Dev-C++基础教程

习题解答

庄燕文 王素琴 王碧艳 编著

前言

这是一本与《Dev-C++基础教程》配套的习题解析。本书主要针对中小学生学习了DEV-C++ 语法、语义、常用算法后，引导学生用程序设计的思维和逻辑解决问题。本书的习题从题意出发，分析解题的思路和算法，然后给出参考程序，重点帮助读者理解算法原理，教会读者很多实用的编程技巧，培养良好的编程习惯。

本书的习题内容丰富，读者可以自己动手实践编写程序，然后和题解中的方法比较，希望读者在理解该书题解的同时，能锐意开掘，探本索源，找到更好的解答办法，写出更高效的程序。

该习题涵盖中学生信息学奥林匹克竞赛中的很多基础算法，可以做为中小学生学习程序设计的入门习题，也可以做为本科生初学程序设计的课后习题。

目录

第1章 Dev-C++入门

第2章 Dev-C++语言简介

第3章 顺序结构

第4章 选择结构

第5章 循环结构

第6章 函数

第7章 数组

第8章 指针

第9章 字符串

第10章 文件

第11章 输入与输出流

第12章 链表

# Dev-C++入门

1. [解析] 该题考察学生用Dev-C++编写简单的程序和C++的基本语法。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

const float PI=3.14; //定义一个常数变量pi

int main () //主程序

{

float s,r,c;

cout<<"please input radius :"<<endl; // 输出"输入半径的提示"

cin>>r; // 输入半径

s=PI\*r\*r; // 计算面积

c= 2\*PI\*r; // 计算周长

cout<<"s="<<s<<endl;

cout<<"c="<<c<<endl;

system("pause");

return 0;

}

[输出结果]

please input radius: // 显示

5 // 用户输入，表示圆的半径

s = 78.5 // 显示面积

c = 31.4 // 显示周长

2. [问题分析] 凡是数学中已经定义的字母，在程序设计中一律沿用。如用a表示边长，用s表示面积，用c表示周长。  
[算法设计] s=a\*a，c=4\*a.

[参考程序]  
#include <iostream>

using namespace std;

int main()

{

float a,s,c;

cout<<"输入正方形的边长：";cin>>a;

s=a\*a;

c=4\*a;

cout<<"s="<<s<<endl;

cout<<"c="<<c<<endl;

system("pause");

return 0;

}

# Dev-C++语言简介

1. [解析] C++ 标识符命名规则：

(1) 必须由一个字母(a~z,A~Z)或下划线(\_)开头

(2) 标识符的其它部分可以用字母，下划线或数字(0~9)组成

(3) 区分大小写

(4) 不能与C++关键字相同

[参考答案]

合法标识符：worth,begin,a34

非法标识符：false,x\*y,$we,char,…set

2. (1) 31/(5%2)

[解析] 这是一个算术表达式。分析其计算结果时首先要掌握两点：

① 算术运算符的优先级。C++的算术运算符中，()的优先级最高，\*、/、%的优先级比+、－高，且它们都是从左到右结合的

运算符两侧类型不同的运算数，在运算中给出类型转换的规律。运算所得结果的类型与运算数类型一致。

在第一题中，括号的优先级最高，先求5%2 = 1，原表达式变成31/1, 根据规定的除法运算规律，所得结果的类型与运算数类型一致，因为”/”号两侧都是整型，因此结果也应该整型。求得31;

[答案] 31

(2) 17-4\*3/7

[解析] 按照优先级的高低，\*的优先级比-高，先算\*法，4\*3=12，原表达式变成17 – 12/7。/的优先级比-高，再算12/7，根据规定的除法运算规律，所得结果的类型与运算数类型一致，因为”/”号两侧都是整型，因此结果也应该整型。计算得12/7 = 1,原表达式变成17 – 1，最后算17-1得16

[答案] 16

(3) 31.0/(5%2)

[解析] 括号的优先级最高，先算括号里面的，5%2 =1，原表达式变成31.0/1, 当运算符两侧为一个为整型数，另一个为实型数时，系统首先把着整型数转换为实型数，使运算符两边的类型取得一致然后进行运算，经转换后表达式为31.0/1.0，最终结果是31.0

[答案] 31.0

(4) 17-4\*3.0/7

[解析] 由于\*的运算级高于-，先算4\*3=12.0，原表达式变成17 – 12.0/7，/的优先级比-高，计算12.0/7，当运算符两侧为一个为整型数，另一个为实型数时，系统首先把着整型数转换为实型数，使运算符两边的类型取得一致然后进行运算，12.0/7.0=1.71429，原表达式变成17-1.71429，当运算符两侧为一个为整型数，另一个为实型数时，系统首先把着整型数转换为实型数，使运算符两边的类型取得一致然后进行运算，经转换后表达式为17.00000-1.71429，最终结果是15.2857

[答案] 15.2857

3. (1) [解析] 在初学者阶段，读者可能只接触到MAIN(主)函数，但作为基础知识，读者应该明确且牢记：c++程序是由函数组成的。每个c++语言的源程序可以包含多个函数，但只能有一个主函数。因此选A是片面的。在c++程序中不存在“过程”这一程序结构。

在正确的程序设计中，要求对源程序有详尽的注释，以便于阅读和软件的维护：c++程序中的注释行，由/\*开头，由\*/结束：”/”和”\*”必须紧接着，在这两个字符之间不得插入空格。在选项D中“/“和”\*”之间都插有空格，很多初学者常犯这样的错误。  
 以上分析可知，本题的正确答案是B。

(2) [解析] 将数学表达式写出c++语言表达式是编写c++程序的基本功之一。此例中给出了一个代数式，用c++语言表示应该是一个算术表达式。在书写算术表达式时，不允许使用分数的形式，只能按运算的先后顺序写在一行上，必须要添加圆括号以保证运算的正确性。由选项可知，同一个数学算式可以写成不同的c++语言表达式。

选项A中，由于除号后的c\*d两侧未加圆括号，按照运算规则，将先算出(a\*b)的乘积，然后被c除，计算的结果再乘上d。它所描述的代数式为，与原算式不符，显然是错误的。

选项B中，按优先级应当先做（C\*D）运算，然后将它去除a\*b的乘积，因此是正确的。  
 选项C中，按从左到右的顺序，先做a/c的运算，描述的代数式为，与原式等效。  
 选项D中，按从左到右的顺序，先做a\*b然后除c，再除d。运算的过程相当于代数式，因此与原算式等效。

题目要求选出不正确的，所以本题的答案是A。

(3) [解析] C++语言中的表达式种类较多，本章中涉及的就有算术表达式、强制类型转换表达式、赋值表达式以及逗号表达式等。读者应该记清它们各自的特征。

本例中选项A末尾有一个分号，从而构成了赋值语句而不是表达式。

选项B是由2个表达式构成的逗号表达式：第1个是赋值表达式、第2个是由自加运算符构成的表达式，语法上没有错误。

选项C中使用了类型说明符int ，若作为强制类型转换表达式，int 两侧必须带有圆括号，这里没有括号是不合法的。

选项D从形式上看似乎是一个赋值表达式，因为c++语言中允许诸如a=b=2这类的赋值表达式。但c++语言也明确规定：赋值运算符（=）左侧只能是变量不允许是常量和算术表达式。选项D的a+7=c+d在赋值号左侧出现了算术表达式，因此是不合法的。

本题的正确答案为B。

4. (1) [解析] C++语言规定：可以在定义的同时给变量赋初值。

其形式为：类型名 变量名=数值[，变量名=数值，……]

按此格式可得本例题所要求的定义语句；

int a=0，b=0；

需要提醒读者的是：对于定义时没有置初值的变量，其值是不确定的随机数，使用时特别加以注意。

[答案] int a = 0,b = 0;

(2) [解析] 这是一个算术表达式。分析其计算结果时首先要掌握两点：①算术运算符的优先级。运算符两侧类型不同的运算数，在运算中给出类型转换的规律。

本例中涉及了加、除以及求余三种运算。按运算的优先级先计算1/2，根据规定除法运算所得结果的类型与运算数类型一致，因为”/”号两侧都是整型，因此结果也应该整型。故1/2的结果不是0.5而应舍去小数部分得到结果0；按优先级，接下来做求余运算56%10，56除10以后的余数是6，因此所得结果为整数6，现在表达式变成了3.5+0+6。按+号的综合性，应该自左至右进行运算，因此先做3.5+0，当运算符两侧为一个为整型数，另一个为实型数时，系统首先把着整型数转换为实型数，使运算符两边的类型取得一致然后进行运算，如3.5+0+6，则首先把0转换为实型数0.0，然后进行运算，得3.5；接着进行3.5+6的运算，经转换后表达式为3.5+6.0，最终结果是9.5。

(3) [解析] 在c++语言中，凡是双目运算符都可以和赋值运算符一起组成复合运算符。在赋值表达式中可以包含这些复合运算符，且运算符按“自右至左”的组合顺序进行。因此，表达式k-=k-=k+1的求解过程为：①先计算最右边的“k-=k+1”，相当于k=k-(k+1)，把7带入表达式中：k=7-(7+1)，结果把-1赋给了k。②再计算“k+=-1”，相当于k=k+(-1)，把当前k的值（-1）带入表达式中，k=-1+(-1)，最后把-2赋给了k。故本题的正确答案是-2。

(4) [解析] 先做a=a-a\*a；把12带入得到a=12-12\*12，a=-132.再a=a+(-132)，

a=-132-132=-264

故本题的正确答案是D。

5. (1) [解析] 考察C++的基本语法。该题主要有两点问题。

① main函数是C++程序的主函数，函数名称后面要有(),函数名称前面有类型说明符int。完整的形式应为int main()

② cout 插入符的方向有问题，应该为cout << v;

[答案]

#include <iostream>

usingnamespace std；

int main //改成int main()  
{

int a，b，c，v；  
 a=2，b=3，c=4

v=a\*b\*c；

cout>>v； //改成cout << v;

system(“pause”)；

return 0；

}

(2) [解析] 考察C++的基本语法。

该题主要有两点需要注意的。

① C++的输入输出流头文件中规定了cin,cout是小写形式，若改成大写，系统会默认成变量处理，会出现语法错误

② C++变量是区分大小写的，所以使用时要多小心。不然会出现没有定义的错误。

[答案]

#include <iosteam>

using namespace std；

int main()

{

float L，W，H，V；

cout<<”input L W H :”；CIN>>L>>W>>H；//改为cin >> L >>W>>H

v=L\*W\*H；//改为V = L \*W \*H

cout<<v<<endl；//改为cout << V << endl;

system(“pause”)；

return 0；

}

(3) [解析] 考察C++的const常量。const常量必须在第一次声明时就初始化，用变量或常量初始化都可以，只是初始化一次以后它的值就不能再改变了。若要改变它的值，使用普通变量即可。

[答案]

int main()  
{

const int w=5；//const常量不能修改，改为int w = 5  
float a；  
a=2\*w；  
w++；  
cout<<w<<’’<<a<<endl；

system(“pause”)；

return 0；   
}

# 顺序结构

1. [解析] 考察变量间的交换，算法如下：

(1) 开辟工作单元Ｅ。

(2) Ｅ←A；A←B；B←C；C←D；D←Ｅ。

交换过程读者可以参考教材第三章例3-1中图3-3。

[参考程序]

#include<iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int a = 89, b = 76, c = 95, d = 62, e; //变量申明和初始化

cout << "a=" << a << " " << " b = "<< b << " " << " c = "<< c << " "<< " d = "<< d << endl; //输出变量的初始值

e = a;

a = b;

b = c;

c = d;

d = e;

cout << "a=" << a << " " << " b = "<< b << " " << " c = "<< c << " "<< " d = "<< d << endl; //输出移动后变量的值

system("pause");

return 0;

}

[输出结果]

a = 89 b = 76 c = 95 d = 62 //交换前的值

a = 76 b = 95 c = 62 d = 89 //交换后的值

2. [解析] 本题主要考察变量的格式化输出。

C++标准库提供的支持格式输出的操作符函数有控制输出宽度、填充字符和精度的方法。需要注意的是在使用这些操作符函数时需要包含头文件iomanip。这里给读者介绍下输出宽度的函数：setw(宽度值)和设置输出填充字符的函数：setfill (填充字符)。

