C++ 小抄表

基本框架

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef long double ld;

const ll mod7 = 1e9 + 7;
const ll mod9 = 998244353;
const ll INF = 2 * 1024 * 1024 * 1023;
const char nl = '\n';

int main() {
    cin.tie(nullptr)->sync_with_stdio(false);

    int t;
    cin >> t;

    auto solve = [&]() {
        // TODO
        cout << nl; // 使用 endl 会导致 tie 失效
    };

    while (t--) {
        solve();
    }
    return 0;
}
```

类型转换

```cpp
string str = "";
char *ptr = nullptr;
char chs[100] = { 0 };
ptr = const_cast< char * >(str.c_str()); // string -> char*
double d = atof("0.23");                 // string -> double
int i = atoi("1021");                    // string -> int
long l = atol("303992");                 // string -> long
sprintf(chs, "%f", 2.3);                 // double -> char[]
strcpy(chs, ptr);                        // char* -> char[]
string str(chs);                         // char[] -> string
```

数组

```cpp
char arr[] = "hello";
arr[0] = 'X';              // 正确，修改数组元素
const char *ptr = "world"; // 应使用 const，因为字符串字面值不可改
```

动态数组 vector

```cpp
vector< int > arr(sz, val);                       // sz 和 val 可选
vector< vector< int > > dp(m, vector< int >(n)); // m * n 的数组
arr.empty(), arr.size() arr.push_back(val), arr.pop_back(), arr[i] = val
```

字符串 string

```
string str = "ABCDEFG";
str.size(), str.empty()
str.push_back('X'), str.pop_back(), str[i] = 'X'
str == s2, str.substr(start, len)
```

哈希表 unordered_map

```
unordered_map<string, int> map;
map.size(), map.empty(), map.count(key)
map.insert({key, val}), map.erase(key), map[key] = val
```

哈希集合 unordered_set

```
unordered_set<string> set;
set.size(), set.empty(), set.count(key)
set.insert(key), set.erase(key)
```

队列 queue

```
queue<string> q;
q.size(), q.empty(), q.push(val), q.pop(), q.front()
```

堆栈 stack

```
stack<string> s;
s.size(), s.empty(), s.push(val), s.pop(), s.top()
```

运算符重载

作为类成员时，重载二元运算符参数为另一个对象，一元运算符不需额外参数。作为全局函数时，重载二元运算符需要两个参数，一元运算符需要一个参数。示例：

```
// 作为成员函数
Complex Complex::operator+(const Complex &a) const {
  return Complex(real + a.real, img + a.img);
}
// 作为全局函数
Complex operator+(const Complex &a, int b) {
  return Complex(a.real + b, a.img);
}
// 类中声明全局函数为友元
friend Complex operator+<>(...);
```

工具函数

```
sort(v1.begin(), v1.end(), greater<int>());
sort(v1.begin(), v1.end(), [](int a, int b) { return a > b; });
reverse(v1.begin(), v1.end());
binary_search(v1.begin(), v1.end(), target);
```