C++Primer

第六章、函数

6.4、求阶乘

```
/******编写函数求阶乘********/
#include<iostream>
using namespace std;

int fact(int val)
{
    if (val < 0)
    {
        return -1;
    }
    int i, fact = 1;
    for (i = 1; i != val + 1; ++i)
    {
        fact *= i;
    }
    return fact;
}
int main604()
{
    int num;
    cout << "请输入一个数：";
    cin >> num;
    cout << num << "的阶乘为：" << fact(num) << endl;
    system("pause");
    return 0;
}
```

6.5 求一个数的绝对值

```
/************求一个数的绝对值************/
#include<iostream>
using namespace std;

double myABS(double num)
{
    if (num < 0)
    {
        num *= -1;
    }
    return num;
}
```

```
int main605()
{
    double num;
    cout << "请输入一个数";
    cin >> num;
    cout << num << "绝对值为：" << myABS(num) << endl;
    system("pause");
    return 0;
}
```

6.6

```
#include<iostream>
using namespace std;
double myADD(double val1, double val2)//val1 和val2是形参
{
    double result = val1 + val2;//result是普通局部变量
    static unsigned iCnt = 0;//iCnt是静态局部变量
    ++iCnt;
    cout << "该函数已经累计执行了" << iCnt << "次" << endl;
    return result;
}
int main()
{
    double num1, num2;
    cout << "请输入两个数";
    while (cin >> num1 >> num2)
    {
        cout << num1 << "与" << num2 << "的求和结果是：" << myADD(num1, num2) << endl;
    }
    system("pause");
    return 0;
}
```

6.7

```
#include<iostream>
using namespace std;
unsigned myCnt()
{
    static unsigned iCnt = -1;
    ++iCnt;
    return iCnt;
}
int main()
{
    cout << "请输入任意字符后按回车继续";
    char ch;
    while(cin >> ch)
    {
```

```
        cout << "输入的字符为：" << ch << "函数执行的次数为：" << myCnt() << endl;
    }
    system("pause");
    return 0;
}
```

6.10

```
#include<iostream>
using namespace std;
void mySWAP(int *p, int *q)
{
    int temp;
    temp = *p;
    *p = *q;
    *q = temp;
}
int main()
{
    int a = 5, b = 6;
    int *r = &a, *s = &b;
    cout << "交换前a = " << a << "b = " << b << endl;
    mySWAP(r, s);
    cout << "交换后a = " << a << "b = " << b << endl;
    system("pause");
    return 0;
}
```

6.12

```
#include<iostream>
using namespace std;
void mySWAP(int &p, int &q)
{
    int temp = p;
    p = q;
    q = temp;
}
int main()
{
    //区别：无需使用指针来间接指向变量，可以直接使用，美观，而且避免指针拷贝
    int a = 5, b = 6;
    cout << "交换前a = " << a << "b = " << b << endl;
    mySWAP(a, b);
    cout << "交换后a = " << a << "b = " << b << endl;
    system("pause");
    return 0;
}
```

6.13

```
#include<iostream>
using namespace std;
void a(int);//传值参数  实参和形参相互独立，实现拷贝给形参进行计算，实参本身没有改变
void b(int&);//传引用参数
//引用其实是内存地址的另一个别名，传递的是内存地址，所以实参和形参指向的是同一片内存空间
int main()
{
    int s = 0, t = 10;
    a(s);
    cout << s << endl;
    b(t);
    cout << t << endl;
    system("pause");
    return 0;
}
void a(int i)
{
    ++i;
    cout << i << endl;
}
void b(int &j)
{
    ++j;
    cout << j << endl;
}
```

6.17

```
#include<iostream>
#include<string>
using namespace std;

bool isHaveBig(const string str)
{
    for (auto c : str)
        if (isupper(c))
            return true;
    return false;
}

void lowerCase(string &str)
{
    for (auto &c : str)
        c = tolower(c);
}
int main()
{
    cout << "请输入一个字符串：" << endl;
```

```
    string str;
    int i;
    cin >> i;
    cin >> str;
    if (isHaveBig(str))
    {
        lowerCase(str);
        cout << "转换后的字符串是：" << str
            <<endl;
    }
    else
    {
        cout << "该字符串不含大写字母无需转换" << endl;
    }
    system("pause");
    return 0;
}
```

6.21比较两个数的大小值和指针

```
#include<iostream>
using namespace std;
int myCompare(const int val, const int *p) //第一个是值，所以不用改变，而后一个比较的是指针指向的值，所以也不需要改变
{
    return(val > *p) ? val : *p;
}
int main()
{
    int num1, num2, num3;
    cout << "请输入两个数";
    cin >> num1 >> num2;
    num3 = myCompare(num1, &num2);
    cout << "较大的数为：" << num3 << endl;
    system("pause");
    return 0;
}
```

