面向对象编程基础

本课程入选教育部产学合作协同育人项目
课程主页:**http://cpp.njuer.org**
课程老师:陈明 **http://cv.mchen.org**

```
ppt和代码下载地址
git clone https://gitee.com/cpp-njuer-org/book
```

第5章

语句

- 简单语句
- 语句作用域
- 条件语句
- 迭代语句
- 跳转语句
- try语句块和异常处理

简单语句

- **表达式语句**：一个表达式末尾加上分号，就变成了表达式语句。
    - 作用是执行表达式并丢弃求值结果

```
ival+5;//没用的表达式语句
cout << ival;//有用的表达式语句
```

简单语句

- **空语句**：只有一个单独的分号。
  - 使用空语句应该加上注释，从而让读代码的人知道这条语句是有意省略的。
  - 别漏写分号，也别多写分号。
  - 语法上需要一条语句，逻辑上不需要时，使用空语句。

```
;//空语句
while(cin>>s&&s!=sought)
    ;//空语句
ival = v1+v2;;//正确,第二个分号是空多余的语句
//额外的分号导致死循环
while(iter!=svec.end());//多了的;是while循环体
    ++iter;                 //不属于循环
//多余的空语句并非总是无害的
```

简单语句

- **复合语句（块）**：用花括号 {} 括起来的语句和声明的序列。一个块就是一个作用域。
  - 在块内引入的名字只能在块内部以及嵌套在块中的子块访问。
  - 语法上需要一条语句，逻辑上需要多条语句，则应使用复合语句。

```
while(val<=10){
    sum+=val;
    ++val;
}
while(cin>>s&&s!=sought)
    {}//空块
//块不以分号结束
//空块，内部没有任何语句的一对花括号。作用等价于空语句。
```

练习

```
// 什么是空语句？什么时候会用到空语句？

//只含义一个单独的分号的语句是空语句。如;
//如果在程序的某个地方，语法上需要一条语句但是逻辑上不需要，
//此时应该使用空语句。
while (cin >> s && s != sought)
    ;
```

练习

```
// 什么是块？什么时候会用到块？

//用花括号括起来的语句和声明的序列就是块。
{
    // ...
}
//如果在程序的某个地方，语法上需要一条语句，
//而逻辑上需要多条语句，此时应该使用块
while(val<=10){
    sum+=val;
    ++val;
}
```

练习

```
//使用逗号运算符重写while循环，使它不再需要块，
//观察改写之后的代码可读性提高了还是降低了。

while (val <= 10)
    sum += val, ++val;
//代码的可读性降低
```

语句作用域

```
//可以在if switch  while for语句的控制结构内定义变量，
//定义在控制结构当中的变量，
//只在相应语句内部可见.
while(int i=get_num()){
    cout<<i<<endl;
}
i = 0;//错误：循环外无法访问i

//如果其它代码也要访问，则变量必须定义在语句外部。
auto beg = v.begin();
while (beg!=v.end()&&*beg>=0){
    ++beg;
}
if(beg==v.end()){
    //...
}
```

练习

```
//说明下列例子的含义，如果存在问题，试着修改它。

(a) while (string::iterator iter != s.end()) { /* . . . */ }

(b) while (bool status = find(word)) { /* . . . */ }
        if (!status) { /* . . . */ }
```

- (a) 这个循环试图用迭代器遍历string，但是变量的定义应该放在循环的外面，目前每次循环都会重新定义一个变量，而且没初始化，明显是错误的。
- (b) 这个循环的while和if是两个独立的语句，if语句中无法访问status变量，正确的做法是应该将if语句包含在while里面。

条件语句

- if 语句
    - 根据条件决定控制流
- switch 语句
    - 计算一个整型表达式的值，根据值从几条执行路径中选择一条

IF语句

- 作用是：判断一个指定条件是否为真，根据结果决定是否指定另外一条语句
- 两种形式

```
if(condition)
    statement
```

```
if(condition)
    statement
else
    statement2
```

```
//condition必须用圆括号包起来，类型必须能转换成布尔类型
//statement statement2 通常时块语句
```

使用IF ELSE 语句

```
//ifelse.cpp
#include <iostream>
using std::endl; using std::cin; using std::cout;
#include <vector>
using std::vector;
#include <string>
using std::string;
#include <iterator>
using std::begin; using std::end;

const vector<string> scores = {"F", "D", "C", "B", "A", "A++"};
vector<unsigned> grades;
int main()
{
    // read a set of scores from the input
    unsigned grade;
    while (cin >> grade)
        grades.push_back(grade);
```

```
    // now process those grades
    for (auto it : grades) {   // for each grade we read
        cout << it << " " ;    // print the grade
        string lettergrade;    // hold coresponding letter grade
        // if failing grade, no need to check for a plus or minus
        if (it < 60)
            lettergrade = scores[0];
        else {
            lettergrade = scores[(it - 50)/10];  // fetch the letter grade
            if (it != 100)  // add plus or minus only if not already an A++
                if (it % 10 > 7)
                    lettergrade += '+';   // grades ending in 8 or 9 get a +
                else if (it % 10 < 3)
                    lettergrade += '-';   // grades ending in 0, 1, or 2 get a -
        }
        cout << lettergrade << endl;
   }

    return 0;
}
```

- 嵌套`if else` 语句
- 注意使用花括号
    - 使用缩进，代码看起来正确。使用花括号，避免混淆不清。
- 悬垂`else:`用来描述在嵌套的`if else`语句中，如果`if`比`else`多时如何处理的问题。
    - `else`匹配最近没有配对的`if`
- 使用花括号控制执行路径

