C++primer 第五版 第六章 答案

6.1 答：实参是形参的初始值，实参的类型必须与对应的形参类型匹配。形参是在函数的形参列表中进行定义的，我们实参的值去初始化形参。

（1）形参变量只有在被调用时才分配内存单元，在调用结束时， 即刻释放所分配的内存单元。因此，形参只有在函数内部有效。 函数调用结束返回主调函数后则不能再使用该形参变量。

（2）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。

（3）实参和形参在数量上，类型上，顺序上应严格一致， 否则会发生“类型不匹配”的错误。

（4）函数调用中发生的数据传送是单向的。 即只能把实参的值传送给形参，而不能把形参的值反向地传送给实参。 因此在函数调用过程中，形参的值发生改变，而实参中的值不会变化。

（5）当形参和实参不是指针类型时，在该函数运行时，形参和实参是不同的变量，他们在内存中位于不同的位置，形参将实参的内容复制一份，在该函数运行结束的时候形参被释放，而实参内容不会改变。而如果函数的参数是指针类型变量，在调用该函数的过程中，传给函数的是实参的地址，在函数体内部使用的也是实参的地址，即使用的就是实参本身。所以在函数体内部可以改变实参的值。

6.2 答：（a）返回类型错误；（b）没有定义函数的返回类型；（c）函数体缺少左花括号；（d）函数体缺少一对花括号；

6.3 答：

```
#include <iostream>
#include<string>
using namespace std;
int fact(int n)
{
    int val=1;
    for (int ret = 2; ret <= n; ret++)
    {
        val *= ret;
    }
    return val;
}
int main()
{
        cout<<fact(2);
}
```

6.4 答：

```
int fact(int n)
{
    int val=1;
    while (n > 1)
```

```
    {
        val *= n;
        n--;
    }
    return val;
}
```

6.5 答：return n > 0 ? n : (-1)*n;

6.6 答：形参是一种自动对象，函数开始的时候为形参申请存储空间，因为形参定义在函数体组用域之内，所以一旦函数终止，形参也就被销毁。局部对象如果是非静态的，那么其生存周期是从其被创建到函数体结束。静态局部变量在程序路劲第一经过对象定义语句是初始化，并且直到程序终止才被销毁，在此期间即使对象所在的函数结束执行也不会对它有影响。

例：

```
#include <iostream>
#include<string>
using namespace std;
int fact(int n)
{
    int i = 2;
    static int cnt = 0;
    ++cnt;
    cout << cnt * 2 <<    ' ';
    return n > 0 ? n : (-1)*n;

}
int main()
{
    for (int j = 0; j != 4; ++j)
    {
        cout << fact(j) << endl;
    }
}
```

6.7 答：

```
#include <iostream>
#include<string>
using namespace std;
int fact()
{
    static int cnt = 0;
    return cnt++;
}
int main()
{
    for (int j = 0; j != 4; ++j)
    {
        cout << fact() << endl;
    }
```

```
}
```
6.8 答：chapter6.h 文件
```
#ifndef chapter6_h
#define chapter9_h
int fact();
#endif chapter6_h
```
6.9 答：chapter6.cpp 文件
```
#include <iostream>
#include<string>
#include"chapter6.h"
using namespace std;
int fact()
{
	static int cnt = 0;
	return cnt++;
}
```
chapter6.cpp main 文件
```
#include <iostream>
#include<string>
#include"chapter6.h"
using namespace std;
int main()
{
	for (int j = 0; j != 4; ++j)
	{
		cout << fact() << endl;
	}
}
```
6.10 答：
```
#include <iostream>
#include<string>
using namespace std;
void swap(int *ip1, int *ip2)
{
	int temp;
	temp = *ip1;
	*ip1 = *ip2;
	*ip2 = temp;
}
int main()
{
	int ival1 = 1, ival2 = 3;
	int *p1 = &ival1, *p2 = &ival2;
	swap(p1, p2);
	cout << ival1 << ' ' << ival2;
```

```
}
```
6.11 答：
```
void reset(int &i)   {      i = 0;   }
```
6.12 答：
```
void swap(int &ip1, int &ip2)
{
      int temp;
      temp = ip1;
      ip1 = ip2;
      ip2 = temp;
}
int main()
{
      int ival1 = 1, ival2 = 3;
      swap(ival1, ival2);
      cout << ival1 << ' ' << ival2;
}
```
6.13 答：第一种是传值调用，在函数的调用过程中不会修改实参的值，第二种是传地址引用调用，在函数调用过程会修改实参的值。

