**函数应该在被调用的前面声明**

int GetSize(string str); // 函数声明

int main()

{

auto size = GetSize("ABC");

cout << "Press any key to end" << endl;

cin.get();

return 0;

}

int GetSize(string str) // 函数定义，如果没有声明会声明并定义

{

return str.length();

}

**局部静态对象**

int GetSize(string str)

{

static int callNum = 0; // 程序生成时初始，调用函数时不会调用这段代码

callNum++; // 每次调用函数时加1

cout << callNum << endl;

return str.length();

}

**参数类型**

void Add(int &num); // 引用类型，Add内部更改，外部也会更改

// 不能改变num的值（其实默认int的传值参数，改变num外部也不会改变）

// 编程规范中，如果不想改变外部num的值，应该将他设为const

int AddConst(const int num)

{

return num + 1;

}

// 数组参数，数组参数默认使用引用类型

void ArrayFun(const int \*arr, const int \*arrEnd);

void ArrayFun(const int arr[]);

void ArrayFun(const int arr[10]); // 期望传如10个元素，实际不一定

// 指针参数

void PFun(int \*p);

void PFun(int \*p[10]); // 10个指针

void PFun(int (\*p)[10]); // 指向数组的指针

**initializer\_list类型**

#include <initializer\_list>

// 与vector相似，initializer\_list是可变数组，但initializer\_list中的元素是常量

void ArrFun(initializer\_list<int> arr);

**不要返回局部对象的引用或指向局部对象的指针**

// 这里定义返回类型为引用类型

const string &Fun1()

{

return "abc";

// 如上代码相当与

// 这里定义了局部变量temp，变量的地址放在栈中的某一地址xxxx中

string temp = "abc";

// 这里return temp，因为函数返回引用类型，所以返回的是栈中的某一地址xxxx

return temp;

// 函数推出后会清理函数内部分配的栈内存，所有地址xxxx的内容（变量的地址）被清理

}

**引用返回左值**

string str = "abc";

Fun1(str) = ""; // 如果函数返回的是引用类型

**返回数组**

返回数组无法复制，返回数组的两种方式，难以返回局部内定义的变量

string ss[10] = { "ABC", "EF" };

string (\*Fun1())[10]

{

return &ss;

}

using stringArrry = string[10];

stringArrry\* Fun2()

{

return &ss;

}

// 以下返回局部变量地址为错误方式

string (\*Fun1())[10]

{

string ss[10] = { "ABC", "EF" };

// ss是堆的地址

// &ss返回的是栈的地址，栈内存在退出函数后清理

return &ss; // 错误

}

推荐使用的数组返回方式

vector<string> Fun2()

{

vector<string> list = { "ABC", "EF" };

return list;

}

**函数指针**

vector<string> Fun()

{

vector<string> list = { "ABC", "EF" };

return list;

}

int main()

{

// 指向函数的指针

vector<string> (\*FunP)();

FunP = Fun;

for(auto s : FunP()){

cout << s << endl;

}

return 0;

}