C++中的key-value，pair，如果希望将两个value视为一个单元，都可以使用pair。

pair是一个struct类型，所有成员都是public访问类型。

例：程序stl\_test13

*std*::*pair*<int, float> p(42, 3.14f);

*cout* << "first: " << p.*first* << " second: " << p.*second* << *endl*;

*cout* << "first: " << *get*<0>(p) << " second: " << *get*<1>(p) << *endl*;

输出为：

first: 42 second: 3.14

first: 42 second: 3.14

pair的构造函数

例：程序stl\_test13

class Foo {

public:

Foo(*tuple*<int, float>)

{

*cout* << "Foo::Foo(tuple)" << *endl*;

}

// 变长参数模板

template<typename... Args>

Foo(Args... args)

{

*cout* << "Foo::Foo(args...)" << *endl*;

// 输出变长参数

write(args...);

}

private:

// 这里利用模板的推导功能，每次取出第一个参数t

// 将其他的参数继续作为变长参数args

// 类似一个递归

template<typename T, typename ...Args>

void write(const T& t, Args... args)

{

*cout* << t << ",";

write(args...);

}

// 递归的出口

// 变长参数的最后一个参数

template<typename T>

void write(const T& t)

{

*cout* << t << *endl*;

}

};

*tuple*<int, float> t(1, 2.2f);

*tuple*<int, *string*> t1(4, "a");

*pair*<int, Foo> p1(42, t);

// piecewise\_construct作为第一实参，分段构造

// 传入两个tuple，45强制转换为tuple，为p2的first元素

// p2的second元素是一个Foo对象，这里会调用 template<typename... Args>

// Foo(Args... args)构造函数，接受tuple元素而不是tuple整体

*pair*<int, Foo> p2(*piecewise\_construct*, *make\_tuple*(45), t);

输出为：

Foo::Foo(tuple)

Foo::Foo(args...)

1,2.2

pair之间的比较：只有两个元素完全相同，pair才相等

first元素的优先级高，只有first元素相等，才会去比较second元素。

tuple（元组，不定数的值组）

例：程序stl\_test14

// 两个>之间不需要再加空格

*tuple*<*string*, int, int, *complex*<double>> t;

*tuple*<int, float, *string*> t1(41, 6.3f, "nico");

*cout* << "first " << *get*<0>(t1) << " second " << *get*<1>(t1) << " third " << *get*<2>(t1) << *endl*;

输出为：

first 41 second 6.3 third nico

make\_tuple

例：程序stl\_test14

*string* str = "default";

auto x = *make\_tuple*(str); // x类型为tuple<string>

*get*<0>(x) = "my value";

*cout* << "x: " << *get*<0>(x) << *endl*;

*cout* << "str: " << str << *endl*;

auto y = *make\_tuple*(ref(str)); // y类型为tuple<string&>

*get*<0>(y) = "my value";

*cout* << "y: " << *get*<0>(y) << *endl*;

*cout* << "str: " << str << *endl*;

输出为：

x: my value

str: default

y: my value

str: my value

std::tie()