输出宽度的函数：setw(宽度值)，如输出cout << setw(5) << 3 <<endl;就表示将整数3以宽度5的格式输出，当需要的宽度小于5时，左补0。这样就实现了在整数5左边输出4个空格的目的。运行结果为□□□□5。

设置输出填充字符的函数：setfill (填充字符)

此函数常与setw()函数联合使用，达到向不满设置输出宽度的空间填入指定字符的目的，不设置则填充空格。cout<<setw(5)<<setfill('#')<<123<<endl，指将整数123以宽度5的格式输出，运行结果为##123。

读者可以参考第三章的例题3-3，3-4。这里列举直接输出空格和使用宽度函数输出空格的方法。

[参考程序]

方法一：

#include<iostream> //预编译命令

using namespace std; //使用名字空间

int main () //主函数

{ int a = 1,b = 2,c = 3,d = 4,e = 5; //变量定义和初始化

cout<<a<<" "<<b<<" "<<c<<" "<<d<<" "<<e

<<endl; //直接输出空格

system("pause");

return 0;

}

方法二：

#include<iostream> //预编译命令

#include<iomanip> //格式化输出函数的头文件

using namespace std; //使用名字空间

int main () //主函数

{

int a=1,b=2,c=3,d=4,e=5; //变量定义和初始化

cout<<setw(1)<<a<<setw(3)<<b<<setw(4)<<c<<setw(6)<<d<<setw(10)<<e<<endl; //使用输出宽度函数输出空格

system("pause");

return 0;

}

[输出结果]

1□□2□□□3□□□□□4□□□□□□□□□5 //显示结果

3. [解析] 本题考察格式化输入输出，本题采用宽度函数设置输出宽度。读者可以参考本章习题的第2小题。

[参考程序]

#include<iostream> //预编译命令

#include<iomanip> //格式化输出函数的头文件

using namespace std; //使用名字空间

int main () //主函数

{ int a,b,c; //变量申明

cout << “please input the value a and b:”<< endl; //输入a，b的提示信息

cin >>a>>b; //输入a,b

cout<<setw(8)<< a<< endl; // 设置a输出的格式

cout<<"+"<< setw(7)<<b<<endl; // 设置‘+’输出的位置

cout<<"-----------"<<endl; //输出“-----------”

c = a+b; //计算a+b的值赋给c

cout<<setw(8)<<c<<endl; //设置输出的格式

system("pause");

return 0;

}

[输出结果]

please input the value a and b: //显示

4587 2124 //用户输入

4587 //显示加法结果

+ 2124

----------------

6766

4. [解析] 本题主要考察输出精度，宽度等格式化输出。

设置输出精度的函数：setprecision(有效位数)。此函数用来指明显示浮点型数据的有效位数，可用的最后一位的值为是四舍五入的值。如cout<<setprecision(3)<<3.333333<<endl；运行结果为3.33。

本题将输出精度和输出宽度结合就可以实现对齐的目的，读者可以参考第三章例题3-6。

[参考程序]

#include<iostream> //预编译命令

#include<iomanip> //格式化输出函数的头文件

using namespace std; //使用名字空间

int main () //主函数

{

float a, b, e; //变量申明

int c, d; //变量申明

cin >> a >> b >> c >> d; //输入变量

cout << setiosflags(ios::fixed) << setw(8) << setprecision(c) << a << endl;

cout << setiosflags(ios::fixed) << "+" << setw(8 - c + d - 1) << setprecision(d) << b << endl;

//以上两行格式化输出a,‘+’，b

cout << "-----------" << endl; // 输出“---------”

e = a + b; //计算e的值

cout << setw(8) << e << endl; //格式化输出e

system("pause");

return 0;

}

[输出结果]

1.2345 2.3456 3 3 //输入

1.235

+ 2.346 //输出

--------------

3.580

5. [解析] 本题主要考察小时和分钟的转化，转换方式如下：1小时= 60分钟，若要用分钟a表示m小时n分钟，最后的结果a=(m%24)\*60+n。若要用小时和分钟表示分钟a，则小时m = a / 60, 分钟n = a – m\*60。可以参考第三章例题3-10

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int a,b; //变量申明，a代表小时，b代表分钟

cin >> a >> b; //输入变量a,b

if(a >= 24)

a = a % 24; //过24点记为0.

cout << a \* 60 + b << endl; //输出用分钟表示的结果

system("pause");

return 0;

}

[输出结果]

5 34 //输入小时和分钟

334 //输出分钟表示的结果

6. [解析] 本题主要考察华氏温度和摄氏温度的转换。按照转换公式是：C=(5/9)(F-32)进行转换，这里需要注意的一点是，这里的5/9中/两边均是整数，如果将其中之一从整型数化为实型数的话，计算的结果会是0。若将5转化为5.0，原表达式变成C = (5.0/9)(F-32)。计算5.0/9时，运算符两侧为一个为整型数，另一个为实型数时，系统首先把着整型数转换为实型数，使运算符两边的类型取得一致然后进行运算。这样就能得到正确的转换结果。

[参考程序]

#include<iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

float a,b,c,d,e; //变量申明

cin >> a >> b >> c >> d >> e; //输入转换前的变量值

cout <<(float)(5.0/9) \*(a - 32) << " " << (float)(5.0/9) \* (b - 32) << " " << (float)(5.0/9) \* (c - 32) << " "<< (float)(5.0/9) \*(d - 32) << " " << (float)(5.0/9) \*(e - 32) << endl; //输出转换后的变量值

system("pause");

return 0;

}

[输出结果]

-10 0 10 37 100 //输入的华氏温度

-23.3333 -17.7778 -12.2222 2.77778 37.7778//输出的摄氏温度

# **选择结构**

1. [解析] 该题主要考察if-else来比较变量的大小的方法。参考教材第四章的例4-2，例4-3，这里介绍四种方法。

[参考程序]

方法一：

逐层判断，这种方法可以锻炼自己的逻辑思维的能力。

参考程序如下：

#include <iostream> //预编译命令

using namespace std； //使用名字空间

int main() //主函数

{

float a，b，c； //变量申明

cin>>a>>b>>c； //输入变量值

if (a>b) //比较求的中间值

{

if (a>c)

{

if(b>c) cout<<b<<endl；

else cout<<c<<endl；

}

else cout<<a<<endl；

}

else

{

if(b>c)

{

if(a>c)cout <<a<<endl；

else cout<<c<<endl；

}

else cout<<b<<endl；

}

system("pause")；

return 0；

}

方法二：

假设变量a中存放这些数中最大的值。如不是就交换，这样就保证变量a中存放的是最大值。只要比较b和c，找出他们中的较大数值即可。

参考程序如下：

#include <iostream> //预编译命令

using namespace std； //使用名字空间

int main() //主函数

{

float a，b，c，d； //变量申明

cin>>a>>b>>c； //输入变量值

if(b>a)

{d=a；a=b；b=d；}

if(c>a)

{d=c；c=a；a=d；} //以上四行表示用a存放变量最大值

if(b>c) cout<<b<<endl；

else cout<<c<<endl； //以上两行表示输出中间值

system("pause")；

return 0；

}

方法三：

假设b是之间的数，可能的所有情况。在考虑a是中间数的所有情况，最后考虑c是中间数的所有情况。

参考程序如下：

#include <iostream> //预编译命令

using namespace std； //使用名字空间

int main() //主函数

{

float a，b，c； //变量申明

cin>>a>>b>>c； //输入变量值

if(a>b && b>c) cout<<b<<endl； //通过比较输出中间值

if(c>b && b>a) cout<<b<<endl； //通过比较输出中间值

if(b>a && a>c) cout<<a<<endl； //通过比较输出中间值

if(c>a && a>b) cout<<a<<endl； //通过比较输出中间值

if(a>c && c>b) cout<<c<<endl； //通过比较输出中间值

if(b>c && c>a) cout<<c<<endl； //通过比较输出中间值

system("pause")；

return 0；

}

方法四：

改进方法三。当然最简洁的方法是数学方法。

If((a-b)\*(b-c)>0) cout<<b<<endl；

这样一条语句可以代替方法三中的两条语句：

if(a>b && b>c) cout<<b<<endl；

if(c>b && b>a) cout<<b<<endl；

参考程序如下：

#include <iostream> //预编译命令

using namespace std； //使用名字空间

int main() //主函数

{

float a，b，c； //变量申明

cin>>a>>b>>c； //输入变量值

if((a-b)\*(b-c)>0)cout<<b<<endl； //输出中间值

else

if((b-a)\*(a-c)>0)cout<<a<<endl； //输出中间值

else cout<<c<<endl； //输出中间值

system("pause")；

return 0；

}

求最大值、最小值的方法同学们可以按照这四种思路编写程序。

[输出结果]

2.3 23.4 3.4 //输入的三个实数

3.4 //输出的中间值

2. [解析] 该题主要考察一元二次方程求根需注意的地方，考察学生思维的严密性。

要注意的地方是：

(1) 需分别讨论系数a为0和不为0的情况,利用a可以区分该方程是一元一次方程还是一元二次方程。

(2) 判别式b\*b-4\*a\*c计算出来后赋值给一个变量，这样引用的时候就不需要重新计算，减少了计算时间。利用b\*b-4\*a\*c可以判断该方程有没有实根。

[参考程序]

#include <iostream> //预编译命令

#include <cmath> //加入求平方根函数的头文件

using namespace std; //使用名字空间

int main() //主函数

{

float a,b,c; //变量申明，分别表示方程的三个参数

float x1,x2; //变量申明，分别表示方程的两根

cin >> a >> b >> c; //输入变量a,b,c

float d = b\*b - 4 \* a \* c; //求判别式的值

if(a == 0) //该方程为一元一次方程

{

cout << "此方程为一元一次方程" << endl;

cout << "x1 = x2 = " << (-c) / b;

}

else if( d >= 0) //该方程为一元二次方程

cout << " x1 = " << (-b + sqrt(d)) / (2 \* a) << "x2 = " << (-b - sqrt(d)) / (2 \* a)<< endl;

else

{

cout << "此方程没有实根" << endl;

}

system("pause");

return 0;

}

[输出结果]

2 5 3 //输入的a,b,c的值

x1 = -1 x2 = -1.5 //输出的根

3. [解析] 该题考察运算符的优先级：算术运算符>关系运算符>逻辑运算符，(1) a为真，!a为假，所以&&操作结果为假;(2)a&&b为假，!(a&&b)为真，故整体表达式为真，(3) a为真，!a为假，所以&&操作结果为假,(4)==左边得1，右边也是1，故等号成立。

[答案] (2),(4)

4. [解析] 该题考察一种多路径分支控制语句。在执行多路径分支控制时，用switch语句写出的程序比用if语句写的程序更简洁、清晰。且不易出错。其中switch后的表达式的类型可以是整型、字符型，但不能是实型。所以这里需要注意的地方是将条件不等式转换为整数。根据题意中奖金税的条件不等式，可以知道根据奖金的千位值就可以得到其税率。其中，千位值可以通过奖金/1000得到，再利用switch来求解。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int rewards,n; //变量申明

cin >> rewards; //输入奖金

if(rewards <= 1000) //将某一范围内的奖金转换为整数

n = 0;

else

n = rewards / 1000;

switch(n) //根据奖金，求税率

{

case 0: cout << n\*0\*1000 << endl; break;

case 1: cout << n\*0.03\*1000 << endl; break;

case 2: cout << n\*0.05\*1000< endl; break;

case 3: cout << n\*0.07\*1000 << endl; break;

case 4: cout << n\*0.09\*1000 << endl; break;

case 5: cout << n\*0.12\*1000< endl; break;

case 6: cout << n\*0.16\*1000 << endl; break;

case 7: cout << n\*0.18\*1000 << endl; break;

case 8: cout << n\*0.20\*1000 << endl; break;

case 9: cout << n\*0.22\*1000 << endl; break;

default: cout << n\*0.25\*1000 << endl;

}

system("pause");

return 0;

}

[输出结果]

