lvxiaoxin的C/C++代码清单

Table of Content

0.0 Basic Header

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
#include <stack>
#include <queue>
#include <set>
#include <map>
#include <string>
#include <cmath>
#include <cctype>
#include <cstdlib>
#include <ctime>
#include <iomanip>

using namespace std;

const int MOD = 1e9 + 7;
const int MAXN = 1e5 + 3;

int main()
{
    //freopen("in.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);


    return 0;
}
```

1.0 I/O

小数输出 C

```
// x = 6.34;
printf("%f", x); //输出6.34

printf("%2f", x); //输出6.34（位数足够，原样输出）

printf("%5f", x); //输出 空格空格6.34（左端补）

printf("%.1f". x); //输出6.3

printf("%o", a); //八进制输出

printf("%x", a); //十六进制输出

printf("%u", a); //作为无符号数输出
```

C++

```
#include <iomanip> //IO各种函数的头文件

// a = 6.34
cout << setiosflags(ios::fixed) << setprecision(1) << a << endl;
//输出 6.3

cout << setprecision(1) << a << endl;
//输出6
```

字符串读取

```
cin 读取并忽略开头的所有空白字符（空格、回车、制表），读取字符直至遇到空白字符，读取终止

getline(cin, str)，两个参数，第一个是输入流对象，第二个是字符串名，不能忽略行开头的换行符，只要getline遇到换行符，就会读取字符并返回，能读取除换行符之外的所有空白符

例：
    string str1, str2

    cin >> str1;
    cin >> str2;

    cout << str1 << endl;
    cout << str2 << endl;

    输入：

    hello world

    输出：
```

```
    hello
    world

    输入：

    hello
    world

    输出：

    hello
    world

    getline(cin, str1);
    getline(cin, str2);

    cout << str1 << endl;
    cout << str2 << endl;

    输入：

    hello world

    输出：
    不会有输出，因为hello world\n 全部被读取到了str1，str2还没读取

    输入：

    hello
    world

    输出：

    hello
    world
```

例：

```
  int a;
  string s;
  cin >> s;
  cin >> a;

  cout << s;
  cout << a << endl;

  输入：
  e
```

```
  2

  输出:
  e2
  （说明回车符其实是丢掉的）
```

各种输入注意

```
scanf
%d 格式输入：默认分隔符是所有的空白字符（空格、回车、制表）
%c 无分隔符，可能会受到之前输入的影响
%s 默认分割符是所有的空白字符（空格、回车、制表），输入后自动加入'\0'

例
  int a;
  char c;
  scanf("%d", &a);
  scanf("%c", &c);
  printf("%d\n%c\n", a, c);

  输入:
  1\n

  输出:
  1

  (a = 1, c = '\n')

  如想继续向c输入字符，需要中间加个getchar()承接回车符
  scanf("%d", &a);
  getchar();
  scanf("%c", &c);

例
  char a[10];
  scanf("%s", a);
  printf("%s\n", a);

  输入:
  string string2

  输出:
  string

  （空格结束了输入的继续）
```

2.0 STL

vector

```
vector<int> p;
int a = 5;

p.begin(); // firset element point
p.end(); // last element point

p.push_back(a); // Add element at the end
p.pop_back(); // Delete last element

vector<int>::iterator it;
it = p.begin();
p.insert(it, a);

p.erase(p.begin(), p.end()); // Erase the element

p.clear(); //清空p

for(int i=0; i<10; ++i) p.push_back(i);
cout << p.size(); //输出10
cout << p.capacity(); //输出16(意思是第一次分配了16，当超过16会再分配)

for(vector<int>::iterator it = p.begin(); it != p.end(); ++it)
{
    cout << *it;
}

sort(p.begin(), p.end()); //从小到大 -- 头文件 #include <algorithm>
reverse(p.begin(), p.end()); //反转p --头文件 #include <algorithm>


如果是自定义数据结构类型:
struct Node
{
    int x;
    int y;
};

vector<Node> p;

则应该
sort(p.begin(), p.end(), cmp);

其中cmp为自定义的比较函数：
```

```
//相当于定义了小于号
struct cmp
{
    bool operator()(Node a, Node b)
    {
        return a.x > b.x;
    }
};
```

queue

```
//普通队列
queue<int> myQue;

//队列清零操作
while(!myQue.empty())
{
    myQue.pop();
}
int a = 9;
myQue.push(a);//入队
myQue.pop();//出队
myQue.size(); //队列元素个数
myQue.front(); //访问最前的元素，不弹出
myQue.back(); //访问最后的元素，不弹出


//优先队列 --basic
priority_queue<int> myQue; //默认为最大堆
priority_queue<int, vector<int>, greater<int> > myQue;//此时为最小堆

//清零操作
while(!myQue.empty())
{
    myQue.pop();
}
myQue.size(); //队列元素个数
myQue.top(); //访问最顶端（最大/小)元素，不弹出
myQue.push(a); //入队操作
myQue.pop(); //出队操作

如果是自定义结构：

如果：
struct Node
{
    int value;
    int id;
```

```
    Node(int v, int i): value(v), id(i) {}
    friend bool operator < (const struct Node &a, const struct Node &b)
    {
        return a.value < b.value;
    }
};

priority_queue<Node> myQue; //此时myQue是最大堆
```

如果:

```
struct Node
{
    int value;
    int id;
    Node(int v, int i): value(v), id(i) {}
    friend bool operator > (const struct Node &a, const struct Node &b)
    {
        return a.value > b.value;
    }
};

priority_queue<Node, vector<Node>, greater<Node> > myQue; //此时myQue是最小堆
```

方法二:

声明比较函数cmp

如果:

```
struct node{
  int idx;
  int key;
  node(int a=0, int b=0):idx(a), key(b){}
};

struct cmp{
  bool operator()(node a, node b){
    return a.key > b.key;
  }
};

priority_queue<node, vector<node>, cmp> q; //此时q是最小堆
```

如果:

```
struct node{
  int idx;
  int key;
  node(int a=0, int b=0):idx(a), key(b){}
```

```
};

struct cmp{
  bool operator()(node a, node b){
    return a.key < b.key;
  }
};

priority_queue<node, vector<node>, cmp> q; //此时q是最大堆
```

cctype

```
char c;

//注意：一下针对的都是char类型
isalnum(c) //检查是否是数字或者字母，是-1，否-0
isdigit(c) //检查是否是十进制数字字符，'7'而不是7，是-1，否-0
isxdigit(c) //检查是否是十六进制数字字符， 是-1，否-0
isalpha(c) //检查是否是字母，是-1，否-0
isblank(c) //检查是否是空格' ',是-1，否-0
islower(c) //检查是否小写字母，是-1，否-0

tolower(c);返回小写的c字符
toupper(c);返回大写的c字符
```

3.0 一些语法模板

重载运算符：

例：

```
struct Node
{
    int x;
    int y;

    Node(int a, int b)
    {
        x = a;
        y = b;
    }
    friend ostream &operator << (ostream &out, Node a);

    Node& operator = (Node& a)
    {
        x = a.x;
        y = a.y;
        return *this;
    }

    Node& operator + (Node& a)
    {
        x += a.x;
        y += a.y;
        return *this;
    }

    bool operator < (const Node& a) const
    {
        return this->x < a.x;
    }
    // 从而可以直接用priority_queue<Node> 获得最大堆
};

friend ostream &operator << (ostream &out, Node a)
{
    out << a.x << " " << a.y << endl;
    return out;
}
```