1. STL

STL是StandardTemplateLibrary的简称，标准模板库

从根本上说，STL时一些容器的集合，这些容器有 list，set，map，vector等，

STl也是算法和其他一些组建的结合

STL的目的时标准化组件，这样就可以不用重新开发，可以使用现成的组件

在C++标准中，STL被组织为下面的13个头文件：<algorithm>、<deque>、<functional>、<iterator>、<array>、<vector>、<list>、list>、<map>、map>、<memory>、<numeric>、<queue>、<set>、<unordered\_set>、<stack>和<utility>。

容器(containers)

[迭代器](https://baike.sogou.com/v8083528.htm" \t "https://baike.sogou.com/_blank)(iterators)

空间配置器(allocator)

STL的组成：

配接器(adaptors)

算法(algorithms)

[仿函数](https://baike.sogou.com/v11037539.htm" \t "https://baike.sogou.com/_blank)(functors)

本文只涉及其中的 容器、迭代器、算法。

****一、容器（containers）****

1、容器的定义

       在数据存储上，有一种对象类型，它可以持有其它对象或指向其它对像的指针，这种对象类型就叫做容器。很简单，容器就是保存其它对象的对象，当然这是一个朴素的理解，这种“对象”还包含了一系列处理“其它对象”的方法。

2、容器的种类

二、运算符重载

为什么要用运算符重载?

答：在普通运算符无法满足需求时

例如：当一类的输出有很多元素时，总不能建一个类就一个一个输出吧。

两个类相加时。

1. Vector operator + (Vector lhs, Vector rhs)
3. {
5. Vector result = **new** Vector(lhs);
7. result.x += rhs.x;
9. result.y += rhs.y;
11. result.z += rhs.z;
13. **return** result;
15. }
17. }
19. }

运算符重载与深复制（深拷贝）与浅复制（浅拷贝）一样，在没有深复制（深拷贝）的情况下。

C++中大部分你的内置运算符都可以重载（自定义）与运算符

重载的运算符是带有特殊名称的函数，函数名是由关键字 operator 和其后要重载的运算符符号构成的。与其他函数一样，重载运算符有一个返回类型和一个参数列表。

1. Box operator+(**const** Box&);

声明加法运算符用于把两个 Box 对象相加，返回最终的 Box 对象。大多数的重载运算符可被定义为普通的非成员函数或者被定义为类成员函数。如果我们定义上面的函数为类的非成员函数，那么我们需要为每次操作传递两个参数，如下所示：

1. Box operator+(**const** Box&, **const** Box&);

以下为实例：

1. #include <iostream>
2. **using** **namespace** std;
4. **class** Box
5. {
6. **public**:
8. **double** getVolume(**void**)
9. {
10. **return** length \* breadth \* height;
11. }
12. **void** setLength( **double** len )
13. {
14. length = len;
15. }
17. **void** setBreadth( **double** bre )
18. {
19. breadth = bre;
20. }
22. **void** setHeight( **double** hei )
23. {
24. height = hei;
25. }
27. Box operator+(**const** Box& b)   // 重载 + 运算符，用于把两个 Box 对象相加
28. {
29. Box box;
30. box.length = **this**->length + b.length;
31. box.breadth = **this**->breadth + b.breadth;
32. box.height = **this**->height + b.height;
33. **return** box;
34. }
35. **private**:
36. **double** length;      // 长度
37. **double** breadth;     // 宽度
38. **double** height;      // 高度
39. };
40. // 程序的主函数
41. **int** main( )
42. {
43. Box Box1;                // 声明 Box1，类型为 Box
44. Box Box2;                // 声明 Box2，类型为 Box
45. Box Box3;                // 声明 Box3，类型为 Box
46. **double** volume = 0.0;     // 把体积存储在该变量中
48. // Box1 详述
49. Box1.setLength(6.0);
50. Box1.setBreadth(7.0);
51. Box1.setHeight(5.0);
53. // Box2 详述
54. Box2.setLength(12.0);
55. Box2.setBreadth(13.0);
56. Box2.setHeight(10.0);
58. // Box1 的体积
59. volume = Box1.getVolume();
60. cout << "Volume of Box1 : " << volume <<endl;
62. // Box2 的体积
63. volume = Box2.getVolume();
64. cout << "Volume of Box2 : " << volume <<endl;
66. // 把两个对象相加，得到 Box3
67. Box3 = Box1 + Box2;
69. // Box3 的体积
70. volume = Box3.getVolume();
71. cout << "Volume of Box3 : " << volume <<endl;
73. **return** 0;
74. };

运算结果： Volume of Box1 : 210

Volume of Box2 : 1560

Volume of Box3 : 5400

|  |  |
| --- | --- |
| 双目算术运算符 | + (加)，-(减)，\*(乘)，/(除)，% (取模) |
| 关系运算符 | ==(等于)，!= (不等于)，< (小于)，> (大于>，<=(小于等于)，>=(大于等于) |
| 逻辑运算符 | ||(逻辑或)，&&(逻辑与)，!(逻辑非) |
| 单目运算符 | + (正)，-(负)，\*(指针)，&(取地址) |
| 自增自减运算符 | ++(自增)，--(自减) |
| 位运算符 | | (按位或)，& (按位与)，~(按位取反)，^(按位异或),，<< (左移)，>>(右移) |
| 赋值运算符 | =, +=, -=, \*=, /= , % = , &=, |=, ^=, <<=, >>= |
| 空间申请与释放 | new, delete, new[ ] , delete[] |
| 其他运算符 | **()**(函数调用)，**->**(成员访问)，**,**(逗号)，**[]**(下标) |