练习

```
//写一段程序，使用if else语句实现把数字转换为字母成绩的要求。
#include <iostream>
#include <vector>
#include <string>
using std::vector; using std::string; using std::cout; using std::endl; using std::cin;

int main()
{
    vector<string> scores = { "F", "D", "C", "B", "A", "A++" };
    for (int g; cin >> g;)
    {
        string letter;
        if (g < 60)
        {
            letter = scores[0];
        }
```

```
        else
        {
            letter = scores[(g - 50) / 10];
            if (g != 100)
                letter += g % 10 > 7 ? "+" : g % 10 < 3 ? "-" : "";
        }
        cout << letter << endl;
    }

    return 0;
}
```

练习

```
//改写上一题的程序，使用条件运算符代替if else语句。
#include <iostream>
#include <vector>
#include <string>
using std::vector; using std::string; using std::cout; using std::endl; using std::cin;
int main()
{
    vector<string> scores = { "F", "D", "C", "B", "A", "A++" };
    int grade = 0;
    while (cin >> grade)
    {
        string lettergrade = grade < 60 ?
                             scores[0] : scores[(grade - 50) / 10];
        lettergrade += (grade == 100 || grade < 60) ? "" : (grade % 10 > 7) ? "+" :
                             (grade % 10 < 3) ? "-" : "";
        cout << lettergrade << endl;
    }
    return 0;
}
```

练习

```
//改写下列代码段中的错误。
(a) if (ival1 != ival2)
        ival1 = ival2
    else
        ival1 = ival2 = 0;
(b) if (ival < minval)
        minval = ival;
        occurs = 1;
(c) if (int ival = get_value())
        cout << "ival = " << ival << endl;
    if (!ival)
        cout << "ival = 0\n";
(d) if (ival = 0)
        ival = get_value();
```

- (a) ival1 = ival2 后面少了分号。
- (b) 应该用花括号括起来。
- (c) if (!ival) 应该改为 else。
- (d) if (ival = 0) 应该改为 if (ival == 0)。

练习

```
//什么是悬垂else？C++语言是如何处理else子句的？

//用来描述在嵌套的if else语句中，如果if比else多时如何处理的问题。
//C++使用的方法是else匹配最近没有配对的if。
```

SWITCH 语句

- 提供了一条便利的途径，使得我们能够在若干固定选项中作出选择。

```cpp
//count_aeiou.cpp
#include <iostream>
using std::cin; using std::cout; using std::endl;

int main()
{
    // initialize counters for each vowel
    unsigned aCnt = 0, eCnt = 0, iCnt = 0, oCnt = 0, uCnt = 0;

    char ch;
```

```
while (cin >> ch) {
    // if ch is a vowel, increment the appropriate counter
    switch (ch) {
        case 'a':
            ++aCnt;
            break;
        case 'e':
            ++eCnt;
            break;
        case 'i':
            ++iCnt;
            break;
        case 'o':
            ++oCnt;
            break;
        case 'u':
            ++uCnt;
            break;
    }
}
```

```
    // print results
    cout << "Number of vowel a: \t" << aCnt << '\n'
         << "Number of vowel e: \t" << eCnt << '\n'
         << "Number of vowel i: \t" << iCnt << '\n'
         << "Number of vowel o: \t" << oCnt << '\n'
         << "Number of vowel u: \t" << uCnt << endl;

    return 0;
}
```

- case关键字和它对应的值一起被称为case标签，必须是整型常量表达式。

```
char ch=getVal();
int ival=42;
switch(ch){
case 3.14://错误 标签不是一个整数
case ival;//错误 标签不是一个常量
}
```

SWITCH 内部的控制流

```cpp
//count_aeiou2.cpp
#include <iostream>
using std::cin; using std::cout; using std::endl;

int main()
{
    char ch;
    // initialize counters for each vowel
    unsigned vowelCnt = 0;
    unsigned otherCnt = 0;  // count anything that isn't a vowel
```

```
    while (cin >> ch) {
        // if ch is a vowel, increment the appropriate counter
        switch (ch) {
        //case标签不一定非得换行。可写在一行
        //不要省略break语句，若无break，最好加注释解释逻辑
            case 'a': case 'e': case 'i': case 'o': case 'u':
                ++vowelCnt;
                break;
            default:
                ++otherCnt;
                break;
        }
    }
    // print results
    cout << "Number of vowels: \t" << vowelCnt << '\n'
         << "Total non-vowels : \t" << otherCnt << '\n';

    return 0;
}
```

漏写BREAK容易引发缺陷

```
//不正确的逻辑
    while (cin >> ch) {
        // if ch is a vowel, increment the appropriate counter
        switch (ch) {
            case 'a':
                ++aCnt;
            case 'e':
                ++eCnt;
            case 'i':
                ++iCnt;
            case 'o':
                ++oCnt;
            case 'u':
                ++uCnt;
        }
    }
```

DEFAULT 标签

- 如果没有一个case标签能匹配上switch表达式的值，程序将执行紧跟在default标签后的语句。

```
switch (ch) {
    case 'a': case 'e': case 'i': case 'o': case 'u':
        ++vowelCnt;
        break;
    default:
        ++otherCnt;
        break;
}
//即使不准备在default下做任何工作，定义一个default标签也是有用的
//告诉读者已经考虑了默认情况
```

SWITCH 内部的变量定义

- c++语言规定，不允许跨过变量的初始化语句直接跳转到该变量作用域的另一位置。

```
//不合法
case true:
    string file_name;
    int ival=0;
    int jval;
    break;
case false:
    jval = next_num();
    if(file_name.empty())
        //...
```

```
case true:
{
    //ok
    string file_name=get_file_name();
    break;
}
case false:
    if(file_name.empty())//错误 file_name 不在作用域
```