6.22

```
#include<iostream>
using namespace std;
//该函数即不交换指针，也不交换指针所指的内容
void SwaPointer1(int *p, int *q)
{
    int *temp = p;
    p = q;
    q = temp;
}
```

```
//该函数交换指针指的内容
void SwaPointer2(int *p, int *q)
{
    int temp = *p;
    *p = *q;
    *q = temp;
}
//该函数交换指针本身的值，即交换指针所指的内存地址
void SwaPointer3(int *&p, int *&q)
{
    int *temp = p;
    p = q;
    q = temp;
}
int main()
{
    int a = 5, b = 10;
    int *p = &a, *q = &b;
    cout << "交换前：" << endl;
    cout << "p的值是" << p << "q的值是：" << q << endl;
    cout << "p所指的值是：" << *p << ",q所指的值是：" << *q << endl;
    SwaPointer1(p, q);
    cout << "交换后：" << endl;
    cout << "p的值是" << p << "q的值是：" << q << endl;
    cout << "p所指的值是：" << *p << ",q所指的值是：" << *q << endl;

     a = 5, b = 10;
      p = &a, q = &b;
    cout << "交换前：" << endl;
    cout << "p的值是" << p << "q的值是：" << q << endl;
    cout << "p所指的值是：" << *p << ",q所指的值是：" << *q << endl;
    SwaPointer2(p, q);
    cout << "交换后：" << endl;
    cout << "p的值是" << p << "q的值是：" << q << endl;
    cout << "p所指的值是：" << *p << ",q所指的值是：" << *q << endl;

     a = 5, b = 10;
     p = &a, q = &b;
    cout << "交换前：" << endl;
    cout << "p的值是" << p << "q的值是：" << q << endl;
    cout << "p所指的值是：" << *p << ",q所指的值是：" << *q << endl;
    SwaPointer3(p, q);
    cout << "交换后：" << endl;
    cout << "p的值是" << p << "q的值是：" << q << endl;
    cout << "p所指的值是：" << *p << ",q所指的值是：" << *q << endl;
    system("pause");
    return 0;
}
```

6.23

```cpp
#include<iostream>
using namespace std;
//参数是常数整型指针
void print1(const int *p)
{
    cout << *p << endl;
}
//参数有两个，分别是常亮整型指针和数组的容量
void print2(const int *p, const int sz)
{
    int i = 0;
    while (i != sz)
    {
        cout << *p++ << endl;
        ++i;
    }
}
//参数有两个，分别是数组的首尾边界
void print3(const int *b, const int *e)
{
    for (auto q = b; q != e; ++q)
        cout << *q << endl;
}
int main()
{
    int i = 0, j[2] = { 0,1 };
    print1(&i);
    print1(j);
    print2(&i, 1);
    print2(j, sizeof(j) / sizeof(*j));
    auto b = begin(j);
    auto e = end(j);
    print3(b, e);

    system("pause");
    return 0;
}
```

6.27

```cpp
#include<iostream>
using namespace std;
int iCount(initializer_list<int> il)
{
    int count = 0;
    for (auto val : il)
    {
        count += val;
    }
    return count;
}
```

```
int main()
{
    //使用列表初始化的方式构建initializer_list<int>对象
    //然后把它作为实参传递给函数iCount
    cout << "1,6,9的和是：" << iCount({ 1,6,9 }) << endl;
    cout << "4,5,9,18的和是" << iCount({ 4,5,9,18 }) << endl;
    cout << "10,10,10,10,10,10,10,10,10,10的和是：" << iCount({
10,10,10,10,10,10,10,10,10,10 }) << endl;
    system("pause");
    return 0;
}
```

6.32

```
int &get(int *arry,int index)
{
    return array[index];
}
int main()
{
    int ia[10];
    for(int i = 0; i != 10;++i)
    {
        get(ia,i) = i;
    }
}
```

6.33

```
#include<iostream>
#include<vector>
using namespace std;
void print(vector<int> vInt, unsigned index)
{
    unsigned sz = vInt.size();
    if(!vInt.empty() && index<sz)
    {
    cout << vInt[index] << endl;
    print(vInt, index + 1);
    }
}
int main633()
{
    vector<int> v = { 1,3,5,7,9,11,13,15 };
    print(v, 0);
    system("pause");
    return 0;
}
```

6.42

```
#include<iostream>
#include<string>
using namespace std;

string make_plural(size_t ctr, const string &word, const string &ending = "s")
{
    return (ctr > 1) ? word + ending : word;
}

int main()
{
    cout << "success的单数形式是：" << make_plural(1, "success", "es");
    cout << "success的复数形式是：" << make_plural(2, "success", "es") << endl;
    //一般情况下调整该函数只需要两个实参
    cout << "failure的单数形式是：" << make_plural(1, "failure") << endl;
    cout << "failure的单数形式是：" << make_plural(2, "failure") << endl;

    system("pause");
    return 0;
}
```