5500 //输入的奖金额

660 //输出的奖金税

5. [解析] 该题考察用switch来实现运算符的判断和操作。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int a,b,d; //整型变量申明

char c; //字符型变量申明

cin >> a >> b; //输入要进行运算的两数

cin >> c; //输入运算符

switch(c) //根据不同的运算符，求运算结果

{

case '+' : cout << a + b << endl; break;

case '-' : cout << a - b << endl; break;

case '\*': cout << a \* b << endl; break;

case '/' : {

f(b == 0)

cout << "除数不能为0" << endl;

else

cout << a / b << endl;

}

}

system("pause");

return 0;

}

[输出结果]

1 2 + //输入的运算数和运算符

3 //输出的计算结果

6. [解析] 月份的天数除二月外，其他都是固定的，某个月所属的季节也是固定的。二月的天数需要判断当前年份是否是闰年，该题考察闰年的判断方法，不同月份的日期天数，考察学生用合适的选择语句来实现。只要满足以下两个条件其中之一即为闰年：

(1)该年份能被4且能被100整除；

(2)该年份不能被400整除。

[参考程序]

#include <iostream> //预编译命令

#include <cstring> //字符串的头文件

using namespace std; //使用名字空间

int main() //主函数

{

int year,month,days; //整型变量申明

string season; //string类型变量申明

cin >> year >> month; //输入年份和月份

bool leap; //布尔变量申明，表示是否为闰年

if (month >= 2 && month <= 4) //根据月份判断季节

season = "Spring";

else if(month >= 5 && month <= 7)

season = "Summer";

else if(month >= 8 && month <= 10)

season = "Autumn";

else

season = "Winter";

if(!(year % 4) && (year % 100) || !(year % 400)) //根据年份判断是否为闰年

leap = 1;

else

leap= 0;

switch(month) //根据月份得到该月的天数

{

case 1:

case 3:

case 5:

case 7:

case 8:

case 10:

case 12:

days = 31; break;

case 4:

case 6:

case 9:

case 11:

days = 30; break;

case 2:

if(leap)

days = 29;

else

days = 28;

}

//输出该月的天数和季节

cout << "days = " << days << " " << "season is " << season << endl; system("pause");

return 0;

}

[输出结果]

2011 4 //输入的年份和月份

days = 30 season is Spring //输出该月的天数和季节

7. [解析] 实足年龄是指该学生从出生到现在总共经历了的年月日数。所以只需将当前日期减去学生的生日。从最低位开始，如果相减后出现负值的话，只需从高位借1即可。例如一个学生的生日是1984年7月19日，当前日期是2011年4月6日，想减后的年数是27，月数是-3，日数是-13。这里的日期是负数，我们从月份借1，日期变成-13+30 = 17,月数变成-3-1 = -4，也是负数，所以需从年份借1，月数变成-4+12 = 8，年数就变成27-1= 26，所以该学生的实足年龄是26岁8月17天。

[参考程序]

#include <iostream> //编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int y0,m0,d0,y1,m1,d1,age; //变量申明

cout << "请输入此人的出生日期："; //提示信息

cin >> y0 >> m0 >> d0; //输入学生的出生年月日

cout << "请输入当前日期："; //输出提示信息

cin >> y1 >> m1 >> d1; //输入当前的日期

int y,m,d; //变量申明

d = d1- d0; //输出出生日期和当前日期的差距

m = m1 - m0;

y = y1 - y0;

if(d < 0) //如果日期小于0，则从月份中借一位，日期加30，月份减1

{

d = d + 30;

m --;

}

if(m < 0) //如果月份小于0，则从年份中借一位，月份加12，年份减1

{

m = m + 12;

y --;

}

cout << "年:" << y << " 月:" << m << " 日:" << d << endl; //输出实足年月日

system("pause");

return 0;

}

[输出结果]

请输入此人的出生日期：1947 7 24 //输入的出生日期

请输入当前日期 2011 4 5 //输入的当前日期

年：63 月：8日：11 //输出的实足年龄

8. [解析] 该题考察多条件选择语句的运用。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int amount; //整型变量申明

float discount; //字符型变量申明

cin >> amount; //输入购物金额

if(amount < 200) //根据购物金额求折扣率

discount = 1;

else if(amount >= 200 && amount < 500)

discount = 0.9;

else if(amount >= 500 && amount < 1000)

discount = 0.8;

cout << amount \* discount << endl; //购物金额和折扣率求应付金额

system("pause");

return 0;

}

[输出结果]

520 //输入的购物金额

416 //输出的应付金额

9. [解析] 该题考察构成三角形的条件，判断不同类型三角形的条件。

(1) 构成三角形的条件是：两边之和大于第三边，两边之差小于第三边。假设a,b,c是三角形的三边，则满足a+b > c && b+c > a && a+c > b就能构成三角形，因为a – b < c,b – c < a, a – c < b可以从上面的三个不等式推出。

(2) 当三角形的三边a,b,c的其中两条边相等时，是等腰三角形

(3) 当三角形三边a,b,c三边都相等时，是等比三角形

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

float a,b,c; //实型变量申明

cin >> a >> b >> c; //输入三角形的三边长

//判断是否能构成三角形，若能构成三角形，判断是什么三角形

if(a>0.0 && b>0.0 && c>0.0 && a + b > c && b + c > a && a + c > b)

{

cout << "这三边能构成三角形！"<< endl;

if(a == b == c)

cout <<"该三角形是等边三角形"<< endl;

else if(a == b || b == c || a == c)

cout <<"该三角形是等腰三角形"<< endl;

else

cout <<"该三角形是一般三角形"<<endl;

}

else

cout << "这三边不能构成三角形" << endl;

system("pause");

return 0;

}

[输出结果]

1 2 3 //输入的三角形三边长

这三边不能构成三角形 //输出的三角形信息

10. [解析] 该题考察奇偶选择条件。当一个数能被2整除时，是偶数。否则是奇数。判断一个数是否能被2整除，只需将该数模2看余数是否为0。当余数为0时，说明可以整除，是偶数，否则是奇数。

[参考程序]

#include<iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int a; //整型变量申明

cin >> a; //输入变量a的值

if(a % 2 == 0) //若a能被2整除，则是偶数

cout << "is Even" << endl;

else //若a不能被2整除，则是奇数

cout << "is Odd" << endl;

system("pause");

return 0;

}

[输出结果]

45 //输入的整数

is Odd //输出的判断信息

# 循环结构

1. [解析] 翻N番是原来的2^N倍所以翻一番是原来的2倍，两番是原来的4倍。此题需要注意的地方是告诉你的是去年的总产值，多少年后总产值是相对于今年来说的。题意中告诉的去年的总产值是100万元，则今年总产值是100\*1.06=106万元。翻两翻后总产值是(2^2)\*100 = 400。所以题意要求的是在每年平均增长6%的趋势下，多少年后能由总产值是106万元增长到400万元。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

float product = 100; //product代表去年总产值

float discount = 1.06; //discount是增长率

product = product \* discount; //计算后得到的product是今年总产值

int i;

for(i = 1; ; i ++)

{

if(product < 400)

product = product \* discount; //经过i年后若总产值大于400，退出

else

break;

}

cout << i << endl; //输出经过的年份

system("pause");

return 0;

}

[输出结果]

24 //翻两翻需要经历的年份

2. [解析] 输入一个大于1的自然数N，根据N的奇偶，对N进行转换。当N大于1时，该过程重复执行；当N小于等于1的时候，输出N的值，看N是否为1.当N大于1时，要重复执行上述过程，所以用while循环实现。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int n; //变量申明

cin >> n; //输入变量n

while(n > 1) //当n >1时，循环执行

{

if(n % 2 == 0)

n = n / 2;

else

n = 3 \* n + 1;

}

cout << "n = " << n << endl; //当n <=1时，输出n的值

system("pause");

return 0;

}

[输出结果]

1000 //输入的整数n

n = 1 //输出的结果

3. [解析] 该题指定了自然数的范围，用for循环。先定义一个变量sum代表和，初始化为0，给定1~100的自然数，判断是否为偶数，若是，则将其与sum相加，结果赋给sum。求奇数和的方法类似。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int i,sum; //变量申明

sum = 0; //将sum初始化为0

for(i = 1; i < 100; i ++) //对1~100的自然数进行循环

{

if( i % 2 == 0) //判断是否是偶数

sum += i; //若是偶数，则与sum相加，赋给sum

}

cout << "sum = " << sum << endl; //输出结果

system("pause");

return 0;

}

[输出结果]

sum = 2450 //输出的求和结果

4. [解析] 首先定义变量sn,初始化为0。然后将a赋给第一项，根据前一项求后一项的值，且每一项都和sn相加，结果赋给sn。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int a,n,i=1,sn=0,tn=0; //变量申明和初始化，sn表示和，tn表示当前项的值

cout<<"a，n="; //输入变量的提示信息

cin>>a>>n; //输入变量

while(i<=n) //i表示当前项a的个数

{

tn=tn+a; // tn是当前项的值

sn=sn+tn; //每一项都和sn相加，结果赋给sn

a=a\*10; //此处a是为保存下一项的高位值

++i;

}

cout<<sn<<endl; //输出求和结果

system("pause");

return 0;

}

[输出结果]

a,n = 2 7 //输入的a,n值

2469134 //输出的求和结果

5. [解析] 表达式的值要注意求每一项的值和求和的方法。此题还需注意的地方是如果变量定义为整型，则求每一项的值时需要做强制类型转换为实型。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int i; //整型变量申明

float sum = 0; //和定义为实型

for(i = 2; i <=100; i ++)

sum += (float)1 / i; //每一项经过强制类型转换为实型

cout <<"sum = " <<sum <<endl; //输出求和的结果

system("pause");

return 0;

}

[输出结果]

sum = 4.18738

6. [解析] 该题需要注意两点：

(1) 通项的计算方法

(2) 通项中整型转换为实型的强制类型转换

[参考程序]

#include<iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

float sum = 0; //”和”申明为实型变量，并初始化为0

int i = 1;

for(i = 1; i <= 19; i ++)

sum +=(float) i / (i + 1); //求每一项的方法，注意强制类型转换

cout << "sum = " << sum << endl; //输出和

system("pause");

return 0;

}

[输出结果]

sum = 16.4023

7. [解析] 该题需要注意三点：

(1) 通项的计算方法；

(2) 奇偶项的正负号差异；

(3) 通项中整型转换为实型的强制类型转换。

[参考程序]

#include<iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int i,t = 1; //变量的申明和初始化，t代表每一项的分子，含符号

float sum = 0; //和申明为实型变量，初始化为0

for(i = 2; i <= 100; i ++)

{

sum += (float)t / i; //求每一项，强制类型转换

t = -t; //每一项的符号变化

}

cout << "sum = " << sum << endl; //输出和

system("pause");

return 0;

}

[输出结果]

sum = 0.311828

8. [解析] 该题的计算方法和第7题类似，除t的初始化(t表示每一项的分子，含符号)，t初始化为-1。

[参考程序]

#include<iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int i,t = -1; //变量的申明和初始化，t代表每一项的分子，含符号

float sum = 0; //和申明为实型变量，初始化为0

for(i = 2; i <= 100; i ++)

{

sum += (float)t / i; //求每一项，强制类型转换

t = -t;

}

cout << "sum = " << sum << endl; //输出和

system("pause");

return 0;

}

[输出结果]

sum =-0.311828

9. [解析] 该题告诉了整数的范围1~100,用for循环

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int i,sum = 0; //变量的申明和初始化

for(i = 1; i <= 100; i ++)

{

if(i % 2 == 0 && i % 3 ==0) //判断i是否满足条件

sum += i; //若满足，则求和

}

cout << "sum=" << sum << endl; //输出结果

system("pause");

return 0;

}

[输出结果]

sum =816

10. [解析] 该题告诉了求和的范围，和每一项的值，用for循环实现。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int i,sum = 0; //变量的申明和初始化

for(i = 1; i <= 9; i ++)

sum += i \* 10 + i; //注意求每一项的方法

cout << "sum = " << sum << endl; //输出结果

system("pause");

return 0;

}

[输出结果]

sum =495

11. [解析] 该题考察使用循环的格式化输入输出，下三角形的条件(列的下标小于等于行的下标)。该题需要注意三点：

(1) 第0行第0列\*号的打印，第0行第1~9列表头的打印。

(2) 第0列第1行~第9行的打印。

(3) 下三角形的打印。

[参考程序]