练习

```
//编写一段程序，使用一系列if语句统计从cin读入的文本中有多少元音字母。
#include <iostream>
using std::cout; using std::endl; using std::cin;
int main(){
    unsigned aCnt = 0, eCnt = 0, iCnt = 0, oCnt = 0, uCnt = 0;
    char ch;
    while (cin >> ch){
        if (ch == 'a') ++aCnt;
        else if (ch == 'e') ++eCnt;
        else if (ch == 'i') ++iCnt;
        else if (ch == 'o') ++oCnt;
        else if (ch == 'u') ++uCnt;
    }
    cout << "Number of vowel a: \t" << aCnt << '\n'
        << "Number of vowel e: \t" << eCnt << '\n'
        << "Number of vowel i: \t" << iCnt << '\n'
        << "Number of vowel o: \t" << oCnt << '\n'
        << "Number of vowel u: \t" << uCnt << endl;
    return 0;
}
```

练习

```
//写一段程序，既统计元音字母的小写形式，也统计元音字母的大写形式
#include <iostream>
using std::cin; using std::cout; using std::endl;
int main(){
    unsigned aCnt = 0, eCnt = 0, iCnt = 0, oCnt = 0, uCnt = 0;
    char ch;
    while (cin >> ch)
        switch (ch){
        case 'a':case 'A':++aCnt;break;
        case 'e':case 'E':++eCnt;break;
        case 'i':case 'I':++iCnt;break;
        case 'o':case 'O':++oCnt;break;
        case 'u':case 'U':++uCnt;break;}
    cout << "Number of vowel a(A): \t" << aCnt << '\n'
        << "Number of vowel e(E): \t" << eCnt << '\n'
        << "Number of vowel i(I): \t" << iCnt << '\n'
        << "Number of vowel o(O): \t" << oCnt << '\n'
        << "Number of vowel u(U): \t" << uCnt << endl;
    return 0;
}
```

练习

```
//修改统计元音字母的程序，使其也能统计空格、制表符、和换行符的数量。
#include <iostream>
using std::cin; using std::cout; using std::endl;
int main(){
    unsigned aCnt = 0, eCnt = 0, iCnt = 0, oCnt = 0, uCnt = 0,
    spaceCnt = 0, tabCnt = 0, newLineCnt = 0;char ch;
    while (cin >> std::noskipws >> ch)  //noskipws(no skip whitespce)
        switch (ch){
        case 'a':case 'A':++aCnt;break;
        case 'e':case 'E':++eCnt;break;
        case 'i':case 'I':++iCnt;break;
        case 'o':case 'O':++oCnt;break;
        case 'u':case 'U':++uCnt;break;
        case ' ':++spaceCnt;break;
        case '\t':++tabCnt;break;
        case '\n':++newLineCnt;break;}
```

```
    cout << "Number of vowel a(A): \t" << aCnt << '\n'
        << "Number of vowel e(E): \t" << eCnt << '\n'
        << "Number of vowel i(I): \t" << iCnt << '\n'
        << "Number of vowel o(O): \t" << oCnt << '\n'
        << "Number of vowel u(U): \t" << uCnt << '\n'
        << "Number of space: \t" << spaceCnt << '\n'
        << "Number of tab char: \t" << tabCnt << '\n'
        << "Number of new line: \t" << newLineCnt << endl;
    return 0;
}
//其中，使用 std::noskipws可以保留默认跳过的空格。
```

```cpp
//修改统计元音字母的程序，使其能统计含以下两个字符的字符序列的数量：ff、fl和fi。
#include <iostream>

using std::cin; using std::cout; using std::endl;

int main()
{
    unsigned aCnt = 0, eCnt = 0, iCnt = 0, oCnt = 0, uCnt = 0, spaceCnt = 0, tabCnt = 0,
    char ch, prech = '\0';
    while (cin >> std::noskipws >> ch)
    {
```

```
        switch (ch)
        {
        case 'a':
        case 'A':
            ++aCnt;
            break;
        case 'e':
        case 'E':
            ++eCnt;
            break;
```

```
        case 'i':
            if (prech == 'f') ++fiCnt;
        case 'I':
            ++iCnt;
            break;
        case 'o':
        case 'O':
            ++oCnt;
            break;
        case 'u':
        case 'U':
            ++uCnt;
            break;
        case ' ':
            ++spaceCnt;
            break;
        case '\t':
            ++tabCnt;
            break;
```

```
        case '\n':
            ++newLineCnt;
            break;
        case 'f':
            if (prech == 'f') ++ffCnt;
            break;
        case 'l':
            if (prech == 'f') ++flCnt;
            break;
        }
        prech = ch;
```

```
    }

    cout << "Number of vowel a(A): \t" << aCnt << '\n'
        << "Number of vowel e(E): \t" << eCnt << '\n'
        << "Number of vowel i(I): \t" << iCnt << '\n'
        << "Number of vowel o(O): \t" << oCnt << '\n'
        << "Number of vowel u(U): \t" << uCnt << '\n'
        << "Number of space: \t" << spaceCnt << '\n'
        << "Number of tab char: \t" << tabCnt << '\n'
        << "Number of new line: \t" << newLineCnt << '\n'
        << "Number of ff: \t" << ffCnt << '\n'
        << "Number of fl: \t" << flCnt << '\n'
        << "Number of fi: \t" << fiCnt << endl;

    return 0;
}
```

```
//下面显示的每个程序都含有一个常见的编码错误，指出错误在哪里，然后修改它们。
(a) unsigned aCnt = 0, eCnt = 0, iouCnt = 0;
    char ch = next_text();
    switch (ch) {
        case 'a': aCnt++;
        case 'e': eCnt++;
        default: iouCnt++;
   }
```

