字符串 split

```cpp
void split(const string &s, vector<string> &tokens, const string &delemiters
     = " ")
{
    string::size_type lastPos = s.find_first_not_of(delemiters, 0);
    string::size_type pos = s.find_first_of(delemiters, lastPos);
    while (string::npos != pos || string::npos != lastPos)
    {
        tokens.emplace_back(s.substr(lastPos, pos - lastPos));
        lastPos = s.find_first_not_of(delemiters, pos);
        pos = s.find_first_of(delemiters, lastPos);
    }
}
```

字符串与数字相互转换

```cpp
    // 数字转字符串
    string to_string (int val);
    string to_string (long val);
    string to_string (long long val);
    string to_string (unsigned val);
    string to_string (unsigned long val);
    string to_string (unsigned long long val);
    string to_string (float val);
    string to_string (double val);
    string to_string (long double val);

    // 字符串转数字
    stoi; stol; stoll; stof; stod; stold;

    // 浮点数输出
    #include<iomanip>
    cout << setiosflags(ios::fixed);   // 用一般的方式输出，而不是科学记数法
    cout << setprecision(2);           // 保留两位有效数字
    cout << setiosflags(ios::showpos); // 强制显示符号
```

优先队列

```cpp
struct Node
{
    int _x;
    int _y;
    Node(int x, int y) : _x(x), _y(y) {}
};

struct cmp1 // 大根堆排序规则
{
    bool operator()(const Node &lhs, const Node &rhs)
    {
        return lhs._x < rhs._x || (lhs._x == rhs._x && lhs._y < rhs._y);
    }
};

struct cmp2 // 小根堆排序规则
{
    bool operator()(const Node &lhs, const Node &rhs)
    {
        return lhs.__x > rhs.__x || (lhs._x == rhs._x && lhs._y > rhs._y);
    }
};

priority_queue<int, vector<int>, less<int>> q1;    // 大根堆
priority_queue<int, vector<int>, greater<int>> q2; // 小根堆
priority_queue<Node, vector<Node>, cmp1> q3;       // 结构体大根堆
priority_queue<Node, vector<Node>, cmp2> q4;       // 结构体小根堆
```

堆

```cpp
vector<int> vec = {1, 4, 2, 3, 5};
make_heap(vec.begin(), vec.end(), greater<int>()); // 小顶堆

// 插入元素
vec.emplace_back(20);
push_heap(vec.begin(), vec.end(), greater<int>());

// 删除元素
int val = vec[0]; // 堆顶元素
pop_heap(vec.begin(), vec.end(), greater<int>());
vec.pop_back();

// tag: 大顶堆只需要将greater<int>()改为less<int>(), 也可以自定义规则
```

STL sort

```cpp
struct Node
{
    int _x;
    int _y;
    Node(int x, int y) : _x(x), _y(y) {}
};

struct cmp1
{
    bool operator()(const Node &lhs, const Node &rhs)
    {
        return lhs._x < rhs._x || (lhs._x == rhs._x && lhs._y < rhs._y);
    }
};

struct cmp2
{
    bool operator()(const Node &lhs, const Node &rhs)
    {
        return lhs._x > rhs._x || (lhs._x == rhs._x && lhs._y > rhs._y);
    }
};

vector<int> vec = {1, 4, 2, 3, 5};
sort(vec.begin(), vec.end(), less<int>());    // 升序排序
sort(vec.begin(), vec.end(), greater<int>()); // 降序排序

vector<Node> vecNode = {{1, 2}, {1, 1}, {2, 3}, {2, 2}};
sort(vecNode.begin(), vecNode.end(), cmp1()); // 升序排序
sort(vecNode.begin(), vecNode.end(), cmp2()); // 降序排序

// 升序排序
sort(vecNode.begin(), vecNode.end(), [&](const Node &lhs, const Node &rhs)
    { return lhs._x < rhs._x || (lhs._x == rhs._x && lhs._y < rhs._y); });

// 降序排序
sort(vecNode.begin(), vecNode.end(), [&](const Node &lhs, const Node &rhs)
    { return lhs._x > rhs._x || (lhs._x == rhs._x && lhs._y > rhs._y); });
```

下一个排列

```cpp
void nextPermutation(vector<int>& nums) {
    int i = nums.size() - 2;
    while (i >= 0 && nums[i] >= nums[i + 1]) {
        i--;
    }
    if (i >= 0) {
        int j = nums.size() - 1;
        while (j >= 0 && nums[i] >= nums[j]) {
            j--;
        }
        swap(nums[i], nums[j]);
    }
    reverse(nums.begin() + i + 1, nums.end());
}
```

取整与四舍五入

```cpp
floor; // 向下取整
ceil;  // 向上取整
round; // 仅仅对小数点后一位四舍五入

// 如果要保留有效小数数位，可以先乘后除
double x = 1.5684;
double y = round(x * 100) / 100;  // 保留两位有效数字
```

cctype

```cpp
isalnum(); // 判断一个字符是不是alphanumeric，即大小写英文字母或是数字
isalpha(); // 判断一个字符是不是alphabetic，即英文字母
isdigit(); // 判断一个字符是不是数字
tolower(); // 将大写转换为小写
toupper(); // 将小写转换为大写
```