6.56

```
#include<iostream>
#include<vector>
using namespace std;
int func1(int a, int b)
{
    return a + b;
}
int func2(int a,int b)
{
    return a - b;
}
int func3(int a, int b)
{
    return a * b;
}
int func4(int a, int b)
{
    return a / b;
}
void Compute(int a, int b, int(*p)(int, int))
{
    cout << p(a, b) << endl;
}
int main()
{
    int i = 5, j = 10;
    decltype(func1)*p1 = func1, *p2 = func2, *p3 = func3, *p4 = func4;
```

```
    vector<decltype (func1)* > vF = { p1,p2,p3,p4 };
    for (auto p : vF) //遍历vector中的每个元素，依次调用四则运算函数
    {
        Compute(i, j, p);
    }
    system("pause");
    return 0;
}
```

第七章、类

7.6重写add，read，print，函数

```
#include<iostream>
using namespace std;
#include "Sales_data.h"
Sales_data add(const Sales_data &lhs, const Sales_data &rhs)
{
    Sales_data sum = lhs;
    sum.combine(rhs);
    return sum;
}
std::istream &read(istream &is, Sales_data &item)
{
    is >> item.bookNo >> item.units_sold >> item.sellingprice >> item.saleprice;
    return is;
}
ostream &print(ostream &os, const Sales_data &item)
{
    os << item.isben() << " " << item.units_sold << " " << item.Sellingprice << " " <<
item.saleprice << " " << item.discount;
    return os;
}
```

7.9

```
istream &read(istream &is, Person &per)
    {
        is >> per.strName >> per.strAddress;
        return is;
    }
    ostream &print(ostream &os, const Person &per)
    {
        os << per.getName() << per.getAddress();
        return os;
    }
```

7.15 、添加默认构造函数

```
Person() = default;
    Person(const string &name, const string &add)
    {
        strName = name;
        strAddress = add;
    }
    Person(istream &is) { is >> *this; }
```

7.23&7.24、Screen类和默认构造函数

```
class Screen
{
public:
    Screen() = default;//默认构造函数
    Screen(unsigned ht, unsigned wd) : height(ht), width(wd), contents(ht * wd, ' ') {}
    Screen(unsigned ht, unsigned wd, char c) : height(ht), width(wd), contents(ht * wd, c)
{}
    ~Screen();

private:
    unsigned height = 0, width = 0;
    unsigned cursor = 0;
    string contents;

};
```

7.25、三个函数

```
public:
    Screen& move(unsigned r, unsigned c)
    {
        cursor = r * width + c;
        return *this;
    }

    Screen& set(unsigned r, unsigned c, char ch)
    {
        contents[r * width + c] = ch;
        return *this;
    }
    Screen& display()
    {
        cout << contents;
        return *this;
    }
};
```

第八章、IO操作

8.1

```
#include<iostream>
using namespace std;
istream & f(istream &in)
{
    int v;
    while (in >> v, !in.eof())//直到遇到文件结束符才停止读取
    {
        if (in.bad())
            throw runtime_error("IO流错误");
        if (in.fail())
        {
            cerr << "数据错误，请重试:" << endl;
            in.clear();
            in.ignore(100, '\n');
            continue;
        }
        cout << v << endl;
    }
    in.clear();
    return in;
}

int main()
{
    cout << "请输入一些整数，按Ctrl+z结束" << endl;
    f(cin);
    system("pause");
    return 0;
}
```

8.4

```
//////编写函数，以读模式打开一个文件，将其内容读到一个string的vector中，将每一行作为一个单独的元素存与veector中


#include<iostream>
#include<fstream>
#include<string>
#include<vector>
using namespace std;
int main()
{
    ifstream in("data");
    if (!in)
    {
        cerr << "无法打开输入文件" << endl;
        return -1;
    }
```

```
    string line;
    vector<string>words;
    while (getline(in, line))//从文件中读取一行
    {
        words.push_back(line);//添加到vector中
    }
    in.close(); //读入完毕，关闭文件
    vector<string>::const_iterator it = words.begin();//迭代器
    while (it!=words.end())//遍历vector
    {
        cout << *it << endl;
        ++it;
    }
    system("pause");
    return 0;
}
```

8.5

```
//将while(getline(in,line))改写成为(while(in>>line))
```

8.9

```
#include<iostream>
#include<sstream>
#include<string>
#incluse<stdexcept>
using namespace std;
istream & f(istream &in)
{
    int v;
    while (in >> v, !in.eof())//直到遇到文件结束符才停止读取
    {
        if (in.bad())
            throw runtime_error("IO流错误");
        if (in.fail())
        {
            cerr << "数据错误，请重试:" << endl;
            in.clear();
            in.ignore(100, '\n');
            continue;
        }
        cout << v << endl;
    }
    in.clear();
    return in;
}

int main()
{
```

```
    ostringstream msg;
    msg<<"C++ Primer 第五版"<<endl;
    istringstream in(msg.set());
    f(in);
    system("pause");
    return 0;
}
```