(a) 少了break语句。应该为：

```
unsigned aCnt = 0, eCnt = 0, iouCnt = 0;
char ch = next_text();
switch (ch) {
    case 'a': aCnt++; break;
    case 'e': eCnt++; break;
    default: iouCnt++; break;
}
```

```
(b) unsigned index = some_value();
    switch (index) {
        case 1:
            int ix = get_value();
            ivec[ ix ] = index;
            break;
        default:
            ix = ivec.size()-1;
            ivec[ ix ] = index;  }
```

(b) 在default分支当中，ix未定义。应该在外部定义ix。

```
unsigned index = some_value();
int ix;
switch (index) {
    case 1:
        ix = get_value();
        ivec[ ix ] = index;
        break;
    default:
        ix = static_cast<int>(ivec.size())-1;
        ivec[ ix ] = index;}
```

```
(c) unsigned evenCnt = 0, oddCnt = 0;
    int digit = get_num() % 10;
    switch (digit) {
        case 1, 3, 5, 7, 9:
            oddcnt++;
            break;
        case 2, 4, 6, 8, 10:
            evencnt++;
            break;}
```

(c) case后面应该用冒号而不是逗号。

```
unsigned evenCnt = 0, oddCnt = 0;
int digit = get_num() % 10;
switch (digit) {
    case 1: case 3: case 5: case 7: case 9:
        oddcnt++;
        break;
    case 2: case 4: case 6: case 8: case 0:
        evencnt++;
        break;}
```

```
(d) unsigned ival=512, jval=1024, kval=4096;
    unsigned bufsize;
    unsigned swt = get_bufCnt();
    switch(swt) {
        case ival:
            bufsize = ival * sizeof(int);
            break;
        case jval:
            bufsize = jval * sizeof(int);
            break;
        case kval:
            bufsize = kval * sizeof(int);
            break;
    }
```

(d) case标签必须是整型常量表达式。

```
const unsigned ival=512, jval=1024, kval=4096;
    unsigned bufsize;
    unsigned swt = get_bufCnt();
    switch(swt) {
        case ival:
            bufsize = ival * sizeof(int);
            break;
        case jval:
            bufsize = jval * sizeof(int);
            break;
        case kval:
            bufsize = kval * sizeof(int);
            break;
    }
```

迭代语句

- 迭代语句通常称为循环，重复执行操作直到满足某个条件才停下来。
    - while for在执行循环体前检查条件
    - do while 先执行循环体，再检查条件

WHILE 语句

- 只要条件为真，while语句就重复执行循环
- 条件部分可以是一个表达式或带初始化的变量声明
    - 条件本身或循环体设法改变表达式的值，否则可能死循环

```
while(condition)
    statement
```

- **while**：当不确定到底要迭代多少次时，使用 while循环比较合适，比如读取输入的内容。

使用WHILE 循环

```
vector<int> v;
int i;
while(cin>>i){
    v.push_back(i);
}
auto beg = v.begin();
while(beg!=v.end()&&*beg>=0)
    ++beg;
if(beg==v.end())
    //此时v中所有元素大于等于0
```

练习

```
//编写一段程序，从标准输入中读取若干string对象并查找连续重复出现的单词，
//所谓连续重复出现的意思是：一个单词后面紧跟着这个单词本身。要求记录连续
//重复出现的最大次数以及对应的单词。如果这样的单词存在，输出重复出现的
//最大次数；如果不存在，输出一条信息说明任何单词都没有连续出现过。
//例如：如果输入是：
//how now now now brown cow cow
//那么输出应该表明单词now连续出现了3次。

#include <iostream>
#include <string>

using std::cout; using std::cin; using std::endl;
using std::string; using std::pair;
```

```cpp
int main()
{
    pair<string, int> max_duplicated;
    int count = 0;
    for (string str, prestr; cin >> str; prestr = str)
    {
        if (str == prestr) ++count;
            else count = 0;
        if (count > max_duplicated.second) max_duplicated = { prestr, count };
    }
    if (max_duplicated.first.empty()) cout << "There's no duplicated string." << endl;
    else cout << "the word " << max_duplicated.first << " occurred " << max_duplicated.se
    return 0;
}
```

传统的FOR语句

- 语法for (init-statement; condition; expression) statement
    - init-statement 必须是三种形式的一种：声明语句、表达式语句、空语句。
    - 也可看做for (initializer; condition; expression) statement

传统FOR循环的执行流程

- for (init-statement; condition; expression) statement
    1. 循环开始，执行一次 init-statement
    2. 判断 condition 条件为假终止
    3. 条件为真，执行循环体
    4. 执行expression .重复执行2.3.4直到条件为假终止

FOR语句头中的多重定义

- init-statement 可以定义多个对象，只能有一条声明语句
- 变量基础类型相同

省略FOR语句头的某些部分

- **for**： for语句可以省略掉 init-statement， condition和 expression的任何一个；**甚至全部。**
    - 如无需初始化，空语句作为init-statement
    - 省略condition，等价于条件部分写一个true.循环体必须有语句负责退出循环
    - 省略expression,则条件部分或循环体改变变量的值