#include<iostream> //预编译命令

#include<iomanip> //格式输入输出的头文件

using namespace std; //使用名字空间

int main () //主函数

{

int i,j,k; //变量的申明

cout<<setw(3)<<"\*"<<setw(3)<<"|"; //格式化输出”\*|”

for(i=1;i<=9;i++)

{

cout<<setw(6)<<i; //格式化输出乘法表的第0行的后9列

}

cout<<endl; //输出换行符，以便输出后面的行

for(i=1;i<=9;i++)

{

cout<<setw(3)<<i<<setw(3)<<"|"; //格式化输出后9行的第一列

for(j=1;j<=i;j++) //注意j的范围，该题输出的是下三角形

{

k=i\*j; //计算第i行第j列的乘积

cout<<setw(6)<<k; //格式化输出结果

}

cout<<endl; //输出换行符

}

cout<<endl; //输出换行符

system("pause");

return 0;

}

[输出结果] 如表5-2所示。

12. [解析] 该题考察使用循环的格式化输入输出，上三角形的条件(行的下标小于等于列的下标)，该题需要注意三点：

(1) 第0行第0列\*号的打印，第0行第1~9列表头的打印。

(2) 第0列第1行~第9行的打印。

(3) 上三角形的打印。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

cout << '\*' << "|"<< "\t"; //格式化输出”\*|”

for( int j = 1; j < 10; j ++)

cout << j << "\t"; //格式化输出乘法表的第0行的后9列

for(int i = 1; i < 10; i ++)

{

cout << i << "|" << "\t"; //格式化输出后9行的第一列

for(int j = 1; j < 10; j ++)

{

if(j < i) cout << "\t"; //若不是上三角形的元素，输出’\t’

else

cout << j \* i << "\t"; // 格式化输出第i行第j列的乘积

}

}

system("pause");

return 0;

}

[输出结果] 如表5-3所示.

13. [解析] 该题考察三位数每一位上数字的求法。用i表示三位数，在百位上的数字a = i / 100,十位上的数字b= (i - a \* 100) / 10，个位上的数字c = (i - a \* 100) % 10。该题可以利用的条件有四个：

(1) 整数i是三位数： 100 <=i<=999

(2) 末尾数字c比首位数字a大： c > a

(3) 首位数字a比中间数字b大: a > b

(4) a,b,c相加等于a,b,c相乘：(a + b + c) == a \* b \* c

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int a,b,c,i; //变量的申明

for(i = 100; i <= 999; i ++) //对所有三位数依次进行判断

{

a = i / 100; //a表示百位数字

b = (i - a \* 100) / 10; //b表示十位数字

c = (i - a \* 100) % 10; //c表示个位数字

if(c > a && a > b && (a + b + c) == a \* b \* c) //判断i是否满足条件

cout << i << endl; //满足则输出结果

}

system("pause");

return 0;

}

[输出结果]

213 //输出小明家的门牌号码

14. [解析] 该题考察学生的推理能力，通过数字等式推理出A,BC,DEF,GHIJ的范围，然后进行循环，最后根据相等条件找到A,B,C,D,E,F,G的值。根据D是DEF百位上的值，求和的结果GHIJ是四位数，则易给G = 1，D = 9,H= 0，则A,B,C,E,F,I,J的取值范围是2~8，且它们互不相同。所以，总结下该题可以利用的条件有四条：

(1) 等式A+BC+DEF=GHIJ

(2) G = 1,D = 9,H= 0

(3) A,B,C,E,F,I,J的取值范围是2~8,可以推得 22<X<88, 922<Y<988,1022<Z<1088，2<A<8。

(4) A,B,C,E,F,I,J互不相同

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int X,Y,Z,A; //用X,Y,Z,A表示BC,DEF,GHIJ,A的值

int G = 1,D = 9,H = 0;

for(X = 22; X < 88; X ++) //根据X求B,C

{

int B = X /10;

int C = X % 10;

if( B == C) //如果B和C相等，则退出

continue;

for(Y = 922; Y < 988; Y ++) //根据Y求的D,E,F

{

int D = Y / 100;

int E= (Y - D \* 100) / 10;

int F = Y % 10;

if( D == E || D == F || E == F || B == D || B == E || B == F || C == D || C == E || C == F || E == H || E == D || E == G|| F == H || F == D || F == G|| C == H || C == D || C == G ) //如果有某些数字相等，则退出

continue;

for(Z = 1022; Z < 1088; Z ++) //根据Z求I,J

{

int I = (Z - 1000) / 10;

int J = (Z -1000) % 10;

if( I == J || I == B || I == C || I == D || I == F || I == E || J == B || J == C || J == D || J == F || J == E|| H == J || G == J)

continue;

for(A = 2; A <= 8; A ++) //对A进行循环，判断是否有相等

{

if(A == B || A == C || A == D || A == E || A == F || A == I || A == J)

continue;

if(A + X + Y == Z)

cout << A << '+' << X << '+'<< Y << '='<< Z << endl;

}

}

}

}

system("pause");

return 0;

}

[输出结果]

//所有满足条件的等式

2+87+964 = 1053 //略去所有的满足条件的等式

15. [解析] 设i表示需要大马的数量，当瓦数是100时，i的范围是[0,100/3];当使用了i匹大马之后，剩余的瓦数是(100 – 3\*i),则需要母马数量的范围是[0,(100 – 3\*i)/2 ];当使用了i匹大马，j匹母马后，剩余的瓦数是(100 – 3 \* i – 2 \*j),则需要的小马的数量是[0,(100 – 3\*i – 2 \*j)\*2]。该题可利用的条件有三条：

(1) 大马，中马，小马i,j,k的范围

(2) 总共马数量是100匹

(3) 总共瓦数是100块

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main()

{

int i,j,k;// i表示需要大马的数量，j表示需要母马的数量，k表示需要小马的数量

for(i = 0; i <= 100 / 3; i ++) //大马的数量进行循环

{

for(j = 0; j <= (100 - 3 \* i) / 2; j ++) //母马的数量进行循环

{

for(k = 0; k <= (100 - 3 \* i - 2 \* j) \* 2; k = k + 2)//小马数量进行循环

{

//总共马的数量是一百匹，瓦的块数是一百块

if((i + j + k )== 100 && (3 \* i + 2 \* j + k / 2) == 100)

//满足条件的情况下，输出大马，母马，小马的数量

cout << "i = " << i << " j = " << j << " k = " << k << endl;

}

}

}

system("pause");

return 0;

}

[输出结果]

//所有满足条件的小马，中马，大马的数量

i = 2 j = 30 k = 68 //略去所有的满足条件

16. 有三种纪念邮票，第一种每套一张售价 2元，第二种每套一张售价 4元，第三种每套 9张售 2元.现用 100 元买了 100 张邮票.问这三种邮票各买几张?

[解析] 设第一种邮票买了i张，则i的范围是[0,100/2];在购买了第一种邮票i张后，购买第二种邮票j张，则j的范围是[0,100 – 2 \* i];在购买了第一种邮票i张，第二种邮票j张后，购买第三种邮票k的范围是[0,100 – 2 \*i – 4 \*j],且三种邮票总的张数是100张，票面价值是100元。

该题可利用的条件有三条：

(1) 第一种，第二种，第三种邮票i,j,k的范围

(2) 总共邮票的张数是100张

(3) 总共钱数是100元

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main()

{

int i,j,k; //i,j,k分别表示买了的第一种，第二种，第三种邮票的套数

for(i = 0; i <= 100 / 2; i ++ ) //购买第一种邮票i张

{

for(j = 0; j <= 100 - 2 \* i; j ++) //购买第二种邮票j张

{

for(k = 0; k <= 100 - 2 \* i - 4 \* j; k ++) //购买第三种邮票k张

{

if( i + j + 9 \* k == 100 && 2 \* i + 4 \* j + 2 \* k == 100)

cout << " i = " << i << " j = " << j << " k = " << k << endl;

}

}

}

system("pause");

return 0;

}

[输出结果]

//所有满足条件的第一种，第二种，第三种邮票的数量

i = 14 j =14 k =8

i = 31 j = 6 k =7

17. [解析] 根据题目给的规律，第i次受伤，得到的抚恤金是2^(i-1)。该题可以利用的条件有两个：

(1) 第i次受伤得到的抚恤金是2^(i-1) 枚金币

(2) 总共的抚恤金是65535枚金币

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

//初始化的sum表示受过一次伤后总的抚恤金，t表示第一次收到的抚恤金

int i = 1,sum = 1,t = 1;

do

{

t = t \* 2; //t表示第i次收到的抚恤金

sum += t; //sum 表示总的抚恤金

i ++; //i表示第i次受伤

}while(sum != 65535);

cout << "i = " << i << endl; //输出受伤的次数

system("pause");

return 0;

}

[输出结果]

i = 16 //总共受伤的次数

18. [解析] 该题的解题思路同15，16，17题。

该题可以利用的条件有两个：

(1) 长短椅数量的范围

(2) 坐在长短椅上的学生人数

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用命名空间

int main() //主函数

{

int i,j; // i,j 分别表示长椅，短椅的张数

for(i = 0; i <= 44 / 8; i ++) //对长椅的张数进行循环

{

for(j = 0; j <= (44 - 8 \* i)/5; j ++ ) //对短椅的张数进行循环

{

if(8 \* i + 5 \* j == 44) //除班长外，总人数是44

cout << " i = " << i << " j = " << j << endl;

}

}

system("pause");

return 0;

}

[输出结果]

i = 3 j = 4 //长短椅的数量

19. (1) [解析] 该题打印下三角星型。下三角形的条件：列j>=行i

该题需要注意的地方有以下几点：

(1)每一行第一个\*号的位置

(2)每一行的\*号的个数

(3)总行数

[参考程序]

#include <iostream> //预编译命令

#include <iomanip> //格式化输入输出的头文件

using namespace std; //使用名字空间

int main() //主函数

{

int i,j,n; //i,j控制行列号，n是总的行数

cin >> n; //输入需要输出的行数

for(i = n; i >= 1; i --)

{

cout << setw(40 - i) << setfill(' '); //控制每行第一个\*的位置

for(j = 1; j <= i; j ++) //合法的列号出输出\*

cout << '\*';

cout << endl; //输出换行符

}

system("pause");

return 0;

}

(2) [解析] 该题和第(1)小题类似。

该题需要注意的地方有以下几点：

(1)每一行第一个\*号的位置

(2)每一行的\*号的个数

(3)总行数

[参考程序]

#include <iostream> //预编译命令

#include <iomanip> //格式化输入输出的头文件

using namespace std; //使用命名空间

int main() //主函数

{

int i,j,n; //i,j表示行列号，n表示总的行数

cin >> n; //

for(i = 1; i <= n; i ++)

{

cout << setw(40) << setfill(' '); //控制每一行\*的位置

for(j = n; j >= i; j --) //j控制每一行\*的个数

cout << '\*';

cout << endl;

}

system("pause");

return 0;

}

(3) [解析] 主要控制每一行\*的位置和个数。

该题需要注意的地方有以下几点：

(1)每一行第一个\*号的位置

(2)每一行的\*号的个数

(3)总行数

[参考程序]

#include <iostream> //预编译命令

#include<iomanip> //格式输出的头文件

using namespace std; //使用名字空间

int main() //主函数

{

int i,j;

for(i = 5; i >= 1; i --)

{

cout << setw(40 - i )<< setfill(' '); //定位每一行第一个\*的位置

for(j = 1; j <= 2 \* i - 1; j ++) //j控制每一行\*的个数

cout << "\*";

cout << endl;

}

system("pause");

return 0;

}

(4) [解析] 分成两部分输出，先输出菱形的上半部分，再输出下半部分。输出上下两部分的时候均需注意以下几点：

该题需要注意的地方有以下几点：

(1)每一行第一个\*号的位置

(2)每一行的\*号的个数

(3)总行数

[参考程序]