下面是不可重载的运算符列表：

* **.**：成员访问运算符
* **.\***, **->\***：成员指针访问运算符
* **::**：域运算符
* **sizeof**：长度运算符
* **?:**：条件运算符
* **#**： 预处理符号

一元重载运算符：

一元运算符只对一个操作数进行操作，下面是一元运算符的实例：

* [递增运算符（ ++ ）和递减运算符（ -- ）](http://www.runoob.com/cplusplus/increment-decrement-operators-overloading.html)
* 一元减运算符，即负号（ - ）
* 逻辑非运算符（ ! ）

一元运算符通常出现在它们所操作的对象的左边，比如 !obj、-obj 和 ++obj，但有时它们也可以作为后缀，比如 obj++ 或 obj--。

前置++：

1. //friend function
2. A& operator++(A& a)
3. {
4. a.dat += 20;
5. **return** a;
6. }
7. //member function
8. A& operator++( )
9. {
10. **this**->dat += 20;
11. **return** \***this**;
12. }
14. }

后置++：

1. 复制代码
2. //friend function
3. A operator++(A&a, **int**)
4. {
5. A c = a;
6. a.dat += 10;
7. **return** c;
8. }
10. //member function
11. A operator++( **int**)
12. {
13. A c = a;
14. **this**->dat += 10;
15. **return** c;
16. }
17. 复制代码

后置++是先使用在++。

所以需要加入一个变量为了区分重载函数，不能以返回值作为判断标准，所以加入了一个没用的占位符int作为判断标准

二元运算符：

二元运算符需要两个参数，下面是二元运算符的实例。我们平常使用的加运算符（ + ）、减运算符（ - ）、乘运算符（ \* ）和除运算符（ / ）都属于二元运算符。就像加(+)运算符。

1. #include <iostream>
2. **using** **namespace** std;
4. **class** Box
5. {
6. **double** length;      // 长度
7. **double** breadth;     // 宽度
8. **double** height;      // 高度
9. **public**:
11. **double** getVolume(**void**)
12. {
13. **return** length \* breadth \* height;
14. }
15. **void** setLength( **double** len )
16. {
17. length = len;
18. }
20. **void** setBreadth( **double** bre )
21. {
22. breadth = bre;
23. }
25. **void** setHeight( **double** hei )
26. {
27. height = hei;
28. }
29. // 重载 + 运算符，用于把两个 Box 对象相加
30. Box operator+(**const** Box& b)
31. {
32. Box box;
33. box.length = **this**->length + b.length;
34. box.breadth = **this**->breadth + b.breadth;
35. box.height = **this**->height + b.height;
36. **return** box;
37. }
38. };
39. // 程序的主函数
40. **int** main( )
41. {
42. Box Box1;                // 声明 Box1，类型为 Box
43. Box Box2;                // 声明 Box2，类型为 Box
44. Box Box3;                // 声明 Box3，类型为 Box
45. **double** volume = 0.0;     // 把体积存储在该变量中
47. // Box1 详述
48. Box1.setLength(6.0);
49. Box1.setBreadth(7.0);
50. Box1.setHeight(5.0);
52. // Box2 详述
53. Box2.setLength(12.0);
54. Box2.setBreadth(13.0);
55. Box2.setHeight(10.0);
57. // Box1 的体积
58. volume = Box1.getVolume();
59. cout << "Volume of Box1 : " << volume <<endl;
61. // Box2 的体积
62. volume = Box2.getVolume();
63. cout << "Volume of Box2 : " << volume <<endl;
65. // 把两个对象相加，得到 Box3
66. Box3 = Box1 + Box2;
68. // Box3 的体积
69. volume = Box3.getVolume();
70. cout << "Volume of Box3 : " << volume <<endl;
72. **return** 0;
73. }
74. 当上面的代码被编译和执行时，它会产生下列结果：
75. Volume of Box1 : 210
76. Volume of Box2 : 1560
77. Volume of Box3 : 5400