练习

```
//说明下列循环的含义并改正其中的错误。
(a) for (int ix = 0; ix != sz; ++ix) { /* ... */ }
    if (ix != sz)
        // . . .
(b) int ix;
    for (ix != sz; ++ix) { /* ... */ }
(c) for (int ix = 0; ix != sz; ++ix, ++sz) { /*...*/ }
```

```
(a) int ix;
    for (ix = 0; ix != sz; ++ix)  { /* ... */ }
    if (ix != sz)
    // . . .
(b) int ix;
    for (; ix != sz; ++ix) { /* ... */ }
(c) for (int ix = 0; ix != sz; ++ix) { /*...*/ }
```

练习

```
//while循环特别适用于那种条件不变、反复执行操作的情况，
//例如，当未达到文件末尾时不断读取下一个值。
//for循环更像是在按步骤迭代，它的索引值在某个范围内一次变化。
//根据每种循环的习惯各自编写一段程序，然后分别用另一种循环改写。
int i;
while ( cin >> i )
    // ...
for (int i = 0; cin >> i;)
    // ...

for (int i = 0; i != size; ++i)
    // ...
int i = 0;
while (i != size)
{
    // ...
    ++i;
}
```

练习

```
//假设有两个包含整数的vector对象，编写一段程序，
//检验其中一个vector对象是否是另一个的前缀。
//为了实现这一目标，对于两个不等长的vector对象，
//只需挑出长度较短的那个，把它的所有元素和另一个vector对象比较即可。
//例如，如果两个vector对象的元素分别是0、1、1、2 和 0、1、1、2、3、5、8，
//则程序的返回结果为真。
#include <iostream>
#include <vector>
using std::cout; using std::vector;
bool is_prefix(vector<int> const& lhs, vector<int> const& rhs)
{
    if(lhs.size() > rhs.size())
        return is_prefix(rhs, lhs);
    for(unsigned i = 0; i != lhs.size(); ++i)
        if(lhs[i] != rhs[i]) return false;
    return true;
}
```

练习

```
int main()
{
    vector<int> l{ 0, 1, 1, 2 };
    vector<int> r{ 0, 1, 1, 2, 3, 5, 8 };
    cout << (is_prefix(r, l) ? "yes\n" : "no\n");
    return 0;
}
```

范围FOR语句

- c++11引入的一种更简单的for语句，可以遍历容器或其它序列的所有元素
- 语法形式为

```
for(declaration:expression)
    statement
```

- expressionn 必须是一个序列，如花括号初始值列表、数组、vector、string
共同特点是能返回迭代器的begin和end成员
- declaration定义一个变量，可以用auto类型说明符。
如需要写操作，循环变量必须声明成引用类型。

范围FOR语句

```
vector<int> v={0,1,2,3,4,5,6,7,8,9}
for(auto &r:v){
    r*=2;
}
//等价于
for(auto beg=v.begin(),end=v.en();beg!=end;++beg){
    auto &r=*beg;
    r*=2;
}
//不能通过范围for语句增加vector对象的元素
```

DO WHILE语句

- 先执行循环体，后检查条件，至少循环一次
- 语法如下

```
do
    statement
while(condition);
//在圆括号包起的条件后面，分号表示语句结束。
```

DO WHILE语句

```cpp
//dowhile.cpp
#include <iostream>
using std::cin; using std::cout; using std::endl;
#include <string>
using std::string;
int main(){
    // repeatedly ask the user for a pair of numbers to sum
    string rsp;  // used in the condition; can't be defined inside the do
    do {
        cout << "please enter two values: ";
        int val1 = 0, val2 = 0;
        cin  >> val1 >> val2;
        cout << "The sum of " << val1 << " and " << val2
             << " = " << val1 + val2 << "\n\n"
             << "More? Enter yes or no: ";
        cin  >> rsp;
    } while (!rsp.empty() && rsp[0] != 'n');
    cout << endl;
    return 0;
}
```

DO WHILE语句

- 不允许在条件部分定义变量

```
do{
    //...
    mumble(foo);
}while(int foo=get_foo());//错误，将变量声明放在了d
```

练习

```
//说明下列循环的含义并改正其中的错误。
(a) do
        int v1, v2;
        cout << "Please enter two numbers to sum:" ;
        if (cin >> v1 >> v2)
            cout << "Sum is: " << v1 + v2 << endl;
    while (cin);
(b)
    do {
        // . . .
    } while (int ival = get_response());
(c) do {
        int ival = get_response();
    } while (ival);
```

练习

```
(a) do { // 应该添加花括号
        int v1, v2;
        cout << "Please enter two numbers to sum:" ;
        if (cin >> v1 >> v2)
            cout << "Sum is: " << v1 + v2 << endl;
    }while (cin);
(b) int ival;
    do {
        // . . .
    } while (ival = get_response()); // 应该将ival 定义在循环外
(c) int ival = get_response();
    do {
        ival = get_response();
    } while (ival); // 应该将ival 定义在循环外
```

练习

```cpp
//编写一段程序，使用do while循环重复地执行下述任务：
//首先提示用户输入两个string对象，然后挑出较短的那个并输出它。
#include <iostream>
#include <string>
using std::cout; using std::cin; using std::endl; using std::string;
int main()
{
    string rsp;
    do {
        cout << "Input two strings: ";
        string str1, str2;
        cin >> str1 >> str2;
        cout << (str1 <= str2 ? str1 : str2)
             << " is less than the other. " << "\n\n"
             << "More? Enter yes or no: ";
        cin >> rsp;
    } while (!rsp.empty() && tolower(rsp[0]) == 'y');
    return 0;
}
```