8.10

```
#include <iostream>
#include<fstream>
#include<string>
#include<sstream>
#include<vector>
using namespace std;

int main()
{
    ifstream in("data");
    if (!in)
    {
        cerr << "无法打开输入文件" << endl;
        return - 1;
    }
    string line;
    vector<string>words;
    while (getline(in,line))
    {
        words.push_back(line);
    }
    in.close();
    vector<string>::const_iterator it = words.begin();
    while (it != words.end())
    {
        istringstream line_str(*it);
        string word;
        while (line_str >> word)
            cout << word << "";
        cout << endl;
            ++it;
    }

    system("pause");
    return 0;
}
```

8.11

```
#include <iostream>
```

```
#include<fstream>
#include<string>
#include<sstream>
#include<vector>
using namespace std;
struct PersonInfo {
    string name;
    vector<string>phones;
};
int main()
{
    string line, word;//分别保存来自输入的一行和单词
    vector<PersonInfo>people;//保存来自输入的所有记录
    istringstream record;
    while (getline(cin, line))
    {
        PersonInfo info;//创建一个保存此记录数据的对象
        record.clear();//重复使用字符串流时，每次都要调用clear
        record.str(line);//记录绑定时刚读入的行
        while (record >> word)                    info.phones.push_back(word);//保存他们
        people.push_back(info);//将此记录追加到people末尾
    }
    system("pause");
    return 0;
}
```

8 .13

```
#include<iostream>
#include<fstream>
#include<sstream>
#include<string>
#include<vector>
using namespace std;
struct PersonInfo {
    string name;
    vector<string>phines;
};

string format(const string &s) { return s; }
bool valid(const string &s)
{
    return true;
}


int main(int argc, char *argv[])
{
    string line, word;  //分别保存来自输入的一行和单词
    vector<PersonInfo>people;//保存来自输入的所有记录
    istringstream record;
```

```
    if (argc != 2)
    {
        cerr << "请给出文件名" << endl;
        return -1;
    }
    ifstream in(argv[1]);
    if (!in)
    {
        cerr << "无法打开输入文件" << endl;
        return -1;
    }
    while(getline(in, line))
    {
        PersonInfo info;//创建一个保存此记录数据的对象
        record.clear();//重新使用字符串流时，每次都要调用clear
        record.str(line);//将记录绑定到刚读入的行
        record >> info.name;//读取名字
        while (record >> word)  //读取电话号码
            info.phines.push_back(word);//保存他们
        people.push_back(info);//将此记录最加到people末尾
    }
    ostringstream os;
    for (const auto &entry : people)       //对pople中没一项
    {
        ostringstream formatted, badNums;//每个循环步创建对象
        for (const auto &nums : entry.phines)
        {
            if (!valid(nums))
            {
                badNums << " " << nums;//将数的字符串形式存入badNums
            }
            else
            {
                //将格式化的字符串“写入”formatted
                formatted << " " << format(nums);
            }
            if (badNums.str().empty()) //没有错误的数
            {
                os << entry.name << " " << formatted.str() << endl;//打印名字 和格式化的数

            }
            else
            {
                //否则打印名字和错误的数
                cerr << "iput error:" << entry.name << "invalid number(s)" << badNums.str()
<< endl;
            }

        }
        cout << os.str() << endl;
    }

    //system("pause");
```

```
    return 0;
}
```

第九章、顺序容器

9.4

```
#include<iostream>
#include<list>
#include<vector>
using namespace std;
bool findint(vector<int>v1, int num)
{
    int i;
    for (i = 1; i < v1.size(); ++i)
    {
        if (v1[i] == num)
        {
            return true;
        }
        return false;
    }
}
int main()
{
    vector<int> v2 = { 1,2,3,4,5,6,7 };
    int num = 8;
    if (findint(v2, num))
    {
        cout << "找到了" << endl;
    }
    else
    {
        cout << "没有找到" << endl;
    }
    system("pause");
    return 0;
}
```

9.5

```
#include<iostream>
#include<vector>
using namespace std;
```

```
vector<int>::iterator serch_vec(vector<int>::iterator beg, vector<int>::iterator end, int
val)
{
    for (; beg != end; beg++)  //遍历范围
    {
        if (*beg == val)    //检查是否与个给定值相等
            return beg;     //搜索成功 返回元素
        return end;         //搜索失败 返回尾迭代器
    }
}
int main()
{
    vector<int> v2 = { 1,2,3,4,5,6,7 };
    cout << serch_vec(v2.begin(), v2.end(), 3) - v2.begin() << endl;
    cout << serch_vec(v2.begin(), v2.end(), 8) - v2.begin() << endl;
    system("pause");
    return 0;
}
```

9.13

```
#include<iostream>
#include<vector>
#include<list>
using namespace std;
int main()
{
    list<int> ilist = { 1,2,3,4,5,6,7 };
    vector<int> ivec = { 7,6,5,4,3,2,1 };
    //容器类型不同，不能使用拷贝初始化
    //vector<double>ivec(ilist);

    //元素类型相同，因此可采用范围初始化
    vector<double>dvec(ilist.begin(), ilist.end());
    //容器类型不同，不能使用拷贝初始化
    //vector<double> dvecl(ivec);

    //元素类型相容，因此可采用范围初始化
    vector<double>dvecl(ivec.begin(), ivec.end());
    cout << dvec.capacity() << " " << dvec.size() << " " << dvec[0] << " " <<
dvec[dvec.size() - 1] << endl;
    cout << dvecl.capacity() << " " << dvecl.size() << " " << dvecl[0] << " " <<
dvecl[dvec.size() - 1] << endl;
    system("pause");
    return 0;
}
```

