案值传递：字符串，数字

/\*

#include <iostream>

using namespace std;

int A(int x);

int main ()

{

cout<<"Enter a number:"<<endl;

int a;

cin>>a;

A(a);

cout<<a;

return 0;

}

int A(int x)

{

return x;

}

\*/

#include <iostream>

#include <string>

using namespace std;

void callme2(string); // pass by value

int main()

{

string headline2("Lettuce Prey");

callme2(headline2);

cout << "headline2: " << headline2 << endl;

return 0;

}

void callme2(string sb)

{

cout << "String passed by value:\n";

cout << " \"" << sb << "\"\n";

}

不可案值传递：对象，交换

void swapv(int a, int b) // try using values

{

int temp;

temp = a; // use a, b for values of variables

a = b;

b = temp;

}

#include <iostream>

using namespace std;

void A(int x);

int main ()

{

cout<<"Enter a number:"<<endl;

int a;

cin>>a;

A(a);A(a);

A(a);

return 0;

}

void A(int x)

{

cout<< x<<endl;

}

/\*

// strngbad.cpp -- StringBad class methods

#include <cstring> // string.h for some

#include "strngbad.h"

using namespace std;

// initializing static class member

int StringBad::num\_strings = 0;

// class methods

// construct StringBad from C string

StringBad::StringBad(const char \* s)

{

len = strlen(s); // set size

str = new char[len + 1]; // allot storage

strcpy(str, s); // initialize pointer

num\_strings++; // set object count

cout << num\_strings << ": \"" << str

<< "\" object created\n"; // For Your Information

}

StringBad::StringBad() // default constructor

{

len = 4;

str = new char[4];

strcpy(str, "C++"); // default string

num\_strings++;

cout << num\_strings << ": \"" << str

<< "\" default object created\n"; // FYI

}

StringBad::~StringBad() // necessary destructor

{

//cout << "\"" << str << "\" object deleted, "; // FYI

//--num\_strings; // required

//cout << num\_strings << " left\n"; // FYI

delete [] str; // required

}

std::ostream & operator<<(std::ostream & os, const StringBad & st)

{

os << st.str;

return os;

}

// vegnews.cpp -- using new and delete with classes

// compile with strngbad.cpp

#include <iostream>

#include "strngbad.h"

using namespace std;

void callme2(StringBad); // pass by value

int main()

{

StringBad headline2("Lettuce Prey");

callme2(headline2);

callme2(headline2);

//cout << "headline2: " << headline2 << endl;

return 0;

}

# 一、P383 超级重要

**注意：注意：注意：关于引用（红色的）返回值如果是新创的，不可以引用；如果本来就存在的全局变量，可以引用（比如改变顺序）**

**参数是引用。将参数声明为引用目的是为了提高效率。如果案值传递TIME对象，代码的功能相同，但引用传递速度更快，使用的内存少。**

**然而返回值不能用引用。因为函数将创建一个新的YIME对象（SUm），来表示另外的两个TIME对象。返回对象（如代码所做那样）将创建对象副本，而调用函数可以使用它。然而，如果返回类型为TIME&，则引用对象将是sum。但由于sum是局部变量，在函数结束后被删除，因此引用指向一个不存在的对象。使用返回类型TIME意味着程序再删除sum之前构造他的拷贝，调用函数将使用该拷贝。**

**1.1**

**const Time Time::Sum(const Time & t) const**

**{**

**Time sum;**

**sum.minutes = minutes + t.minutes;**

**sum.hours = hours + t.hours + sum.minutes / 60;**

**sum.minutes %= 60;**

**return sum;**

**}**

**1.2.**

**const Stock & Stock::topval(const Stock & s) const**

**{**

**if (s.total\_val > total\_val)**

**return s;**

**else**

**return \*this;**

**}**

// strquote.cpp -- different designs

#include <iostream>

#include <string>

using namespace std;

string version1(const string & s1, const string & s2);

const string & version2(string & s1, const string & s2); // has side effect

//const string & version3(string & s1, const string & s2); // bad design

int main()

{

string input;

string copy;

string result;

cout << "Enter a string: ";

getline(cin, input);

copy = input;

cout << "Your string as entered: " << input << endl;

result = version1(input, "\*\*\*");

cout << "Your string enhanced: " << result << endl;

cout << "Your original string: " << input << endl;

result = version2(input, "###");

cout << "Your string enhanced: " << result << endl;

cout << "Your original string: " << input << endl;

/\*

cout << "Resetting original string.\n";

input = copy;

result = version3(input, "@@@");

cout << "Your string enhanced: " << result << endl;

cout << "Your original string: " << input << endl;

// cin.get();

// cin.get();\*/

return 0;

}

string version1(const string & s1, const string & s2)

{

string temp;

temp = s2 + s1 + s2;

return temp;

}

const string & version2(string & s1, const string & s2) // has side effect

{

s1 = s2 + s1 + s2;

// safe to return reference passed to function

return s1;

}

const string & version3(string & s1, const string & s2) // bad design

{

string temp;

temp = s2 + s1 + s2;

// unsafe to return reference to local variable

return temp;

}

// swaps.cpp -- swapping with references and with pointers

#include <iostream>

void swapr(int & a, int & b); // a, b are aliases for ints

void swapp(int \* p, int \* q); // p, q are addresses of ints

void swapv(int a, int b); // a, b are new variables

int main()

{

using namespace std;

int wallet1 = 300;

int wallet2 = 350;

cout << "wallet1 = $" << wallet1;

cout << " wallet2 = $" << wallet2 << endl;

cout << "Using references to swap contents:\n";

swapr(wallet1, wallet2); // pass variables

cout << "wallet1 = $" << wallet1;

cout << " wallet2 = $" << wallet2 << endl;

cout << "Using pointers to swap contents again:\n";

swapp(&wallet1, &wallet2); // pass addresses of variables

cout << "wallet1 = $" << wallet1;

cout << " wallet2 = $" << wallet2 << endl;

cout << "Trying to use passing by value:\n";

swapv(wallet1, wallet2); // pass values of variables

cout << "wallet1 = $" << wallet1;

cout << " wallet2 = $" << wallet2 << endl;

// cin.get();

return 0;

}

void swapr(int & a, int & b) // use references

{

int temp;

temp = a; // use a, b for values of variables

a = b;

b = temp;

}

void swapp(int \* p, int \* q) // use pointers

{

int temp;

temp = \*p; // use \*p, \*q for values of variables

\*p = \*q;

\*q = temp;

}

void swapv(int a, int b) // try using values

{

int temp;

temp = a; // use a, b for values of variables

a = b;

b = temp;

}