有些算法可以接受用户自定义的辅助性函数，借以提高弹性和能力。

例：程序stl\_test36

int square(int value)

{

return value \* value;

}

template<typename T>

inline void PrintElements(const T& coll, const *string*& optstr = "")

{

*cout* << optstr.*c\_str*();

for (const auto& elem : coll)

{

*cout* << elem << ' ';

}

*cout* << *endl*;

}

int *main*(int argc, char\* argv[])

{

*set*<int> coll1;

*vector*<int> coll2;

for (int i = 1; i <= 9; ++i)

{

coll1.*insert*(i);

}

// 自定义的square作为算法的实参

*transform*(coll1.*cbegin*(), coll1.*cend*(), *back\_inserter*(coll2), square);

PrintElements(coll1, "initialized: ");

PrintElements(coll2, "squared: ");

return 0;

}

输出为：

initialized: 1 2 3 4 5 6 7 8 9

squared: 1 4 9 16 25 36 49 64 81

判断式辅助函数：Predicate，STL要求，面对相同的值，Predicate必须得到相同的结果。参考C++标准库 第2版.pdf 10.1.4章节。

单参判断式：

例：程序stl\_test37

// 判断是否为质数

bool IsPrime(int number)

{

number = *abs*(number);

if (number == 0 || number == 1)

{

return false;

}

int divisor;

for (divisor = number / 2; number % divisor != 0; --divisor)

{

}

return divisor == 1;

}

int *main*(int argc, char\* argv[])

{

*list*<int> coll;

for (int i = 24; i <= 30; ++i)

{

coll.*push\_back*(i);

}

// 查找第1个质数

auto pos = *find\_if*(coll.*cbegin*(), coll.*cend*(), IsPrime);

if (pos != coll.*end*())

{

*cout* << \*pos << " is first prime number found" << *endl*;

}

else

{

*cout* << "no prime number found" << *endl*;

}

return 0;

}

双参判断式：

例：程序stl\_test38

struct Person{

public:

Person(*string* first\_name, *string* last\_name)

{

first\_name\_ = first\_name;

last\_name\_ = last\_name;

}

public:

*string* first\_name\_;

*string* last\_name\_;

};

// 双参判断式

bool PersonSortCriterion(const Person& p1, const Person& p2)

{

return p1.last\_name\_.*compare*(p2.last\_name\_) < 0

|| (p1.last\_name\_.*compare*(p2.last\_name\_) == 0

&& p1.first\_name\_.*compare*(p2.first\_name\_) < 0);

}

int *main*(int argc, char\* argv[])

{

*deque*<Person> coll{ { "Jam", "Tom" }, { "Jo", "Robert" }, {"Trump", "Bill"} };

*sort*(coll.*begin*(), coll.*end*(), PersonSortCriterion);

for (const auto& elem : coll)

{

*cout* << "first name " << elem.first\_name\_.*c\_str*()

<< " last name " << elem.last\_name\_.*c\_str*() << *endl*;

}

return 0;

}

输出为：

first name Trump last name Bill

first name Jo last name Robert

first name Jam last name Tom

使用Lambda作为算法实参：

Lambda：“在表达式或语句内指明函数行为”的定义式。

使用Lambda作为算法实参，更简单直观，可以避免丑陋的代码。

例：程序stl\_test39

*deque*<int> coll = { 1, 3, 19, 5, 13, 7, 11, 2, 17 };

int x = 5;

int y = 12;

// Lambda表达式作为算法实参

// [=]传值方式

auto pos = *find\_if*(coll.*cbegin*(), coll.*cend*(), [=](int i) {return i > x && i < y; });

if (pos != coll.*end*())

{

*cout* << "first elem > 5 and < 12: " << \*pos << *endl*;

}