//输入行数，输出菱形

#include <iostream> //预编译命令

#include <iomanip> //格式输出的头文件

using namespace std;// 使用名字空间

int main() //主函数

{

int i,n,j; //i,j分别控制行列

cout<<"输入行数：";cin>>n; //输入n的值

for(i=-n/2;i<=n/2;i++) //菱形共n行

{

cout<<setw(40-(n/2-abs(i))); //每行第1个\*的位置

for(j=1;j<=2\*(n/2-abs(i))+1;j++) //每行\*的个数

cout<<'\*';

cout<<endl; //换行

}

system("pause");

return 0;

}

(5) [解析] 图5-5(a)该题与第(1)题类似。

该题需要注意的地方有以下几点：

(1)每一行第一个数字的位置

(2)每一行的数字的个数

(3)每一行的不同位置数字的值

(4)总行数

[参考程序]

#include <iostream> //预编译命令

#include <iomanip> //格式化输出的头文件

using namespace std; //使用名字空间

int main() //主函数

{

int i,j,n; //i控制行，j控制列，n表示总行数

cin >> n; //输入n

for(i = 1; i <= n; i ++)

{

cout << setw(40) << setfill(' '); //定位每一行\*出现的位置

for(j = 1; j <= i; j ++)

cout << i; //定位每i行每j列的值

cout << endl;

}

system("pause");

return 0;

}

#include <iostream> //预编译命令

#include <iomanip> //格式化输出的头文件

using namespace std; //使用名字空间

int main() //主函数

{

int i,j,n; //i控制行，j控制列，n表示总行数

cin >> n; //输入n

for(i = 1; i <= n; i ++)

{

cout << setw(40) << setfill(' '); //定位每一行\*出现的位置

for(j = 1; j <= i; j ++)

cout << j; //定位每i行每j列的值

cout << endl;

}

system("pause");

return 0;

}

#include <iostream> //预编译命令

#include <iomanip> //格式化输出的头文件

using namespace std; //使用名字空间

int main() //主函数

{

int i,j,n; //i控制行，j控制列，n表示总行数

cin >> n; //输入n

char c; //字符型变量申明

for(i = 1; i <= n; i ++)

{

c = i - 1 + 'a'; //确定每一行要输出的值

cout << setw(40) << setfill(' '); //确定每一行输出的位置

for(j = 1; j <= i; j ++) //确定每一行输出的列数

cout << c;

cout << endl;

}

system("pause");

return 0;

}

#include <iostream> //预编译命令

#include <iomanip> //格式化输出的头文件

using namespace std; //使用名字空间

int main() //主函数

{

int i,j,n; 控制行，j控制列，n表示总行数

cin >> n; //输入n

for(i = 1; i <= n; i ++)

{

cout << setw(40) << setfill(' '); //定位每一行\*出现的位置

for(j = 1; j <= i; j ++)

cout << char(96+i); //定位每i行每j列的值

cout << endl;

}

system("pause");

return 0;

}

(6) [解析] 该题只需将打印的星型三角形换成数字或字母。

该题需要注意的地方有以下几点：

(1)每一行第一个数字的位置

(2)每一行的数字的个数

(3)每一行的不同位置数字的值

(4)总行数

[参考程序]

#include <iostream> //预编译命令

#include<iomanip> //格式化输出头文件

using namespace std; //使用名字空间

int main() //主函数

{

int i,j,n; //i,j,n分别控制行号，列号，总行数

cin >> n; //输入n

for(i = 1; i <= n; i ++)

{

cout << setw(40 - i )<< setfill(' '); //控制每一行的起始位置

for(j = 1; j <= 2 \* i - 1; j ++) //控制输出的列数

cout << i; //第i行，第j列的值

cout << endl;

}

system("pause");

return 0;

}

#include <iostream> //预编译命令

#include<iomanip> //格式化输出头文件

using namespace std; //使用名字空间

int main() //主函数

{

int i,j,n; //i,j,n分别控制行号，列号，总行数

cin >> n; //输入n

for(i = 1; i <= n; i ++)

{

cout << setw(40 - i )<< setfill(' '); //控制每一行的起始位置

for(j = 1; j <= 2 \* i - 1; j ++) //控制输出的列数

{

if(j <= i) //分两种情况输出数值

cout << j;

else

cout << 2 \* i - j;

}

cout << endl;

}

system("pause");

return 0;

}

#include <iostream> //预编译命令

#include<iomanip> //格式化输出头文件

using namespace std; //使用名字空间

int main() //主函数

{

int i,j,n; //i,j,n分别控制行号，列号，总行数

char c; //字符型变量申明

cin >> n; //输入n

for(i = 1; i <= n; i ++)

{

c = 'A';

cout << setw(40 - i )<< setfill(' '); //控制每一行的起始位置

for(j = 1; j <= 2 \* i - 1; j ++) //控制输出的列数

{

if(j <= i) //分两种情况输出字符

{

c = i - j + 'A';

cout << c;

}

else

{

c = j - i + 'A';

cout << c;

}

}

cout << endl;

}

system("pause");

return 0;

}

(7) [解析] 与(4)类似，将\*换成数值，分两部分打印。打印每一部分

的时候，需注意以下几点：

(1)每一行第一个数字的位置

(2)每一行的数字的个数

(3)每一行的不同位置数字的值

(4)总行数

[参考程序]

#include <iostream> //预编译命令

#include<iomanip> //格式化输出头文件

using namespace std; //使用名字空间

int main() //主函数

{

int i,j,n; //i,j控制行数，列数，2\*n-1表示总的行数

cin >> n; //输入n

for(i = 1; i <= n/2 ; i ++) //先打印前n-1行

{

cout << setw(40 - i )<< setfill(' ');//定位先n-1行出现数字的第一个位置

for(j = 1; j <= 2 \* i - 1; j ++) //控制第i行第j列的值

{

if(j <= i) //分两种情况打印值

cout << j;

else

cout << 2 \* i - j;

}

cout << endl;

}

for(i = n/2+1; i >= 1; i --) //打印后n行

{

cout << setw(40 - i )<< setfill(' '); //定位后n行出现数字的第一个位置

for(j = 1; j <= 2 \* i - 1; j ++) //控制第i行第j列的值

{

if(j <= i) //分两种情况输出值

cout << j;

else

cout << 2 \* i - j;

}

cout << endl;

}

system("pause");

return 0;

}

20. [解析] 该题考察求素数的方法。素数是只能被1和自身整除的数。判断整数i是否为素数，只需看2~sqrt(i)是否能整除i即可。假如i有大于 sqrt(i)的因子，那么它的另一个因子必小于 sqrt(i)，那么计算机运算时查到这个因子时就可判断它不是素数，因此只需到平方根，而不必查到 i-1。该题程序设计时需注意以下几点：

(1) 求平方根函数sqrt需要加入该函数的头文件#include<cmath>；

(2) 程序设计时j可以从2开始，到该整数i的平方根为止，用i依次去除需要判定的整数，只要存在可以整除该数的情况，即可确定要判断的整数不是素数，否则是素数；

(3) 设置一个计数器count,初值为0，当找到一个素数的时候，计数器加1，当count = 100时，程序退出。

[参考程序]

#include <iostream> //预编译命令

#include<iomanip> //格式化输出的头文件

#include<cmath> //求平方根函数的头文件

using namespace std; //使用名字空间

int main() //主函数

{

int i,j,count = 1; //从3开始找素数，count表示当前的素数数

bool b; //布尔类型b标记整数i是否是素数

cout << setw(5) << "2"; //格式化输出第一个素数”2”

for( i = 3; ; i ++) //测试i是否是素数

{

b = true; //初始时设置为真，若i被整除，则b为假

for(j = 2; j <= sqrt(i); j ++) //判断j是否被整除

{

if(i % j == 0) //若整除，则不是素数

{

b = false; //b设为假

break; /跳出循环

}

}

if(b) //若b是素数

{

++ count; //总的素数数加1

cout << setw(5) << i; //格式化输出i

}

if(count == 100) //当总数达到100时，跳出循环

break;

}

cout << endl;

system("pause");

return 0;

}

[输出结果]

//前100个素数

2 3 5 7 ……

21. [解析] 该题要注意的一点是要找的是最小值，所以找到一个满足条件的就退出。实现该程序的时候需注意以下几点：

(1) 循环的初值i：i = 0;

(2) i需要满足的条件：i % 3 == 2 && i % 5 == 3 && i % 7 == 2

(3) 找的一个满足条件的时候就退出循环

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int i;

for(i = 0; ; i ++) //遍历i，找到满足条件的

{

if(i % 3 == 2 && i % 5 == 3 && i % 7 == 2)

{

cout << "i = " << i << ' '; //输出

break; //找到就退出

}

}

system("pause");

return 0;

}

[输出结果]

i = 24 //输出该数

22. [解析] 该求一个数n的因数有两种方法。

方法一：当j属于[1,n]时，若j能整除n,则j是n的因子,即n的因数个数增1.

方法二：当j属于[1,sqrt(n)]是，若j能整除n，且(n/j)!= j时，则i/j,j均是i的因子，n的因数个数增2。否则，j是n的因子。n的因子数增1。这里给出的参考程序使用了第一种方法，读者可以尝试第二种方法。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int n,count = 0; //n是要输入的自然数，count是它的因子数

cin >> n; //输入n

if(n == 1) count = 1; //如果n是1，则它的因子数是1

for(int i = 1; i <= n; i ++) //若n能被i整除，则它的因子数加1

{

if(n % i == 0)

count += 1;

}

cout << "count = " << count << endl; //输出总的因子数

system("pause");

return 0;

}

[输出结果]

4 //输入n

count = 3 //输出n个因子数

23. [解析] 该题只要从最小的自然数i开始遍历，求该i的因数个数。若i的因子个数等于m，则i即是满足条件的最小解。其中求自然数n的因子的方法请读者参考本章习题第25题。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int m,i,count;

cin >> m; //输入因数个数

if(m == 1) //若m等于1，则该自然数是1

cout << m << endl;

for(i = 2; ; i ++) //从2开始，找因子数是m的最小的自然数

{

count = 0;

for(int j = 1; j <= i; j ++)

{

if(i % j == 0) //j是i的因子

{

count += 1; //i的因数个数加1

}

if(count > m) //若因数个数超过m，则跳出循环

break;

}

if(count == m) //若总的因数个数是m个，则满足条件

{

cout << "i = " << i << endl;//输出i

break; //找到了最小值，跳出

}

}

system("pause");

return 0;

}

[输出结果]

5 //输入因子数目

i= 16 //输出因子数目最小的自然数

24. [解析] 注意每一项的计算办法。前后两项之差小于10-4  可以表示为fabs(前项-后项)<1e-4

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int i = 2;

double s2 = 1,s1 = 1;

double t;

double e = 2.0;

do

{

s2 = s1; //s2表示前一项

s1 = s2 \* i; //s1表示i!

t= 1 / s1; //t表示当前的每一项

e += t; //e表示和

i = i + 1; //i表示项数

}while((double)1 / s2 - (double)1 / s1 >= 1e-4); //停止条件

cout << " e = " << e << endl; //输出e

system("pause");

return 0;

}

[输出结果]

e = 2.71828 //输出e的值

25. [解析] 该题隐含的一个条件是该数是三位数，所以只要取出该数的百位，十位，个位数值，看是否满足水仙花数等式即可。

[参考程序]

#include<iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int i,a,b,c;

for(i = 100; i < 1000; i ++) //对i在其范围内进行遍历

{

a = i / 100; //由i得百位

b = (i - a \* 100) / 10; //由i得十位

c = (i % 100) % 10; //由i得个位

if(a \* a \* a + b \* b \* b + c \* c \* c == i) //水仙花数的条件

cout << i << '\t'; //满足条件的输出

}

cout << endl;

system("pause");

return 0;

}

[输出结果]

153 370 371 407 //1～1000之间的全部“水仙花数”

26. [解析] 当i属于[1,9]时,i可能出现在i的平方数的最右端的一位，当i属于[10,99]时，i可能出现在i的平方数的最右端的两位，当i=100时，i出现在i的平方数最右端的三位。所以只要遍历1~100的所有整数i，求i的平方数的最右端的后三位或者后两位或后一位等于该数，则输出该数。其中一个整数i的后一位等于i%10,后两位等于i%100,后三位等于i%1000.