跳转语句

- c++提供了4种跳转语句
    - break continue goto return

BREAK语句

- **break**：break语句负责终止离它最近的while、do while、for或者switch语句，并从这些语句之后的第一条语句开始继续执行。
    - 只能出现在迭代语句或switch语句内部，作用范围仅限于最近的循环或switch

BREAK语句

```
string buf;
while(cin>>buf&&!buf.empty()){
    switch(buf[0]){
        case '-':
            //处理到第一个空白为止
            for(auto it=buf.begin()+1;it!=buf.end();++it){
                if(*it==' ')
                    break;//#1 离开for循环
                //...
            }
            //break #1 将控制权转移到这里
            //剩余-处理
            break;//#2 离开switch语句
        case '+':
            //...
    }//结束switch
    //结束switch，break #2将控制权转移到这里
}//结束while
```

练习

```cpp
//编写一段程序，从标准输入中读取string对象的序列直到
//连续出现两个相同的单词或者所有的单词都读完为止。
//使用while循环一次读取一个单词，当一个单词连续出现两次时
//使用break语句终止循环。
//输出连续重复出现的单词，或者输出一个消息说明没有任何单词是连续重复出现的。
#include <iostream>
#include <string>
using std::cout; using std::cin; using std::endl; using std::string;
int main()
{
    string read, tmp;
    while (cin >> read)
        if (read == tmp) break; else tmp = read;

    if (cin.eof())  cout << "no word was repeated." << endl;
    //eof(end of file)判断输入是否结束,或者文件结束符
    else cout << read << " occurs twice in succession." << endl;

    return 0;
}
```

CONTINUE语句

- 终止最近的循环中的当前迭代并立即开始下一次迭代。只能在while、do while、for循环的内部。
  - 仅用于离它最近的循环
  - 只有当switch语句嵌套在迭代语句内部时，才能在switch里使用continue
  - 对于while do while，继续判断条件的值
  - 对于传统for循环 继续指向for语句头expression
  - 对于范围for循环 用序列下一个元素初始化循环控制变量

CONTINUE语句

```
string buf;
while(cin>>buf&&!buf.empty()){
    if(buf[0]!='_'){
        continue;//接着读取下一个输入
    }
    //程序执行到这里，说明输入以_开头
}
```

练习

```cpp
//修改序，使其找到的重复单词必须以大写字母开头。
#include <iostream>
using std::cin; using std::cout; using std::endl;
#include <string>
using std::string;
int main(){
    string curr, prev;
    bool no_twice = true;
    while (cin >> curr){
        if (isupper(curr[0]) && prev == curr){
            cout << curr << ": occurs twice in succession." << endl;
            no_twice = false;
            break;
        }
        prev = curr;
    }
    if (no_twice)
        cout << "no word was repeated." << endl;
    return 0;
}
```

GOTO 语句

- 无条件跳转到同一函数内的另一条语句
- 不要在程序中使用goto，它使得程序难理解难修改
- 语法形式:

```
goto label;
//label 用于标识一条语句的标识符
//带标签语句，在语句前有一个标识符以及一个冒号
end: return;//带标签语句，可用作goto的目标
//标签标识符独立于变量和其它标识符，不会干扰
```

GOTO 语句

- 与switch 类似，也不能将程序的控制权从变量的作用域之外转移到作用域之内。

```
//..
    goto end;
    int ix=10;//错误 goto绕过一个带初始化的变量定义
end:
    //错误 绕过ix声明
```

GOTO 语句

- 向后跳过一个已经执行的定义时合法的。跳回变量定义之前意味着系统销毁变量，重新创建它。

```
//向后跳过一个带初始化的变量定义时合法的
begin:
    int sz=get_size();
    if(sz<=0){
        goto begin;
    }
//goto语句执行后将销毁sz，sz重新定义初始化
```

练习

```
//本节的最后一个例子跳回到begin，其实使用循环能更好的完成该任务，
//重写这段代码，注意不再使用goto语句。
// 向后跳过一个带初始化的变量定义是合法的
begin:
    int sz = get_size();
    if (sz <= 0) {
        goto begin;
    }


用 for 循环修改的话就是这样
for (int sz = get_size(); sz <=0; sz = get_size())
    ;
```

TRY语句块和异常处理

- 异常是指存在于运行时的反常行为，超出了函数正常功能的范围
    - 典型异常包括失去数据库连接、意外输入等

TRY语句块和异常处理

- c++中异常处理包括
  - **throw表达式**：异常检测部分使用 throw表达式来表示它遇到了无法处理的问题。我们说 throw 引发 raise了异常。
  - **try语句块**：以 try关键词开始，以一个或多个 catch字句结束。 try语句块中的代码抛出的异常通常会被某个 catch捕获并处理。 catch子句也被称为**异常处理代码**。
  - **异常类**：用于在 throw表达式和相关的 catch子句之间传递异常的具体信息。

THROW 表达式

- 程序异常检测部分使用throw表达式引发异常
    - 表达式类型就是抛出的异常类型

```
if (item1.isbn() != item2.isbn())
    throw runtime_error("Data must refer to same ISBN");
    // if we're still here, the ISBNs are the same
    cout << item1 + item2 << endl;
    } catch (runtime_error err) {
//runtime_error 是标准库异常种类的一种，定义在stdexcept头文件
//初始化runtime_error ,给它提供一个string或c字符串。
```

TRY 语句块

- 通用语法形式是

```
try{
    program-statements
}catch (excetion-declaration){
    handle-statements
}catch (excetion-declaration){
    handle-statements
}//..
//try语句块内声明的变量在块外无法访问，catch子句内也无法访问
```