9.14

```
#include<iostream>
#include<string>
#include<vector>
#include<list>
using namespace std;
int main()
{
list<char*> slist = { "hello","world","!!!" };
vector<string> svec;
//容器类型不同，不可通过拷贝赋值
//svec = slist;
//元素类型相容，可采用范围赋值
svec.assign(slist.begin(), slist.end());
cout << svec.capacity() << " " << svec.size() << endl;
cout << svec[0] << endl;
return 0;
}
```

9.15

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector<int>ivec = { 1,2,3,4,5,6,7 };
    vector<int>ivec1 = { 1,2,3,4,5,6,7 };
    vector<int>ivec2 = { 1,2,3,4,5 };
    vector<int>ivec3 = { 1,2,3,4,5,6,7,8 };
    vector<int>ivec4 = { 1,2,3,4,5,7,6 };
    cout << (ivec == ivec1) << endl;
    cout << (ivec == ivec2) << endl;
    cout << (ivec == ivec3) << endl;
    cout << (ivec == ivec4) << endl;
    ivec.push_back(8);
    ivec.pop_back();
    cout << ivec1.capacity() << " " << ivec1.size() << endl;
    system("pause");
    return 0;
}
```

9.16

```
#include<iostream>
#include<list>
#include<vector>
using namespace std;
```

```
bool l_v_equal(vector<int>&ivec, list<int>&ilist)
{
    //比较list和vector元素个数
    if (ilist.size() != ivec.size())
        return false;
    auto lb = ilist.cbegin();//list首元素
    auto le = ilist.cend();//list尾后地址
    auto vb = ivec.cbegin();    //vector首元素
    for (; lb != le; lb++, vb++)
        if (*lb != *vb) //元素不等，容器不等
            return false;
    return true;//容器相等
}
int main()
{
    vector<int>ivec = {1, 2, 3, 4, 5, 6, 7};
    list<int>ilist = { 1,2,3,4,5,6,7 };
    list<int>ilist1 = { 1,2,3,4,5 };
    list<int>ilist2 = { 1,2,3,4,5,6,7 };
    list<int>ilist3 = { 1,2,3,4,5,6,7 };
    cout << l_v_equal(ivec, ilist) << endl;
    cout << l_v_equal(ivec, ilist1) << endl;
    cout << l_v_equal(ivec, ilist2) << endl;
    cout << l_v_equal(ivec, ilist3) << endl;
    system("pause");
    return 0;
}
```

9.18

```
#include<iostream>
#include<deque>
#include<string>
using namespace std;
int main()
{
    string str;
    deque<string> dstr;
    cout << "请输入：按Ctrl+z结束" << endl;
    while (cin >> str)
    {
        dstr.push_back(str);
    }
    //用cbegin()获取首元素迭代器，遍历deque中所有元素
    for (auto to = dstr.cbegin(); to != dstr.cend(); ++to)
    {
        cout << *to << endl;
    }
    system("pause");
    return 0;
```

```
}
```

9.19

```
#include<iostream>
#include<list>
#include<string>
using namespace std;
int main()
{
    string str;
    liast<string> lstr;
    cout << "请输入：按Ctrl+z结束" << endl;
    while (cin >> lstr)
    {
        dstr.push_back(lstr);
    }
    //用cbegin()获取首元素迭代器，遍历list中所有元素
    for (auto to = lstr.cbegin(); to != lstr.cend(); ++to)
    {
        cout << *to << endl;
    }
    system("pause");
    return 0;
}
```

9.20

```
#include<iostream>
#include<list>
#include<deque>

using namespace std;
int main()
{
    list<int> l_num = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    deque<int>d_odd, d_even;

    for (auto to = l_num.cbegin(); to != l_num.cend(); to++)
    {
        if (*to % 2)
        {
            d_even.push_back(*to);
        }
        else
        {
            d_odd.push_back(*to);
        }
    }
```