[参考程序]

#include<iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int i;

for(i = 1; i <= 100; i ++)

{//判断平方后的值的后三位或者后两位或后一位等于该数

if(i \* i % 10 == i || i \* i % 100 == i || i \* i % 1000 == i )

cout << i << '\t';

}

cout << endl;

system("pause");

return 0;

}

[输出结果]

1 5 6 25 76 //1～100之间的全部“同构数”

27. [解析] 注意通项和求和。后一项的分母等于前一项的分子，后一项的分子等于前一项的分子加分母。此处，若a/b是前一项的值，则后一项的值是(a+b)/a

[参考程序]

#include<iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int i;

float t,m = 1, n = 1,s = 0;

for(i = 0; i < 20 ; i ++)

{

t = m + n; //m是当前一项的分子，n是当前一项的分母

s = s + t / n;//s表示和

m = n;//前一项的分母赋给后一项的分子

n = t;//前一项的分子赋给后一项的分母

}

cout << s << endl ;

system("pause");

return 0;

}

[输出结果]

32.6603 // 数列前20项之和

28. [解析] 该题考察学生的推理能力和循环结构。该题最普通的办法就是对A,B,C,D在其范围内使用四重循环。这里可以观察算式的特点，找到A,B,C,D的联系，降低循环。

(1) 因为A,B,C,D均为一位整数，由一位整数相减得到的差的范围是(-9,9),(B-C)最多从A中借一位，所以A = 1

(2) 由被减数，减数的个位可以知道：D = (C+C)%10,所以该题只要对B,C进行二重循环即可得到结果

[参考程序]

#include<iostream> //预编译命令

using namespace std; //使用名字空间

int main() //主函数

{

int j,k,m,i = 1;// i,j,k,m 代表A,B,C,D的值

for(j = 0; j <= 9; j ++) //对B,C进行二重循环判断

{

for(k = 0;k <= 9; k ++)

{

m = (k+k)%10;

if(1000 + j \* 100 + k \* 10 + m - (k \* 100 + m \* 10 + k) == 100 + j \* 10 + k)

cout << "A = " << i << " B = " << j << " C = "<< k << " D=" << m << endl;

}

}

system("pause");

return 0;

}

[输出结果]

A = 1 B = 0 C = 9 D = 8 // A,B,C,D的值

29. [解析] 该题的解题思路是：对一个数求所有因子，这里的因子是除其自身外的所有因数，然后检查所有因子之和是否等于该整数。查找整数i的因子一般有两种办法：

方法一：当j属于[1,i-1]时，若j能整除i,则j是i的因子。

方法二：当j属于[1,sqrt(i)]是，若j能整除i，且(i/j)!= j时，则i/j,j均是i的因子，否则，j是i的因子。这里给出的参考程序使用了第一种方法，读者可以尝试第二种方法。

[参考程序]

#include<iostream> //预编译命令

using namespace std;

int main()

{

int i,sum;

for(i = 1; i <= 1000; i ++) //求整数i的因子

{

sum = 0;

for(int j = 1; j < i; j ++)

{

if(i % j == 0 ) //若j是i的因子

{

sum += j ; //将j加到sum中

}

}

if(sum == i) //如果和等于该整数，则输出所有因子

{

cout << i << " its factors ";

for(int j = 1; j < i; j ++)

{

if(i % j == 0)

cout << j << ' ';

}

cout << endl;

}

}

cout << endl;

system("pause");

return 0;

}

[输出结果]

//所有完数和它的因子

6 its factors 1 2 3

28 its factors 1 2 4 7 14

496 its factors 1 2 4 8 16 31 62 124 248

30 [解析] 设前两位数字是i,后两位数字是j。则i属于0~9，j属于0~9。通过i,j可以得到四位数的值，然后只要判断它是否是一个完全平方数即可。

程序设计的时候需注意的一点是：由于最小的四位数的平方根是大于31的，所以对平方根的循环判断只需从31开始。

[参考程序]

#include<iostream> //预编译命令

using namespace std;

int main()

{

int i,j,k,c;

for(i = 1; i <= 9; i ++) //i是前两位数字

{

for(j = 0; j <= 9; j ++)//j是后两位数字

{

k = i \* 1000 + i \* 100 + j \* 10 + j;

for(c = 31; c \* c <= k; c ++)//判断k是否是某一个数的平方根

{

if(c \* c == k)

{

cout << "The car number is " << k << endl;

}

}

}

}

system("pause");

return 0;

}

[输出结果]

The car number is 7744

31. [解析] 用四个学生的话来统计该学生能答对的问题数，然后用四重循环枚举湖的大小，最后的限制条件是每个学生答对的问题数是一个。这样就能得到四个湖的大小顺序了。

[参考程序]

#include <iostream> //预编译命令

#include <iomanip>

using namespace std;

//四个变量分别表示洞庭湖，洪泽湖，鄱阳湖，太湖

int dongting, hongze, poyang, tai;

int A() //统计A答对的湖的大小的个数

{

int r = 0;

if (dongting == 1) r++;

if (hongze == 4) r++;

if (poyang == 3) r++;

return r;

}

int B() //统计B答对的湖的大小的个数

{

int r = 0;

if (hongze == 1) r++;

if (dongting == 4) r++;

if (poyang == 2) r++;

if (tai == 3) r++;

return r;

}

int C() //统计C答对的湖的大小的个数

{

int r = 0;

if (hongze == 4) r++;

if (dongting == 3) r++;

return r;

}

int D() //统计D答对的湖的大小的个数

{

int r = 0;

if (poyang == 1) r++;

if (tai == 4) r++;

if (hongze == 2) r++;

if (dongting == 3) r++;

return r;

}

int main()

{

//对可能出现的湖的大小进行遍历

for (dongting = 1; dongting <= 4; dongting++) {

for (hongze = 1; hongze <= 4; hongze++)

{

if (hongze == dongting) //排序一样，则冲突

continue;

for (poyang = 1; poyang <= 4; poyang++)

{

if (poyang == dongting || poyang == hongze) //排序一样，则冲突

continue;

for (tai = 1; tai <= 4; tai++)

{ //排序一样，则冲突

if (tai == dongting || tai == hongze || tai == poyang) continue;

//湖的大小满足A答对一个，B答对一个，C答对一个，D答对一个

if (A() == 1 && B() == 1 && C() == 1 && D() == 1)

cout <<setw(4) << dongting <<setw(4) << hongze <<setw(4)<< poyang <<setw(4)<< tai << endl;

}

}

}

}

system("pause");

return 0;

}

[输出结果]

2 4 1 3 //洞庭湖，洪泽湖，鄱阳湖，太湖的排名

32. [解析] 该题数据规模比较小，直接利用四重循环验证即可。程序设计时应注意以下几点：

(1) 验证定理在100以内的正确性，说明四整数的最大值是10，故程序中使用数组sq[11]存储0~10这11个整数的平方。

(2) 程序中使用变量i,j,k,p做四重循环，四者范围均是0~10，判断sq[i] + sq[j] + sq[k] + sq[p] 是否等于 n，n是100以内的整数

(3) 对于100以内的整数n，只要找到一个组合就行，所以这里设置了一个flag，初始值为1，当找到这样的组合时，flag就设为0，这样就可以避免重复验证。

[参考程序]

#include <iostream> //预编译命令

using namespace std; //使用名字空间

int main()

{

int sq[11] = {0, 1, 4, 9, 16, 25, 36, 49, 64, 81,100}; //模拟得到10以内所有整数的平方的数组

int n, i, j, k, p, flag;

for (n = 0; n <=100; n++)

{

flag = 1;

for (i = 0; i <= 10 && flag; i++) //四重循环查找

for (j = 0; j <= 10 && flag; j++)

for (k = 0; k <= 10 && flag; k++)

for (p = 0; p <= 10 && flag; p++)

if (sq[i] + sq[j] + sq[k] + sq[p] == n)

{

cout << n << " = " << i << '^'<< 2 << " + "<<j << '^'<< 2 << " + "<< k << '^'<< 2 << " + "<< p<< '^'<< 2 << endl;

flag = 0;

}

}

system("pause");

return 0;

}

[输出结果]

//101满足条件的等式

33. [解析] 本题主要对三位的整数100~999进行循环，判断该数本身的个位，十位，百位是否相等，再判断两个整数之间的个位，十位，百位是否相等，然后判断三个整数间的个位，十位，百位是否相等。该题程序设计的时候需要注意以下几点：

(1) 1到9这9个数字组成三位数，说明三位数的任何一位都不会是0，遍历100~999之间的所有三位数要除去含0的三位数。取三位数的个，十，百位上数字的值，若它们乘积为0，则说明该三位数 含有0，除去这样的三位数。

(2) 三位数各位上的数字均不相同，则三位数的范围可以缩小到123~987，这范围之外的三位数均有其中两位上的数字是相同的。

(3) 实现的时候只需对加数a,和b在123~987范围内进行循环，a，b满足本身各位上的数字不相等，且a，b之间也不相等，同时求得的和也在123~987范围内，且c各位上的数字互不相等，与a,b各位上的数字也不冲突，这样的a,b,c就是满足条件的三位数，输出即可。

[参考程序]

#include <iostream> //预编译命令

using namespace std;

int check(int n) //判断该数本身的个位，十位，百位是否相等

{

int a = n % 10;

int b = (n / 10) % 10;

int c = (n / 100) % 10;

if (a \* b \* c == 0)

return 0;

if (a == b || b == c || a == c)

return 0;

return 1;

}

int check2(int m, int n) //判断两个整数之间的个位，十位，百位是否相等

{

if(!(check(m) && check(n)))

return 0;

int a = m % 10;

int b = (m / 10) % 10;

int c = (m / 100) % 10;

int a2 = n % 10;

int b2 = (n / 10) % 10;

int c2 = (n / 100) % 10;

if (a2 == a || a2 == b || a2 == c)

return 0;

if (b2 == a || b2 == b || b2 == c)

return 0;

if (c2 == a || c2 == b || c2 == c)

return 0;

return 1;

}

int main()

{

int a, b, c;

for (a = 123; a <= 987; a++)

for (b = 123; b <= 987; b++)

{ c = a+b;

if (check2(a, b)&&check2(a,c)&&check2(b,c)&&check(c)&& c<=987)

//判断a,b,c间的个位，十位，百位是否相等

cout << a << '+' << b << '=' << c << endl;

}

system("pause");

return 0;

}

[输出结果] //所有满足等式的解，这里例举一个

386+541 = 927  
  
34. [解析] （1）对1~1000内的所有整数判断是否是回文数，然后判断是否是素数，若既是回文数又是素数，则输出该数。程序设计时需注意一下几点：

1) 判断整数i是否是回文数，只需将该整数倒置，看倒置后的整数是否和原整数相等，若相等，则是回文数

2) 判断整数i是否为素数，只需看2~sqrt(i)是否能整除i即可。假如i有大于 sqrt(i)的因子，那么它的另一个因子必小于 sqrt(i)，那么计算机运算时查到这个因子时就可判断它不是素数，因此只需到平方根，而不必查到 i-1

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

bool palindrome(int i) //判断是否是回文数

{

int x;

int s = 0;

x = i;

while(x > 0) //将该整数i倒置，存储在s中

{

s = s \* 10 + x % 10;

x = x / 10;

}

if(s == i)//从左读和从右读是同一个数，则输出

return 1;

else

return 0;

}

bool prime(int i) //判断i是否是素数

{

int flag = 0;

if(i == 0 || i == 1) return 0;

if(i == 2) return 1;

for(int j = 2; j <= sqrt(i); j ++) //j遍历2到sqrt(i),判断j是否整除i

{

if(i % j == 0)//若整除，则不是素数

{

flag = 1;

break;

}

}

if(flag == 1) return 0;

if(flag == 0) return 1;

}

int main()

{

int i;

for(i = 1; i <= 1000; i ++)

{

if(palindrome(i) && prime(i)) //i同时是回文数又是素数

cout << i << '\t';

}

cout <<endl;

system("pause");

return 0;

}

[输出结果]

//1000以内所有的既是回文数同时又是素数的自然数。

2 3 5 7 11 101 131 151 181 191

313 353 373 383 727 757 787 797 919 929

（2） [解析] 该题考察判断回文数和完全平方数的方法。对于回文数来说，左右对称只要逆序输出该整数，看是否和原整数相等。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

bool palindrome(int i) //判断是否是回文数

{

int x;

int s = 0;

x = i;

while(x > 0) //s保存逆序输出原整数

{

s = s \* 10 + x % 10;

x = x / 10;