6.14 答：如果希望在函数调用过程去修改实参的值，要没使用引用传递（或指针），如上面交换两个整数值得例子。如果不希望改变传进来的实参的值，那么就不能用引用传递。如上面那个输出绝对值的例子。

6.15 答：cosnt 引用可以避免拷贝又不会在函数调用的过程中修改实参值，s 如果是非常量引用那么在函数调用的过程会出现被修改的情况。而 occurs 需要在函数调用过程改变，所以用引用，如果是常量引用那么 occurs 的值就不能被修改。不需要再函数调用过程中去改变 c 的值，所以不应该使用引用。

6.16 答：应该设置成为 cosnt 引用，这样既可以避免拷贝，又可以直接使用字符串常量作为参数。

6.17 答：
```
#include <iostream>
#include<string>
using namespace std;
bool my_isupper(const string &s)
{
      string::size_type size = s.size();
      for (decltype(s.size()) index = 0; index != size; index++)
      {
            if (s[index] >= 'A'&& s[index] <= 'Z')
                  return true;
      }
}
string my_tolower(string &s)
{
      if (my_isupper(s))
      {
            string::size_type size = s.size();
            for (decltype(s.size()) index = 0; index != size; ++index)
```

```
                s[index]=tolower(s[index]);
            return s;
        }
        else
            return s;
}
int main()
{
        string s("chenXun");
        s = my_tolower(s);
        cout << s << endl;
}
```

6.18 答：（a）bool compare(matrix&, matrix&);

（b）`vector<int>::iterator chang_val(int , vector<int>::iterator)`

6.19 答：（a）不合法，接收的参数个数；（b）合法；（c）合法；（d）合法；

6.20 答：当引用形参不需要去修改实参值得时候应该用 const，如果形参应该是常量引用而我们把它修改成普通引用，会产生意想不到的结果，比如修改了实参的值。

6.21 答：int(int i , const int *p){return i > *p ? i : *p;}指针的类型是 int*；

6.22 答：void swapPtr（int **p1，int **p2）；

6.23 答：

```
#include<string>
using namespace std;
void print(int matrix[], int ix)
{
        cout << matrix[0] << endl;;
}

int main()
{
        int i = 0, j[2] = { 0, 1 };
        print(j, i);
        return 0;
}
```

6.24 答：没有什么问题，依次输出数组元素；

6.25 答：

```
#include <iostream>
#include <string>
int main(int argc, char *argv[])
{
    if (argc > 2) {
        std::string str = argv[1];
        str += argv[2];
        std::cout << str << std::endl;
    } else {
        std::cout << "error" << std::endl;
```

```
    }
    return 0;
}
```

6.26 答：

```
#include <iostream>
#include <string>
int main(int argc, char *argv[])
{
    for (int i=0; i<argc; i++)
    {
        std::cout << i+1 << "," << argv[i] << std::endl;
    }
    return 0;
}
```

6.27 答：

```
#include<vector>
using namespace std;
int my_add(const int *beg,const int *end)
{
        int sum = 0;
        while ( beg != end )
        {
                sum += *beg++;
        }
        return sum;
}
int main()
{
        int sumVal;
        const int a[] = { 1,1,1,1,1,1,1,1,1,1 };
        sumVal=my_add(a, a+sizeof(a)/sizeof(*a));
        cout << sumVal << endl;
}
```

6.28 答：string

6.29 答：不能，循环变量声明为 const 就不能递加了。

6.30 答：编译通不过，会报告没有返回值；不是所有的路劲都有返回值；

6.31 答：返回局部对象的引用是无效的，当需要给返回的引用赋值，那么返回常量的引用就是无效的。

6.32 答：该程序合法，其功能为：将数组的每一个元素赋值为 0；

6.33 答：

```
#include <iostream>
#include <vector>
using namespace std;
void my_print(vector<int> ivec);
int main()
{
```

```
    vector<int> ivec;
    for (decltype(ivec.size()) i = 0; i != 200; i++)
    {
        ivec.push_back(i);
    }
    my_print(ivec);
    return 0;
}
static size_t i = 0;
void my_print(vector<int> ivec)
{
    if(i != ivec.size())
    {
        cout << ivec[i++]<< endl;
        my_print(ivec);
    }
}
```

6.34 答：如果实参为负数，会发生程序堆栈溢出。

6.35 答：是用 val--无法递归。

6.36 答：string (&fun(string s))[10]

6.37答：

```
string(&fun(string s))[10];
typedef    string sPtr[10];
using sPtr = string[10];
sPtr &fun(string s);
string str[10];
decltype(str) &fun(string s);
auto fun(string s)->string (&)[10];
```