```
    cout << "ODD：" << endl;
    for (auto todd = d_odd.cbegin(); todd != d_odd.cend(); todd++)
    {
        cout << *todd << " "<<endl;
    }
    cout << "Even：" << endl;
    for (auto todd1 = d_even.cbegin(); todd1 != d_even.cend(); todd1++)
    {
        cout << *todd1<<" "<<endl;
    }

    system("pause");
    return 0;
}
```

9.22

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector<int>iv = { 1,1,2,1 };
    int some_val = 1;

    vector <int>::iterator iter = iv.begin();
    int org_size = iv.size(), new_ele = 0;//原大小和新元素个数

    /*vector<int> ::iterator iter = iv.begin(), mid = iv.begin() + iv.size() / 2;*/
    //while (iter != mid)
    //  if (*iter == some_val)
    //      iv.insert(iter, 2 * some_val);
    /*
        循环中未对iter进行递增操作，iter无法向中点推进
        即使加入iter++ 语句由于iv插入元素后，iter已经消失，iter++也不能起到将迭代器向前推进一个元素的作
用
    */
    while (iter != (iv.begin() + org_size / 2 + new_ele))
        if (*iter == some_val)
        {
            iv.insert(iter, 2 * some_val);//iter指向新元素
            new_ele++;
            iter++; iter++; //将iter推进到旧元素的下一个位置
        }
        else
            iter++;//简单推进iter
    for (iter = iv.begin(); iter != iv.end(); iter++)
        cout << *iter << endl;
    system("pause");
    return 0;
}
```

9.24

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector<int> v1 = { 1,2,3,4,5 };
    vector<int> v2;
    cout << "at:" << v2.at(0)<<endl;
    cout << "下标:" << v2[0]<<endl;
    cout << "front:" << v2.front()<<endl;
    int i = 0;
    cout << "begin():"<<*(v1.begin())<<endl;
    system("pause");
    return 0;
}
```

9.26

```
#include<iostream>
#include<vector>
#include<list>
#include<string>
using namespace std;
int main()
{
    vector<int> vec;
    list<int>lis;
    int ia[] = { 0,1,1,2,3,5,8,13,21,55,89 };
    cout << sizeof(ia) /sizeof(ia[0])<<endl ;

    for (int i =  0; i < sizeof(ia) / sizeof(ia[0]); ++i)
    {
        vec.push_back(ia[i]);
        lis.push_back(ia[i]);
    }

    for (auto to = vec.cbegin(); to != vec.cend(); to++)
    {
        if (*to % 2)
            to = vec.erase(to);
    }

    cout << "VEC:" << endl;
    for (auto to = vec.cbegin(); to != vec.cend(); to++)
    {
        cout << *to << endl;
    }
    for (auto to = lis.cbegin(); to != lis.cend(); to++)
    {
        if (*to % 2)
```

```
            ;
        else
            to = lis.erase(to);
    }
    cout << "LIS:" << endl;
    for (auto to = lis.cbegin(); to != lis.cend(); to++)
    {
        cout << *to << endl;
    }
    system("pause");
    return 0;
}
```

9.27

```
#include<iostream>
#include<vector>
#include<list>
#include<string>
#include<forward_list>
using namespace std;
int main()
{
    forward_list<int> flst = { 0,1,2,3,4,5,6,7,8,9 };
    auto prev = flst.before_begin();//表示flst的“首前元素”
    auto curr = flst.begin();//表示flst中的第一个元素
    while (curr != flst.end())
    {
        if (*curr & 1)
        {
            curr = flst.erase_after(prev);//删除它并移动curr
        }
        else
        {
            prev = curr;//移动迭代器curr，指向下一个元素，prev指向
            ++curr;//curr之前的元素
        }
    }
    for (curr = flst.begin(); curr != flst.end(); curr++)
    {
        cout << *curr << endl;
    }
    system("pause");
    return 0;
}
```

9.28

```
#include<iostream>
#include<vector>
#include<list>
```

```
#include<string>
#include<forward_list>
using namespace std;
void pushString(forward_list<string> &f1, const string  &str1, const string  &str2)
{
    auto prev = f1.before_begin();
    auto curr = f1.begin();
    bool isfind = false;
    while (curr != f1.end())
    {
        if (*curr == str1)
        {
            curr = f1.insert_after(curr, str2);
            isfind = true;
        }
        prev = curr;//前驱迭代器向前推进
        curr++;
    }
    if (!isfind)
        f1.insert_after(prev, str2);//未找到给定字符串，插入尾后
}
int main()
{
    forward_list<string> flst = { "11","12","13" };
    string str1 = "11", str2 = "15";
    pushString(flst, str1, str2);
    pushString(flst, "11", "15");
    for (auto curr = flst.cbegin(); curr != flst.cend(); curr++)
    {
        cout << *curr << endl;
    }
    system("pause");
    return 0;
}
```

9.31

```
#include<iostream>
#include<vector>
#include<list>
#include<string>
using namespace std;
int main()
{
    list<int>ilst = { 0,1,2,3,4,5,6,7 };
    auto curr = ilst.begin();   //首节点
    while (curr != ilst.end())
    {
        if (*curr & 1)  //奇数
        {
            curr = ilst.insert(curr, *curr);    //插入到当前元素之前
        curr++; curr++;         //移动到下一元素
```

