第6章 练习

6.1节

6.1

```
实参(argument):
    全称为"实际参数"是在调用时传递给函数的参数．实参可以是常量、变量、表达式、函数等， 无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值，以便把这些值传送给形参。 因此应预先用赋值，输入等办法使实参获得确定值。

形参(parameter):
全称为"形式参数" 由于它不是实际存在变量，所以又称虚拟变量。是在定义函数名和函数体的时候使用的参数,目的是用来接收调用该函数时传入的参数.在调用函数时，实参将赋值给形参。因而，必须注意实参的个数，类型应与形参一一对应，并且实参必须要有确定的值。
```

6.2

```
1．返回s是string类型与函数要求类型不匹配
2．函数缺少返回类型
3．函数形参存在同名
4．函数体缺少花括号
```

6.3

```
long long fact (int n ) {
    // 缺少异常检测
    long long sum = 1;
    while(n > 1)
        sum *= n--;
    return sum;
}
```

6.4

略（main种调用fact函数即可）

6.5

```
cout<<(num > 0 ? num : -num);
```

6.6

略

6.7

```
int Ex6_7() {
    static int count = 0;
    return count++;
}
```

6.8 - 6.9

略

6.2节

6.10

```
void Ex6_10(int *p1, int *p2) {
    int temp = *p1;
    *p1 = *p2;
    *p2 = *p1;
}
```

6.11 - 6.15

略

6.16

```
函数参数应该设定为 const string & s
```

6.17

```cpp
bool Ex6_17(const std::string &s) {
    for (auto sval : s)
        if (!std::isupper(sval))
            return false;
    return true;
}

void Ex6_17_2(std::string &s) {
    for (auto &sval : s)
        if (!std::isupper(sval))
            sval = toupper(sval);
}
```

6.18

```
1. bool compare(const matrix&, const matrix &);
2. vector<int>::iterator change_val(const int val,
vector<int>::iterator viter)l
```

6.19

```
1. 不合法      2. 合法
3. 合法        4. 合法
```

6.20

函数不会改变传入的参数就可以是常量引用

6.21

```cpp
int cmp(int ival, int * pval){
    return ival > *pval ? ival : *pval;
}
```

6.22

```cpp
void exchange(int *pval1, int *pval2){
    int * temp = pval1;
    pval1 = pval2;
    pval2 = temp;
}
```

6.23

略

6.24

```
函数参数中[10]并不代表实际传入的数组大小，有可能会
```

6.25 - 6.26

```
int main(int argc, char** argv)//实参列表
{
    string str;
    for (int i = 1; i != argc; ++i) {
        str += argv[i];
        str += " ";
    }

    cout << str <<endl;
    return 0;
}
```

6.27

```
void Ex6_27(initializer_list<int> ival_list) {
    int sum = 0;
    for (auto ival : ival_list)
        sum += ival;
    cout<<sum<<endl;
}
```

6.28

```
const string & elem
```

6.29

```
需要分情况讨论，并没有强制要求
```

6.3节

6.30

略

6.31

```
返回在函数内创建的变量的引用无效
```

6.32

```
合法；给数组ia赋值
```

6.33

```
void Ex6_33(vector<int> ivec, int i = 0) {
    if (i == ivec.size())
        return;
    cout<<ivec[i++]<<" ";
    Ex6_33(ivec,i);
}
```

6.34

```
如果传入参数是负数，则递归永远不会停止
```

6.35

```
假设val = 5,
则 ival*fac(ival--)  等价于 5*fac(5),最终一直传递5
```

6.36

```
// vector<string> &svec,  将变量名替换成函数名即可
vector<string> & fun();
```

6.37

```
// 类型别名
using vec_string_10 = vector<string>;
vec_string_10 & fun();

// 尾置返回类型
auto fun() -> vector<int>string;

// decltype关键字
decltype(svec) fun();
```

6.38

```
auto attPtr(int i ) -> int (&p)[5]
```

6.4节

6.39

```
1. 顶层const无法区分函数
2. 合法
3. 合法
```

6.5节

6.40

```
1. 合法
2. 不合法；  默认实参右边的参数也要有默认实参
```

6.41

```
1. 不合法
2. 合法
3. 合法；发生了隐式类型转换
```

6.42

略

6.43

```
内联函数建议放头文件中
```

6.44

略（就在函数返回类型前加一个inline关键字即可）

6.45

略

6.46

```
不可以，string不是字面值类型
```

6.47

略

6.48

```
不合理；只要有输入，assert都为真
```

6.6节

6.49

略（知识点已经总结）

6.50 - 6.51

```
1. 二义性；      2. f(int)
3. f(int, int)  4. f(double, double)
```

6.52

```
1. 类型提升
2. 算术类型转换
```

6.53

```
1. 不合法；顶层const起不到区分作用
2. 不合法；同上
3. 合法；
```

6.7节

6.54 - 6.56

```
int fun1(int ival1, int ival2){
    return ival1 + ival2;
}
int fun2(int ival1, int ival2){
    return ival1 - ival2;
}
int fun3(int ival1, int ival2){
    return ival1 * ival2;
}
int fun4(int ival1, int ival2){
    return ival1 / ival2;
}
void Ex6_54_56(){

```

```
    using fun1_ptr = int (*)(int,int);

    vector<fun1_ptr> vec_fun1 = {fun1,fun2,fun3, fun4};
    int ival1= 44, ival2 = 4;
    for (auto fun : vec_fun1)
        cout<<fun(ival1,ival2)<<endl;
}
```