear()`:

This operation would clear all table entries in the hash table, in other words, erase all elements in the hash table.

Your time complexity of this operation should be independent of the size of the hash table.

提交与评分

在 OJ 上提交 `hash_table.hpp` 内的全部内容。

本题的评分由 OJ 分数（60%）和线下 check （40%）两部分构成。线下 check 会在此次作业结束时间之后进行。

注：线下 check 也带有检查学术诚信的含义，当然这不是唯一的手段。如果被认定为抄袭， OJ 的分数也会作废，并且会有惩罚。**特别强调，抄袭来自 generative AI 的代码和抄袭网上的代码是同等处理的，我们建议您在写作业时关闭一切 generative AI 工具。**