```
        }
        else
        {
            curr = ilst.erase(curr);
        }
    }
    for (curr = ilst.begin(); curr != ilst.end(); curr++)
    {
        cout << *curr << " ";
    }
    cout << endl;
    system("pause");
    return 0;
}
//////////////第二种//////////////////////////////////
#include<iostream>
#include<vector>
#include<list>
#include<string>
#include<forward_list>
using namespace std;
int main()
{
    forward_list<int>ifst = { 0,1,2,3,4,5,6,7 };
    auto prev = ifst.before_begin();
    auto curr = ifst.begin();   //首节点
    while (curr != ifst.end())
    {
        if (*curr & 1)  //奇数
        {
            curr = ifst.insert_after(curr, *curr);  //插入到当前元素之前
            prev = curr;    //prev移动到新插入元素
             curr++;            //移动到下一元素
        }
        else
        {
            curr = ifst.erase_after(prev);//删除，curr指向下一元素
        }
    }
    for (curr = ifst.begin(); curr != ifst.end(); curr++)
    {
        cout << *curr << " ";
    }
    cout << endl;
    system("pause");
    return 0;
}
```

9.34

```
#include<iostream>
```

```
#include<vector>
#include<list>
#include<string>
using namespace std;
int main()
{
    vector<int> v1 = { 1,2,3,4,5,6,7,8,9 };
    auto iter = v1.begin();
    string temp;
    while (iter != v1.end())
    {
        if (*iter % 2)
            iter = v1.insert(iter, *iter);
        ++iter;
        for (auto begin = v1.begin(); begin != v1.end(); begin++)
            cout << *begin << " ";
        cout << endl;
        cin >> temp;
    }
    system("pause");
    return 0;
}
```

9.41

```
#include<iostream>
#include<vector>
#include<list>
#include<string>
using namespace std;
int main()
{
    vector<char> ch = { 'a','b','c' };
    string s(ch.begin(), ch.end());
    string s2(ch.data(), ch.size());
    cout << s << endl;
    cout << s2 << endl;
    system("pause");
    return 0;
}
```

9.42

```
#include<iostream>
#include<vector>
#include<list>
#include<string>
using namespace std;
void input_string(string &str)
{
    str.reserve(100);
```

```
    char c;
    while(cin >> c)
    {
        str.push_back(c);
    }
}
int main()
{
    string str;
    input_string(str);
    cout << str << endl;
    system("pause");
    return 0;
}
```

9.43、9.44

```
#include<iostream>
#include<vector>
#include<list>
#include<string>
using namespace std;

void replace(string &s, const string &oldVal, const string &newVal)
{
    int p = 0;
    while((p = s.find(oldVal, p) != string::npos))      //在s中查找oldVal
    {
        s.replace(p, oldVal.size(), newVal);    //将找到的字串替换为newVald的内容
        p += newVal.size();     //下标调整到新插入的内容
    }
    auto l = oldVal.size();
    if (!l)
    {
        return;
    }
    auto curr = s.begin();
    while (curr <= s.end() - l)//末尾少于oldVal长度的部分无需检查
    {
        auto curr1 = curr;
        auto curr2 = oldVal.begin();
        //s中curr开始的字串必须每个字符都与oldVal相同
        while (curr2 != oldVal.end() && *curr1 == *curr2)
        {
            curr1++;
            curr2++;
        }
        if (curr2 == oldVal.end())  //oldVal耗尽--字符串相等
        {
            curr = s.erase(curr, curr1);    //删除s中与oldVal相等部分
            if (newVal.size())
            {
```

```
                curr2 = newVal.end();
                do
                {
                    curr2--;
                    curr = s.insert(curr, *curr2);
                } while (curr2 > newVal.begin());
            }
            curr += newVal.size();  //迭代器移动到新插入内容之后
        }
        else
        {
            curr++;
        }
    }
}
int main()
{
    string s = "tho thru rho!";
    replace(s, "thru", "through");
    cout << s << endl;
    replace(s, "tho", "through");
    cout << s << endl;
    replace(s, "through", "");
    cout << s << endl;
    system("pause");
    return 0;
}
```

9.45、9.46

```
#include<iostream>
#include<vector>
#include<list>
#include<string>
using namespace std;
void name_string(string &name, const string prefix, const string suffix)
{
    name.insert(name.begin(), 1, ' ');
    name.insert(name.begin(), prefix.begin(), prefix.end());//  插入前缀
    name.append("");
    name.append(suffix.begin(), suffix.end());//插入后缀

    name.insert(0, " ");
    name.insert(0, prefix);
    name.insert(name.size(), " ");
    name.insert(name.size(), suffix);
}

int main()
{
    string s = "哈哈";
    name_string(s, "Mr.", "||");
```

```
    cout << s << endl;
    system("pause");
    return 0;
}
```

9.47

```
#include<iostream>
#include<vector>
#include<list>
#include<string>
using namespace std;
void find_char(string &s, const string &chars)
{
    cout << "在" << s << "中查找" << chars << "中字符" << endl;
    string::size_type pos = 0;
    while ((pos = s.find_first_of(chars, pos)) != string::npos)
    {
        cout << "pos:" << pos << ",char:" << s[pos] << endl;
        pos++;//移动到下一个字符
    }
}
int main()
{
    string s = "ab2c3d7E4E6";
    cout << "查找所以数字" << endl;
    find_char(s, "0123456789");
    cout << "查找所以字母" << endl;
    find_char(s, "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ");
    system("pause");
    return 0;
}
```

第二种形式：找数字则只有字母。找字母则只有数字