为什么使用友元函数重载？

答：当 2 个对象相加时是没有顺序要求的，但要重载 **+** 让其与一个数字相加则有顺序要求，可以通过加一个友元函数使另一个顺序的输入合法。、

例子：

1. #include<iostream>
2. **using** **namespace** std;
3. **class** A
4. {
5. **private**:
6. **int** a;
7. **public**:
8. A();
9. A(**int** n);
10. A operator+(**const** A & obj);
11. A operator+(**const** **int** b);
12. **friend** A operator+(**const** **int** b, A obj);
13. **void** display();
14. } ;
15. A::A()
16. {
17. a=0;
18. }
19. A::A(**int** n)//构造函数
20. {
21. a=n;
22. }
23. A A::operator +(**const** A& obj)//重载+号用于 对象相加
24. {
25. **return** **this**->a+obj.a;
26. }
27. A A::operator+(**const** **int** b)//重载+号用于  对象与数相加
28. {
29. **return** A(a+b);
30. }
31. A operator+(**const** **int** b,  A obj)
32. {
33. **return** obj+b;//友元函数调用第二个重载+的成员函数  相当于 obj.operator+(b);
34. }
35. **void** A::display()
36. {
37. cout<<a<<endl;
38. }
39. **int** main ()
40. {
41. A a1(1);
42. A a2(2);
43. A a3,a4,a5;
44. a1.display();
45. a2.display();
46. **int** m=1;
47. a3=a1+a2;//可以交换顺序，相当月a3=a1.operator+(a2);
48. a3.display();
49. a4=a1+m;//因为加了个友元函数所以也可以交换顺序了。
50. a4.display();
51. a5=m+a1;
52. a5.display();
53. }
54. 输出结果：
55. 1
56. 2
57. 3
58. 2
59. 2

（）重载

例子：

1. #include <iostream>
2. **using** **namespace** std;
4. **class** Distance
5. {
6. **private**:
7. **int** feet;             // 0 到无穷
8. **int** inches;           // 0 到 12
9. **public**:
10. // 所需的构造函数
11. Distance(){
12. feet = 0;
13. inches = 0;
14. }
15. Distance(**int** f, **int** i){
16. feet = f;
17. inches = i;
18. }
19. // 重载函数调用运算符
20. Distance operator()(**int** a, **int** b, **int** c)
21. {
22. Distance D;
23. // 进行随机计算
24. D.feet = a + c + 10;
25. D.inches = b + c + 100 ;
26. **return** D;
27. }
28. // 显示距离的方法
29. **void** displayDistance()
30. {
31. cout << "F: " << feet <<  " I:" <<  inches << endl;
32. }
34. };
35. **int** main()
36. {
37. Distance D1(11, 10), D2;
39. cout << "First Distance : ";
40. D1.displayDistance();
42. D2 = D1(10, 10, 10); // invoke operator()
43. cout << "Second Distance :";
44. D2.displayDistance();
46. **return** 0;
47. }
48. 当上面的代码被编译和执行时，它会产生下列结果：
49. First Distance : F: 11 I:10
50. Second Distance :F: 30 I:120

输入\输出重载运算符

C++ 能够使用流提取运算符 >> 和流插入运算符 << 来输入和输出内置的数据类型。您可以重载流提取运算符和流插入运算符来操作对象等用户自定义的数据类型。

在这里，有一点很重要，我们需要把运算符重载函数声明为类的友元函数，这样我们就能不用创建对象而直接调用函数。

下面的实例演示了如何重载提取运算符 >> 和插入运算符 <<。

例子：

1. #include <iostream>
2. **using** **namespace** std;
4. **class** Distance
5. {
6. **private**:
7. **int** feet;             // 0 到无穷
8. **int** inches;           // 0 到 12
9. **public**:
10. // 所需的构造函数
11. Distance(){
12. feet = 0;
13. inches = 0;
14. }
15. Distance(**int** f, **int** i){
16. feet = f;
17. inches = i;
18. }
19. **friend** ostream &operator<<( ostream &output,
20. **const** Distance &D )
21. {
22. output << "F : " << D.feet << " I : " << D.inches;
23. **return** output;
24. }
26. **friend** istream &operator>>( istream  &input, Distance &D )
27. {
28. input >> D.feet >> D.inches;
29. **return** input;
30. }
31. };
32. **int** main()
33. {
34. Distance D1(11, 10), D2(5, 11), D3;
36. cout << "Enter the value of object : " << endl;
37. cin >> D3;
38. cout << "First Distance : " << D1 << endl;
39. cout << "Second Distance :" << D2 << endl;
40. cout << "Third Distance :" << D3 << endl;

43. **return** 0;
44. }
45. 当上面的代码被编译和执行时，它会产生下列结果：
46. $./a.out
47. Enter the value of object :
48. 70
49. 10
50. First Distance : F : 11 I : 10
51. Second Distance :F : 5 I : 11
52. Third Distance :F : 70 I : 10

类成员访问运算符（ -> ）可以被重载，但它较为麻烦。它被定义用于为一个类赋予"指针"行为。运算符 -> 必须是一个成员函数。如果使用了 -> 运算符，返回类型必须是指针或者是类的对象。

运算符 -> 通常与指针引用运算符 \* 结合使用，用于实现"智能指针"的功能。这些指针是行为与正常指针相似的对象，唯一不同的是，当您通过指针访问对象时，它们会执行其他的任务。比如，当指针销毁时，或者当指针指向另一个对象时，会自动删除对象。

间接引用运算符 -> 可被定义为一个一元后缀运算符。也就是说，给出一个类：

class Ptr{

//...

X \* operator->();};

使用：void f(Ptr p ){

p->m = 10 ; // (p.operator->())->m = 10} //使用方式与指针十分相似。