TRY 语句块

```
while (cin >> item1 >> item2) {
    try {
        // execute code that will add the two Sales_items
        // if the addition fails, the code throws a runtime_error exception
        // first check that the data are for the same item
        if (item1.isbn() != item2.isbn())
            throw runtime_error("Data must refer to same ISBN");
        // if we're still here, the ISBNs are the same
        cout << item1 + item2 << endl;
    } catch (runtime_error err) {
        // remind the user that the ISBNs must match
        // and prompt for another pair
        cout << err.what()
             << "\nTry Again?  Enter y or n" << endl;
        char c;
        cin >> c;
        if (!cin || c == 'n')
            break;      // break out of the while loop
    }  // ends the catch clause
}  // ends the while loop
```

```cpp
//throw.cpp
#include <stdexcept>
#include <iostream>
using std::cin; using std::cout; using std::endl;using std::runtime_error;
#include "Sales_item.h"
int main(){
    Sales_item item1, item2;
    while (cin >> item1 >> item2) {
        try {
            if (item1.isbn() != item2.isbn())
                throw runtime_error("Data must refer to same ISBN");
            cout << item1 + item2 << endl;
        } catch (runtime_error err) {
            cout << err.what()
                 << "\nTry Again?  Enter y or n" << endl;
            char c;   cin >> c;
            if (!cin || c == 'n')
                break;      // break out of the while loop
        }  // ends the catch clause
    }  // ends the while loop
    return 0;   // indicate success
}
```

函数在寻找处理代码过程中退出

- 寻找处理代码的过程
  - 首先搜索抛出改异常的函数，没有匹配的catch子句，终止该函数
  - 在调用函数的函数中继续找，如还没有匹配的catch子句，新函数终止，继续找调用它的函数
  - 直到找到适当的catch子句为止
  - 若最终未找到任何匹配的catch子句，程序转到terminate标准库函数，非正常退出。
- 编写异常安全的代码非常困难，超出课程范围。

标准异常

- c++库函数定义了一组异常类
    - exception头文件定义了最通常的异常类exception，只报告异常发生不提供额外信息
    - stdexcept头文件定义了几种常见异常类
    - new头文件定义了bad_alloc异常类
    - type_info头文件定义了bad_cast异常类型

```
// stdexcept头文件定义了几种常见异常类
excpetion            //最常见的问题
runtime_error        //只有在运行时才能检测出的问题
range_error          //运行时错误：生成的结果超出有意义的范围
overflow_error       //运行时错误：计算上溢
underfloe_error      //运行时错误：计算下溢
logic_error          //程序逻辑错误
domain_error         //逻辑错误：参数对应的结果不存在
invalid_argument     //逻辑错误：无效参数
length_error         //逻辑错误：试图创建一个超出类型虽大长度的对象
out_of_range         //使用一个超出有效范围的值
```

- 只能默认初始化 exception bad_alloc bad_cast
- 其它异常类型，应该使用string或c风格字符串初始化对象，不允许默认初始化
- 异常类型只定义一个what的成员函数，没有参数，返回值是c风格字符串的const char*
    - 提供关于异常的一些文本信息
    - 有字符串初值的异常类型，what成员函数返回字符串；其它异常类型，what将返回内容由编译器决定。

练习

```
//编写一段程序，从标准输入读取两个整数，输出第一个数除以第二个数的结果。

#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main()
{
    int i, j;
    cin >> i >> j;
    cout << i / j << endl;
    return 0;
}
```

练习

```
//修改你的程序，使得当第二个数是0时抛出异常。先不要设定catch子句，
//运行程序并真的为除数输入0，看看会发生什么？
#include <iostream>
#include <stdexcept>
int main(void)
{
    int i, j;
    std::cin >> i >> j;
    if (j == 0)
        throw std::runtime_error("divisor is 0");
    std::cout << i / j << std::endl;
    return 0;
}
```

练习

```
//修改上一题的程序，使用try语句块去捕获异常。catch子句应该为用户输出一条提示信息，
//询问其是否输入新数并重新执行try语句块的内容。
#include <iostream>
#include <stdexcept>
using std::cin; using std::cout; using std::endl; using std::runtime_error;
int main(void){
    for (int i, j; cout << "Input two integers:\n", cin >> i >> j; ){
        try {
            if (j == 0)
                throw runtime_error("divisor is 0");
            cout << i / j << endl;}
        catch (runtime_error err) {
            cout << err.what() << "\nTry again? Enter y or n" << endl;
            char c;
            cin >> c;
            if (!cin || c == 'n')
                break;}
    }
    return 0;
}
```

实践课

- 从课程主页 **cpp.njuer.org** 打开实验课 3 界面
    - 使用g++编译代码
    - 编辑一个 **README.md** 文档,键入本次实验心得.
    - 使用git进行版本控制 可使用之前的gitee代码仓库

```
- 云服务器（Elastic Compute Service,简称ECS）
- Aliyun Linux 2是阿里云推出的 Linux 发行版
- Vim是从vi发展出来的一个文本编辑器.
- g++ 是c++编译器
```