```
#include<iostream>
#include<vector>
#include<list>
#include<string>
using namespace std;
void find_not_char(string &s, const string &chars)
{
    cout << "在" << s << "中查找不在" << chars << "中字符" << endl;
    string::size_type pos = 0;
    while ((pos = s.find_first_not_of(chars, pos)) != string::npos)
    {
        cout << "pos:" << pos << ",char:" << s[pos] << endl;
        pos++;//移动到下一个字符
    }
}
int main()
{
```

```
    string s = "ab2c3d7E4E6";
    cout << "查找所以数字" << endl;
    find_char(s, "0123456789");
    cout << "查找所以字母" << endl;
    find_char(s, "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ");
    system("pause");
    return 0;
}
```

9.49

```
#include<iostream>
#include<vector>
#include<list>
#include<string>
#include<fstream>
using namespace std;

void find_longest_word(ifstream &in)
{
    string s, longest_word;
    int max_length = 0;
    while (in >> s)
    {
        if (s.find_first_of("bdfghjklpqty") != string::npos)
            continue;       //包含上出头和下出头字母
        cout << s << " ";
        if (max_length < s.size())  //新单词更长
        {
            max_length = s.size();      //记录长度
            longest_word = s;
        }
    }
    cout << endl << "最长字符串：" << longest_word << endl;
}
int main(int argc,char *argv[])
{
    ifstream in(argv[1]);//打开文件
    if (!in)
    {
        cerr << "无法打开文件" << endl;
        return -1;
    }
    find_longest_word(in);
    system("pause");
    return 0;
}
```

9.50

```
#include<iostream>
#include<vector>
#include<list>
#include<string>
using namespace std;
int main()
{
    vector<string>vs = { "123","+456","-789" };
    int sum = 0;
    for (auto iter = vs.begin(); iter != vs.end(); iter++)
        sum += stoi(*iter);
    cout << "和：" << sum << endl;
    system("pause");
    return 0;
}
```

9.51

- 题目：设计一个类，他有三个unsigned成员，分别表示年、月、日。为其编写函数，接受一个表示日期string参数，你构造函数应该能处理不同数据格式，如january 1,1990、1/1/1900,jan 1 1900等
- 具体算法如下
    1. 若首字符是数字，则为格式2，用stoi提取月份值，若月份不合法，抛出异常，否则转到步骤6
    2. 若首字符不是数组，表明是格式1或者3，首先提取月份值。
    3. 将ds开始的字串与月份简称进行比较，若均不等，抛出异常（若与简称不等、则不可能与全称相等）。
    4. 若与第i个月简称相等，且下一个字符是合法间隔字符，返回月份值。
    5. 否则，检查接下来的字串是否与全称剩余部分相等，若不等，抛出异常，否则，返回月值。
    6. 用stoi提取月份值和年份值，如需要，检查间隔符合法性。

```
////代码复杂看不懂
```

9,52

- 题目：使用stack处理符号化表达式，当你看到一个左括号，将其记录下来，当你在一个左括号之后看到一个右括号，从stack中pop对象，知道遇到左括号，将左括号也一起弹出栈，然后将一个值（括号内的运算结果）push到栈中，表示一个括号的（字）表达式已经处理完毕，被其运算结果所替代。
- 算法步骤如下
    1. 读入一个运算v。

        a. 若栈空或栈顶是左括号，则v是第一个运算数，直接压栈

        b.否则，v前必须是一个运算符，再之前是另一个运算数v，从栈顶弹出这两项，将计算结果压栈即可，否则，就抛出一个“缺少运算符”异常。
    2. 读入了一个左括号，直接压栈
    3. 读入了一个运算符，

a.若栈空或栈顶不是一个运算符，则抛出一个“缺少运算数”异常，注意：若运算符之前是一个右括号，之前也已经处理完毕，栈顶是其计算结果，仍应该是运算数，不影响逻辑

b.否则，运算符压栈

4. 读入了一个右括号，

   a.若栈空，表明之前没有与之配对的左括号，抛出“未匹配右括号”异常。

   b.若栈顶不顺运算数，表明括号内缺少一个运算数，抛出一个异常。

   c.若栈顶不是运算数，表明括号内缺少一个运算数，抛出一个异常

   d.弹出此运算数v，若栈空或栈顶不是左括号，仍抛出“为匹配右括号”异常，否则弹出左括号，把v作为新运算数，执行1中的逻辑。

5. 以上均不是，则出现了非法输入，会在转换为数值是产生异常。
6. 当字符串处理完毕后，判断栈中是否有且只有一个运算数，若是，此值即为表达式运算结果，输出它：否则，表达式非法

-

第十章、泛型算法

10.1

```
```