}

if(s == i) return 1; else return 0;

}

int main()

{

long int i;

for(i = 1; i < 1000000; i ++)

{

if(palindrome(i) && (int)sqrt(i) == sqrt(i)) //是回文数且是完全平方数

cout << i << '\t';

}

cout << endl;

system("pause");

return 0;

}

（3）[问题分析] 需要最大的完全平方回文数，首先需要找到最大的完全平方数。

[算法设计] 找到最大的完全平方数后，依次从大到小，判断该数是否为回文数。

[参考程序]   
#include <iostream>

#include <cmath>

using namespace std;

int weishu(int m)

{

int k=0;

while(m!=0)

{ m=m/10;k++; }

return k;

}

bool symm(int m) // 判断整数m是否为回文数

{

int temp=m,n=0;

while(temp){

n=n\*10+temp%10;

temp=temp/10;

}

return (m==n);

}

int main()

{

cout<<"最大的完全平方回文数：";

for(int n=(int)sqrt(INT\_MAX);;n--){

if(symm(n\*n)){

cout<<n\*n<<endl;

cout<<"N="<<weishu(n\*n)<<endl;

break; }

}

cout<<endl;

system("pause");

return 0;

}

35. [解析] 该题主要考察学生的推理能力和循环结构。

程序设计的时候注意以下几点：

(1) 设i,j,k分别代表最上方，最左边，最右边三个顶点的值，sum代表每条边上的和。由于只能取1~6这6个数字，所以sum的范围是6~15。

(2)根据i,j,k,sum就可以求出其他的三个圈中的数字，根据它们互不相等即可求解。

[参考程序]

#include<iostream> //预编译命令

using namespace std;

int main()

{

int i,j,k,sum;

for(int sum = 6; sum <= 15; sum ++) //对和在范围内进行循环

{//对三个顶点进行循环

for(int i = 1; i <= 6; i ++)//

{

for(int j = 1; j <= 6 && j != i; j ++)

{

for(int k = 1; k <= 6 && k != i && k != j; k ++)

{

int m = sum - i - j;

int n = sum - i - k;

int l = sum - j - k;

if(m != n && n != l && l != m && m != i && m != j && m!= k && n!= i && n!= j && n != k && l != i && l != j && l !=k && m > 0 && m <=6 && n > 0 && n <=6 && l > 0 && l <=6)

cout << i << ' ' << j << ' ' << k << ' ' << m << ' ' << n << ' ' << l << endl;

}

}

}

}

system("pause");

return 0;

}

[输出结果]

//满足条件的组合方式

36. [解析] 最容易想到的是穷举法，桃子的数目一个个地增加，直到猴子能按条件取桃子时为止。但这样简单的穷举效率太低，我们应该尽量利用已知条件，减少穷举的计算次数。我们按拿桃子的顺序把猴子分为第一只到第八只。最后剩rest个，而rest可以均匀分为4 堆，每堆有rest/4 个桃子，这个猴子已吃过一堆和另外的一个，所以轮到第八只猴子拿桃子时，共有5\*(rest/4)+1个，即rest\*5/4+1个桃子。可利用变量total存放该值，total=rest\*5/4+1。并把total的初值设为rest，则上式改为total=total\*5/4+1。这其实是“迭代法”的算法技巧。轮到第七只、第六只、第五只、第四只猴子拿桃子时，分别在前述的total

计算基础上共有total=total\*5/4+1 个桃子。而轮到第三只、第二只、第一只猴子拿桃子时，分别在前述的total计算基础上共有total=total\*3/2+1 个桃子。最后计算出的total结果即为所求解。值得注意的是，在八次迭代计算total时，如果结果不是整数，则要抛弃。

[参考程序]

#include <iostream> //预编译命令

using namespace std;

int main()

{

long int i;

float R0,R1,R2,R3,R4,R5,R6,R7,R8;

for(i = 0; ; i ++) //迭代

{

R8 = i;

R7 = R8 / 4 \* 5 + 1;

R6 = R7 / 4 \* 5 + 1;

R5 = R6 / 4 \* 5 + 1;

R4 = R5 / 4 \* 5 + 1;

R3 = R4 / 4 \* 5 + 1;

R2 = R3 / 2 \* 3 + 2;

R1 = R2 / 2 \* 3 + 2;

R0 = R1 / 2 \* 3 + 2;

//若不是整数，则抛弃

if((int)R8 == R8 && (int)R7 == R7 && (int)R6 == R6 && (int)R5 == R5 && (int)R4 == R4 && (int)R3 == R3 && (int)R2 == R2 && (int)R2 == R2 && (int)R1 == R1 && (int)R0 == R0 )

{ cout << i << endl;

break;

}

}

system("pause");

return 0;

}

[输出结果]

8188 //剩余的桃子数目

37. [解析] 该题只要对首项在其范围内进行遍历，即可得到满足条件的解。

编程时需要注意以下几点：

(1) 题意中给定是正整数的等差数列，隐含条件是公差d>=0,故首项a1的范围是[1,26/4]；

(2)由于第二项a2 = a1 + d，第三项a3 = a2 + d，第四项a4 = a3 + d，所以4\*a1+3\*d = 26,即d = (26-4\*a1)/6。所以该题只需一重循环就可以实现。

[参考程序]

#include<iostream> //预编译命令

using namespace std;

int main()

{

int a1,a2,a3,a4,d;

int i,j;

for(i = 1;i <= (26/4); i ++) //对首项进行遍历

{

a1 = i;

a2 = a1 + d;

a3 = a2 + d;

a4 = a3 + d;

d = (26 - 4 \* a1)/6;

if(a1 + a2 + a3 + a4 == 26 && a1 \* a2 \* a3 \* a4 == 880)

cout << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << endl;

}

system("pause");

return 0;

}

[输出结果]

2 5 8 11 //等差数列

38. [问题分析] 从32到99，它们的平方都是4位数，我们只需在完全平方数中找，这样范围就很小了。

[参考程序]

#include <iostream>

using namespace std;

int main()

{

int i,x,a,b,y;

for(i=32;i<100;i++)

{

x=i\*i;

a=x/100;

b=x% 100;

y=a+b;

if(y\*y==x)cout<<x<<endl;

}

system("pause");

return 0;

}

运行结果  
2015 3015 9801

39. [解析] 对三边长进行遍历，看该三边是否满足构成直角三角形的条件。这里有两个条件：第一，两边之和大于第三边，这是构成三角形的条件。其中两边之差小于第三边可以由两边之和大于第三边推出，所以这个判定条件可以省略。

第二，两直角边的平方和等于斜边的平方，这是构成直角三角形的条件。

[参考程序]

#include<iostream> //预编译命令

using namespace std;

int main()

{

int i,j,k;

for(i = 1; i <= 50; i ++) //遍历三边长

{

for(j = 1; j <= 50; j ++)

{

for(k = 1; k <= 50; k ++)

{//判断直角边的平方和是否是斜边的平方，两边之和大于第三边

if(i \* i + j \* j == k \* k && i +j > k && j + k > i && i +k >j)

cout << "i =" << i << " j = " << j << " k = " << k << endl;

}

}

}

cout << endl;

system("pause");

return 0;

}

[输出结果]

i=48 j = 14 k=50 //满足条件的三角形三边长,未枚举所有

40. [解析] 该题主要将问题抽象给程序逻辑。每对折一次，纸的厚度变成了原来的两倍，经过i次折叠后，纸的厚度是(2^i)\*0.1。该题程序设计时需注意以下几点：

(1) 一张纸厚度是0.1毫米，珠穆朗玛峰高度是8848米，将其化为统一的进制，8848米= 8848000毫米。因为这里纸的厚度是小数，所以设为double型，初始化为0.1

(2) 经过第i次折叠后纸的厚度是(2^i)\*0.1,当(2^i)\*0.1>8848000时，此时的i已经是满足条件的折叠次数了。

[参考程序]

#include<iostream> //预编译命令

using namespace std;

int main()

{

int t = 1;

double s = 0.1;

int i = 0;

while(s < 8848000) //s表示纸当前的厚度

{

i = i + 1; //i表示折叠次数

t \*= 2; //t是2^i

s = t \* 0.1;

}

cout << "i = " << i << endl;

system("pause");

return 0;

}

[输出结果]

i=27 //折叠的次数

41. [解析] 该题递归的判断每一方格的数字是否合法的。所有方格可以视为一个3\*5的矩形，然后将需要填数的方格视为真值。往需要填数的方格里填数，同时判断是否合法，合法的情况是横，竖，对角任何两个相邻的小方格中的两个数都是不连续的。该题程序设计的时候需注意以下几点：

(1) 初始化时： 定义矩阵a[3][5],其中第0，2，4行的第0，2列上的元素设为false，其他位置元素设为true。

(2) 判断a[row][col]是否合法：只需看i属于row-1~row+1,j属于col-1~col+1时，a[row][col]和a[i][j]差的绝对值是否小于等于1，若小于等于1，则不合法，否则合法。该判定条件保证横，竖，对角任何两个相邻的小方格中的两个数都是不连续的。注意，求绝对值函数需加头文件#include<cmath>

(3) 填值时：判断a[i][j]是否为真值，且还未被填值，则将起填值为n，然后判断该填值是否合法，若合法，以n+1继续将该矩阵填值，否则该位置暂时不填充，等待下一个合法值来填充(下一次遍历)。

[参考程序]

#include<iostream> //预编译命令

#include<cmath> //求绝对值函数的头文件

#include <iomanip> //格式输出的头文件

using namespace std;

int a[5][3];

int count = 0;

bool matrix\_init(int row,int col) //矩阵的初始化，需要填数的位置设为真

{

switch(row)

{

case 0:

case 2:

case 4:

{

if(col == 1)

return true;

else

return false;

}

case 1:

default:

return true;

}

}

bool matrix\_islegal(int row,int col) //判断矩阵row行，col列是否合法

{

int i,j;

for(i = row - 1; i <= row + 1; i ++)

{

if(i < 0 || i > 4)

continue;

for(j = col - 1; j <= col + 1; j ++)

{

if(j < 0 || j > 2)

continue;

if((fabs(a[i][j] - a[row][col])) == 1 &&( i == row ||j == col || fabs(i - row) <= 1 || fabs(j - col) <= 1) ) //判断是否合法

return false;

}

}

return true;

}

void matrix\_trace(int n) //递归遍历整个矩阵

{

for(int i = 0; i < 5; i ++)

{

for(int j = 0; j < 3; j ++)

{

if(a[i][j] == -1 && matrix\_init(i,j))

{

a[i][j] = n;

if(matrix\_islegal(i,j))

//若是合法的，则用下一个数字递归执行

matrix\_trace(n + 1);

else a[i][j] = -1;

}

}

}

}

int main()

{

int i,j;

for(i = 0; i < 5; i ++)

{

for(j = 0; j < 3; j ++)

a[i][j] = -1;

}

matrix\_trace(1); //从1开始递归遍历

for(i = 0; i < 5; i ++)

{

for(j = 0; j < 3; j ++)//输出合法的小方格，-1表示该位置不用填数

cout << setw(4) << a[i][j] ;

cout <<endl;

}

system("pause");

return 0;

}

[输出结果]

//满足条件的填数方式，其中-1的位置处是不用填数的

-1 1 1

4 7 5

-1 2 -1

6 8 8

-1 3 -1

42. [解析] 用s1保存原三角形面积，s2保存分离出第K个正方形的面积，s保存剩下的面积，求满足条件的K值即可。该题程序设计时需要注意以下几点：

(1) 原三角形的面积是s1 = a\*a/2

(2) 第K次分离出的正方形面积是(a/(2^k))^2,正方形的个数是2^(k-1)个。

(3) 剩余的面积是 s1 – sum(a / (2 ^i) \* (a / (2 ^ i))\* 2^(i-1)) ,其中i属于1~k，a / (2 ^i) \* (a / (2 ^ i)\* (2^(i-1)) 表示第i次分离出的2^(i-1)个正方形的面积，sum表示i次分割的求和。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

int main()

