C++ 的特性

结构体的构造函数

```cpp
#include<iostream>
using namespace std;

//构造函数的写法1
struct st{
    int a, b;
    st(int _a, int _b){
        a = _a;
        b = _b;
    }
};

//构造函数的写法2
struct st_{
    int a, b;
    st_(int a_, int b_):a(a_), b(b_){}
    //st_(int a_ = 0, int b_):a(a_), b(b_){} 错误
}

int main(){
    st newSt = st(1, 2);//调用结构体的构造函数
    st newSt_(1, 2);
    //st newSt1;//编译错误：函数的缺省值
    cout << newSt.a << newSt.b;
    return 0;
}

/*
构造函数的写法1
结构体名（参数列表）{
    赋值语句
}

构造函数的写法2
结构体名（参数列表）：结构体成员（参数），结构体成员（参数）{}
*/
```

- 构造函数的**返回值是一个结构体**
- 构造函数与结构体**重名**
- 结构体的构造函数在创建结构体是**自动调用**
- 写了构造函数后，原本的无参构造函数会消失，声明是必须使用构造函数

常见输入输出

- `getline()` 读入包含空格的整行数据
    - `getline(cin, 字符串变量名字);`
    - `getline(字符串变量名字, 读入字符上线);`
    - `getline()` 前如有其他操作，必须加入 `getchar()` 来吞掉换行符 `/n`
- 循环读入直到文件结束（判断EOF结束循环）

```
int a, b;
while(cin >> a >> b){

}
```

动态内存分配

```
#incldue<iostream>
using namespace std;

struct node{
    int value;
    node* next;
    node(int _value){
        value = _value;
    }
};
node* head;

int main(){
    head = new node(1);//用new动态开辟head指向node的空间//node变量的地址赋值在head
    head->next = new node(2);//对head内部的next进行操作（用new动态开辟head的next指向node的空间）
    node* second = head->next;//声明一个node型的指针名为second

    cout << head->value << second->value << head->next->value;
    return 0;
}
```

- 动态内存的分配一般用于新建结构体上
- 可以不用释放内存

函数的重载

```
#include<iostream>
using namespace std;

int sum(int a){
    cout<<"���õ��� int sum(int a)"<<endl;
    return a;
}
double sum(double a){
    cout<<"���õ��� double sum(double a)"<<endl;
```

```
    return a;
}
int sum(int a,int b){
    cout<<"���õ��� sum(int a,int b)"<<endl;
    return a+b;
}
int main()
{
    int a = 1;
    int b = 1;
    double c = 1;

    cout<<sum(a)<<endl;
    cout<<sum(c)<<endl;
    cout<<sum(a,b)<<endl;
    return 0;
}
```

- C++中区别不同函数不仅靠函数名，对于不同的函数至少满足一下一种情况：
    - 函数名不同
    - 形参类型或数量不同
    - 有缺省值不引起**调用时**的歧义
        - ```
          int sum(int a){}
          double sum(int a){}
          //返回值不能区别函数
          ```

函数的缺省值

在C++的形参列表中可以允许缺省值的存在，在没有传入对应变量时，使用的就是缺省值

但使用时要遵循：

参数列表从左往右，一旦出现缺省值参数后，后面的参数也必须带有缺省值

```
int sum(int a,int b = 0){
    cout << a << b;
}
int main(){
    sum(1);
    return 0;
}
```

运算符的重载

写法一：

```
#include<iostream>
using namespace std;

struct str{
```
```

```
    int A_score, B_score;
    str(int _A, int _B):A_score(_A), B_score(_B){}
    bool operator < (const str b){
        return this->A_score < b.A_score;
    }
};
/*
bool operator 运算符(const 结构体名称 b){
    return 什么是时候这个运算符对结构体成立;
}
*/
int main(){
    str a(2, 6), b(7, 1);
    cout << (a < b);
    return 0;
}
```

写法二:

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

struct node{
    int x, y;
};

bool operator < (node a, node b){
    return a.x < b.x
}

int main(){
    priority_queue<node> pq;
    pq.push({8,3});
    pq.push({4,6});
    pq.push({5,7});
    while(!pq.empty()){
        cout << pq.top().x << pq.top().y;
        pq.pop();
    }
    return 0;
}
```

优先队列中如果是结构体这种无法直接比较的元素，必须要重载运算符<，使其可以比较。