6.38 答：

```
int odd[] = { 1, 3, 5, 7, 9 };
int even[] = { 0, 2, 4, 6, 8 };
decltype(odd) &arrPtr(int i)
{
    return (i % 2) ? odd : even;
}
```

例：

```
#include<iostream>
using namespace std;
int odd[] = { 1, 3, 5, 7, 9 };
int even[] = { 0, 2, 4, 6, 8 };
```

```
decltype(odd) *arrPtr(int i)
{
    return (i % 2) ? &odd : &even;
}
int main()
{
    cout <<( *arrPtr(4))[2] << endl;;
}
```

6.39 答：（a）顶层 const，重复声明；（b）非法，返回类型不一致而参数相同；（c）重载；

6.40 答：第二个声明错误；第一个形参提供了默认值，那么后面的形参都必须提供默认值；

6.41 答：（a）非法，没有给第一个形参传值；（b）合法；（c）非法，第三个形参必须提供了默认值，那么第二形参不能省略；

6.42 答：

```
#include<iostream>
#include <string>
#include<vector>
using namespace std;
string make_plural(size_t ctr, const string &word = "success",
    const string &ending="es")
{
    return (ctr > 1) ? word + ending : word;
}
int main()
{
    size_t cnt = 1;
    cout << make_plural(cnt, "success", "es") << endl;

    cnt = 2;
    cout << make_plural(cnt, "failure", "s") << endl;
    return 0;
}
```

6.43 答：（a）放在定义文件中，这个函数定义函数的实现；（b）放在头文件中，这是函数声明。

6.44 答：

```
inline const string &
shorterString(const string &s1, const string &s2)
{
        return s1.size() <= s2.size() ? s1 : s2;
}
```

6.45 答：根据需要就可以写成内联函数，没必要非要把所有函数都写成内联函数。一般来说，内联机制用于优化规模较小、流程直接、频繁调用的函数。很多编译器都不支持内联递归函数，而一般一个 75 行的函数也不大可能在调用点内内联的展开。

6.46 答：可以，

```
cosntexpr const string &
shorterString(const string &s1, const string &s2)
{
        return s1.size() <= s2.size() ? s1 : s2;
```

}

6.47 答：在 main 函数体写加上#ifndef NDEBUG　　#endif

6.48 答：循环体就是不断的从输入设备输入字符串，这里 assert 不合理，如果结束输入那么就会提示输入失败。

6.49 答：函数匹配的第一步是选定本次调用对应的重载函数，集合中的函数称为候选函数，候选函数具备两个特征：一是与被调用的函数同名，二是其声明在调用点可见。函数匹配的第二步是考察本次提供的实参，然后从候选函数中选出能被这组实参调用的函数，这些新选出来的函数称为可行函数。可行函数有两个特征：一是其形参数量与本次调用提供的实参数量相等，二是每个实参的类型与对应的形参类型相同，或者能转换成形参类型。

6.50 答：（a）具有二义性。（b）调用 viod f（int）；（c）调用 viod f（int，int）；（d）调用 viod f（double，double）；

6.51 答：

```
#include <iostream>
#include <vector>
#include<string>
#include<cassert>
using namespace std;
void f()
{
	cout << "调用 void f()";
}
void f(int i)
{
	cout << "调用 void f(int i)" << endl;
}
void f(int, int)
{
	cout << "调用 void f(int, int)" << endl;
}
void f(double, double = 3.14)
{
	cout << "调用 void f(double, double = 3.14)" << endl;
}

int main()
{
//	f(2.56,42);
	f(42);
	f(42, 0);
	f(2.56, 3.14);
}
```

6.52 答：（a）3 类型提升,从 char 提升为 int 型；（b）标准转换，从 double 型转换成 int 型；

6.53 答：（a）。（b）没有影响，只是声明重载了而已；（c）有影响，对于第一个那么第二个为非法声明就是重复声明，形参是按值，不能通过形参是否是 const 来重载函数；

6.54 答：typedef int fun(int, int); vector<fun*> v1;

6.55 答：#include <iostream>

```
#include <vector>
using namespace std;
typedef int fun(int, int);

int sum(int a, int b){return a + b;}
int reduce(int a, int b)
{
	if (a >= b)
		return a - b;
	else
		return b - a;
}
int ride(int a, int b){return a*b;}
int divide(int a, int b){return a / b;}
int main()
{
	int a = 20, b = 10;
	vector<fun*> ivec;
	ivec.push_back(sum);
	ivec.push_back(reduce);
	ivec.push_back(ride);
	ivec.push_back(divide);
	for (auto it = ivec.begin(); it != ivec.end(); ++it)
	{
		cout << (*it)(a, b) << endl;
	}
	return 0;
}
```

6.56 答：运行上题代码。