习题1
打印所有水仙花数。
（水仙花数是这样的三位数，满足自身数值等于自身数值各位数字的立方和，
如153=1^3+5^3+3^3）

习题2
判断回文数。
（从键盘读入一个正整数，判断数字是否是回文数。回文数满足正向反向数字一样，
如12321是回文数）

习题3
计算圆周率并打印。
（可利用公式 PI/4 = 1-1/3+1/5-1/7+…，直到最后一项绝对值小于10^（-8）.）

附加题1

```
//编写一段程序，从标准输入中读取若干string对象并查找连续重复出现的单词，
//所谓连续重复出现的意思是：一个单词后面紧跟着这个单词本身。要求记录连续
//重复出现的最大次数以及对应的单词。如果这样的单词存在，输出重复出现的
//最大次数；如果不存在，输出一条信息说明任何单词都没有连续出现过。
//例如：如果输入是：
//how now now now brown cow cow
//那么输出应该表明单词now连续出现了3次。
```

附加题2

```
//编写一段程序，从标准输入中读取string对象的序列直到
//连续出现两个相同的单词或者所有的单词都读完为止。
//使用while循环一次读取一个单词，当一个单词连续出现两次时
//使用break语句终止循环。
//输出连续重复出现的单词，或者输出一个消息说明没有任何单词是连续重复出现的。
```

附加题3

```
//编写一段程序，从标准输入读取两个整数，输出第一个数除以第二个数的结果。
//使用try语句块去捕获异常。catch子句应该为用户输出一条提示信息，
//询问其是否输入新数并重新执行try语句块的内容。
```

```
编辑c++代码和markdown文档,使用git进行版本控制
yum install -y git gcc-c++
使用git工具进行版本控制
git clone你之前的网络git仓库test(或其它名字)
cd test 进入文件夹test
(clone的仓库,可移动旧文件到目录weekN:  mkdir -p weekN ; mv 文件名 weekN;)

vim test1.cpp
g++ ./test1.cpp 编译
./a.out 执行程序

vim test2.cpp
g++ ./test2.cpp 编译
./a.out 执行程序

vim test3.cpp
g++ ./test3.cpp 编译
./a.out 执行程序
```

```
git add . 加入当前文件夹下所有文件到暂存区
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
vim readme.md 键入新内容（实验感想),按ESC 再按：wq退出
git add .
git commit –m "weekN" 表示提交到本地,备注weekN

git push 到你的git仓库

git log --oneline --graph 可看git记录
键入命令并截图或复制文字,并提交到群作业.
cat test* readme.md
```

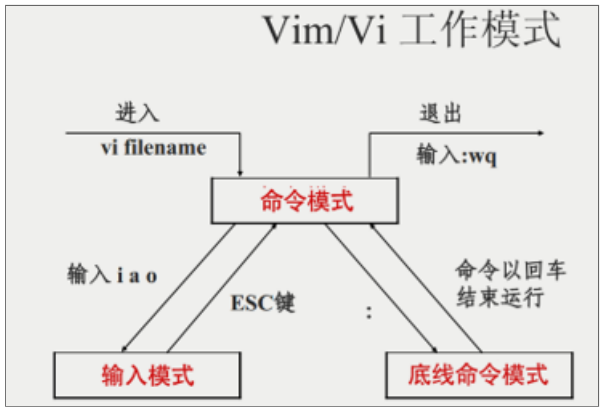
提交

- 截图或复制文字,提交到群作业.
- 填写阿里云平台（本实验）的网页实验报告栏,发布保存.本次报告不需要分享提交
- 填写问卷调查 **https://rnk6jc.aliwork.com/o/cppinfo**

关于使用TMUX

```
sudo yum install -y tmux
cd ~ && wget https://cpp.njuer.org/tmux && mv tmux .tmux.conf
tmux 进入会话 .
前缀按键prefix= ctrl+a,
prefix+c创建新面板,
prefix+"分屏,
prefix+k选上面,prefix+j选下面,
prefix+1选择第一,prefix+n选择第n,
prefix+d脱离会话
tmux attach-session -t 0 回到会话0
```

VIM 共分为三种模式



- 命令模式
  - 刚启动 `vim`,便进入了命令模式.其它模式下按`ESC`,可切换回命令模式
    - `i` 切换到输入模式,以输入字符.
    - `x` 删除当前光标所在处的字符.
    - `:` 切换到底线命令模式,可输入命令.
- 输入模式
  - 命令模式下按下`i`就进入了输入模式.
    - `ESC`,退出输入模式,切换到命令模式
- 底线命令模式
  - 命令模式下按下`:`（英文冒号）就进入了底线命令模式.
    - `wq` 保存退出

VIM 常用按键说明

除了 i, Esc, :wq 之外,其实 vim 还有非常多的按键可以使用.命令模式下:

- 光标移动
  - j下 k上 h左 l右
  - w前进一个词 b后退一个词
  - Ctrl+d 向下半屏 ctrl+u 向上半屏
  - G 移动到最后一行 gg 第一行 ngg 第n行
- 复制粘贴
  - dd 删一行 ndd 删n行
  - yy 复制一行 nyy复制n行
  - p将复制的数据粘贴在下一行 P粘贴到上一行
  - u恢复到前一个动作 ctrl+r重做上一个动作
- 搜索替换
  - /word 向下找word ? word 向上找
  - n重复搜索 N反向搜索
  - :1,$s/word1/word2/g从第一行到最后一行寻找 word1 字符串,并将该字符串取代为 word2

VIM 常用按键说明

底线命令模式下：

- :set nu　　显示行号
- :set nonu 取消行号
- :set paste　　粘贴代码不乱序

【注：把caps lock按键映射为ctrl,能提高编辑效率.】

MARKDOWN 文档语法

```
# 一级标题
## 二级标题
*斜体* **粗体**
- 列表项
  - 子列表项
> 引用
[超链接](http://asdf.com)
![图片名](http://asdf.com/a.jpg)

|表格标题1|表格标题2|
|-|-|
|内容1|内容2|
```

谢谢