{

float a;

int k;

cin >> a;

int i = 1;

double s1,s2,s;

s1 = a \* a / 2;

s2 = a \* a / 4;

k = 1;

s = s1 - s2;

while(s > 0.01 \* s1)

{

i ++;

s2 =s2/4; //s2是第k次分离出的1个正方形的面积

k \*= 2; //k表示第k次分离出的正方形的个数

s -= k\*s2; //第k次分离后剩余的面积

}

cout << "i = " << i << " 边长是" << sqrt(s2)<< " 面积是" << s2 << endl;

system("pause");

return 0;

}

[输出结果]

4 //输入a的值

i=7 边长是0.03125 面积是0.000976563 //输出k的值，边长和面积

43. [解析] 为提高速度,在测试过程中,应该优先排除那些可以快速判断的条件,例如,商的右边第四位为7,m/n的余数应该为0,商的长度为5,此类条件,可以优先排除,达到加速的效果。

[参考程序]

#include <iostream>

#include <cmath>

using namespace std;

int calen(int x) //计算并返回整数x的位数

{

int len = 0;

do

{

len++;

x /= 10;

}

while (x > 0);

return len;

}

int cut(int m, int x) //将整数m除以10的x次方并返回结果

{

while (m > 0 && x > 0)

{

m /= 10;

x--;

}

return m;

}

int gendiv(int m, int n, int\* mx, int\* nx) //将m/n的中间结果保存在两个数组mx和nx中

{

int r = m / n;

int len = calen(r);

int i;

for (i = 0; i < len; i++)

nx[i] = (cut(r, len - 1 - i) % 10) \* n;

mx[0] = cut(m, len - 1);

for (i = 1; i < len; i++)

mx[i] = (mx[i - 1] - nx[i-1]) \* 10 + (cut(m, len - 1 - i) % 10);

mx[len] = m - r \* n;

return len;

}

int check(int m, int n, int\* mx, int\* nx) //判断mx与nx是否符合原题的要求

{

#define BETWEEN(x,a,b) (x>=a && x<=b)

return !nx[3] && !mx[5]

&& BETWEEN(nx[0], 1000, 9999)

&& BETWEEN(nx[1], 100, 999)

&& BETWEEN(nx[2], 100, 999)

&& BETWEEN(nx[4], 1000, 9999)

&& BETWEEN(mx[1], 100, 999)

&& BETWEEN(mx[2], 1000, 9999)

&& BETWEEN(mx[3], 100, 999)

&& BETWEEN(mx[4], 1000, 9999);

}

int main()

{

int m, n, r, f = 0;

int mx[16] = {0}, nx[16] = {0};

for (m = 10000000; m < 99999999; m++)

{

for (n = 100; n < 999; n++)

{

if (m % n)

continue;

r = m / n;

if (7 != (r / 1000) % 10)

continue;

if (calen(r) != 5)

continue;

gendiv(m, n, mx, nx);

if (check(m, n, mx, nx))

{

cout << m << '/' << n << '=' << r << endl;

f = 1;

break;

}

}

if (f)

break;

}

system("pause");

return 0;

}

[输出结果]

12128316/ 124 = 97809

44. [解析] 遍历10~1000内的所有奇数i，看i是否是2~i中间的三个素数之和。该题需要注意的地方有三点：

(1) 验证1000以内的验证哥德巴赫猜想，所以需验证10~1000的所有奇数i，总共的奇数个数是(1000-10)/2 = 495，所以用个计数器count记录10~1000个奇数中满足条件的个数。

(2) 因为分拆方式不唯一，对于奇数i，只要找到一种 满足方式的组合就可以了，所以用数组tag[i]表示i是否已经找到满足方式的组合，若已经找到，就不需要再继续了。

(3) 判断整数j是否为素数，只需看2~sqrt(j)是否能整除j即可。，假如j有大于 sqrt(j)的因子，那么它的另一个因子必小于 sqrt(j)，那么计算机运算时查到这个因子时就可判断它不是素数，因此只需到平方根，而不必查到 j-1。

[参考程序]

#include<iostream> //预编译命令

#include <cmath>

using namespace std;

bool prime(int i) //求素数

{

int flag = 0;

if(i == 0 || i == 1) return 0;

if(i == 2) return 1;

for(int j = 2; j <= sqrt(i); j ++)

{

if(i % j == 0)

{

flag = 1;

break;

}

}

if(flag == 1) return 0;

if(flag == 0) return 1;

}

int main()

{

int i,count;

count = 0;

int tag [1000] = {0};

for(i = 10; i < 1000 && tag[1] == 0; i = i + 2)//遍历10~1000的所有奇数

{

for(int j = 2; j <= i ; j ++) //判断i是否是2~i的三个素数之和

{

if(tag[i] == 1)

break;

if(! prime(j) )

continue;

for(int k = 2; k <= i && prime(k); k ++)

{

if(! prime(j) || !prime(k))

continue;

if((i - j - k) > 0 && prime(i - j - k))

{

cout << i << " = " << j << " + " << k << " + " << i - j - k << endl;

count = count + 1;

tag[i] = 1;

}

else

continue;

}

}

}

cout << " i = " << i << " count = " << count << endl;

system("pause");

return 0;

}

[输出结果] //所有满足条件的等式

i= 1000 count =495 //i表示1000以内，495表示等式数目

45. [算法分析] 从m-n-1到m-1共n项，满足小于m前的n个自然数的要求。

[参考程序]  
#include <iostream>

using namespace std;

int main()

{

int m,n,i,sum=0;

cout<<"input m,n";cin>>m>>n;

cout<<endl;

for(i=m-n-1;i<m;i++)

sum+=i;

cout<<sum<<endl;

system("pause");

return 0;

}

# **函数**

1. [解析] X^N = X^(N-1) \*X,其中X^(N-1)是原问题的一个子问题

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

int recursion(int n,int x)

{

if(n == 1) //递归结束条件

return x;

else

return x \* recursion(n - 1,x ); //递归表达式

}

int main()

{

int X,N;

cin >> X >> N;

int result = recursion(N,X);

cout << result << endl;

system("pause");

return 0;

}

[输出结果]

3 3 //输入X,N的值

27 //输出X^N

2. [解析] 按照给定的表达式得到递归结束条件和递归方程。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

int polynomial(int n, int x)

{

if(n == 0) //递归结束条件

return 1;

else if(n == 1) //递归结束条件

return x;

else //递归方程

return (2 \* n - 1) \* polynomial(n - 1,x) - (n - 1) \* polynomial(n - 1, x) / n;

}

int main()

{

int N,X;

cin >> N >> X;

int result = polynomial(N,X);

cout << result << endl;

system("pause");

return 0;

}

[输出结果]

3 4 //输入X,N的值

44 //勒让得多项式的值

3. [解析] 递归方程N! = (N-1)!\*N;

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

int factorial(int i)

{

if( i == 1) //递归结束条件

return 1;

else

return i \* factorial(i - 1); //递归方程

}

int main()

{

int sum;

sum = factorial(10) + factorial(7) + factorial(8);

cout << "sum = " << sum << endl;

system("pause");

return 0;

}

[输出结果]

sum=3674160 //10!+7!+8!的值

4. [解析] 倒序输出字符串s[n],递归输出后n-1长度的字符串，打印出该串的第一个字符。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

void inversion( char \*str)

{

if(\*str != '\0')

{

inversion(str + 1); //递归方程

cout << \*str;

}

}

int main()

{

char ch[256] = {0};

gets(ch); //输入需要倒序输出的字符串

inversion(ch);

cout << endl;

system("pause");

return 0;

}

[输出结果]

happy day //输入的字符串

yad yppah //倒序的结果

5. [解析] 打印乘法表的程序可以参考习题5.11,5.12，该题在原来基础上每行加了一横线。

[参考程序]

#include <iostream> //预编译命令

#include<iomanip>

using namespace std;

int main()

{

cout<<setw(3)<<"\*"<<setw(3)<<"|";//打印乘法表的开始的\*|

for( int j = 1; j < 10; j ++)

cout << setw(6) << j ; //打印第0行的第1~9列

cout << endl;

//第一行后面加一横线

cout << "----------------------------------------------------------------" << endl;

for(int i = 1; i < 10; i ++)//1~9行后面均加横线

{

cout<<setw(3)<<i<<setw(3)<<"|";

for(int j = 1; j < 10; j ++)

cout<<setw(6)<<j \* i; //打印第1~9行的第1~9列

cout << endl;

cout << "------------------------------------------------------------" << endl;

}

cout << endl;

system("pause");

return 0;

}

6.

[解析] 该题用递归方法求解一个数的整数部分和小数部分。当一个整数x-1 < 1e-6，x的整数部分等于x-1的整数部分加1，x的小数部分等于x-1的小数部分。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

#include<iomanip>

using namespace std;

int partInterger(double x)

{

if(x - 1 < 1e-6)

return 0;

else// x的整数部分等于x-1的整数部分加1

return partInterger(x - 1) + 1;

}

double partDecimal(double x)

{

if(x - 1 < 1e-6)

return x;

else// x的小数部分等于x-1的小数部分

return partDecimal(x - 1);

}

int main()

{

double number1,number2;

cin >> number1>>number2;

cout << "The Interger part is " << partInterger(number1) << setw(6)<< partInterger(number2)<< endl;

cout << "The Decimal part is " << partDecimal(number1) << setw(6)<< partDecimal(number2)<< endl;

system("pause");

return 0;

}

[输出结果]

1.234 2.345 //输入任意两个实数

The interger part is 1 2 //输出整数部分

The Decimal part is 0.234 0.345 //输出小数部分

7. [解析] 用递归函数求解,N个数的最大值，等于第1个数和后N-1个数的最大值中取较大者。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

int max(int\* arr, int len)

{

if(len == 1)

return arr[len-1];

else

{

int m = max(arr+1, len-1); //求后len-1个数的最大值

return arr[0] > m ? arr[0]:m;//第一个数和后len-1个数最大值中的较大者

}

}

int main()

{

int N;

cin >> N;

int \*arr = new int[N];

for(int i = 0; i < N; i ++) cin >> arr[i];

cout << max(arr,N) << endl;

system("pause");

return 0;

}

[输出结果]

5 //输入N，这里要求的是5个数

1 2 8 3 0 //输入5个数

8 //输出最大值

8. [解析] 例如S＝｛1，2，3，4｝，k＝3。细心的读者稍加分析后，不难得出S有6 种不同的划分方案，即划分数为6。其方案为

｛1，2｝∪｛3｝∪｛4｝ ｛1，3｝∪｛2｝∪｛4｝

｛1，4｝∪｛2｝∪｛3｝ ｛2，3｝∪｛1｝∪｛4｝

｛2，4｝∪｛1｝∪｛3｝ ｛3，4｝∪｛1｝∪｛2｝

如果对于任意的S集合和k值，就不能凭籍直觉和经验计算划分数和枚举划分方案了。必须总结出一个数学规律：设n个元素a1…an放入k个无标号盒的划分数为S(n，k)。在配置过程中，有两种互不相容的情况。

(1) 设｛an｝是k个子集中的一个子集，于是把｛a1…an-1｝划分为k－1子集有S(n－1，k－1)个划分数;

(2) 如果｛an｝不是k个子集中的一个，即an必与其它的元素构成一个子集。首先把{a1，…，an-1｝划分成k个子集，这共有S(n－1，k)种划分方式。然后再把an加入到k个子集中的一个子集中去，这有k种加入方式。对于每一种加入方式，都使集合划分为k个子集，因此由乘法原理知，划分数共有 k·ｓ(n－1，k)。

应用加法原理于上述两种情况，得出｛a1，…，an｝划分为k个子集的划分数:

S(n，k)＝S(n－1，k－1)+k·S(n－1，k)(n〉1，k≥1)

下面，我们来确定s(n,k)的边界条件:

(1) 我们不可能把n个元素不放进任何一个集合中去，即ｓ(n，0)＝0;也不可能在不允许空盒的情况下把n个元素放进多于n的k个集合中去，即k＞n时S(n，k)＝0。

(2) 把n个元素放进一个集合或把n个元素放进n个集合，方式数显然是1。即

S(n,1)=1 S(n,n)=1

显然，通过上述分析可得出划分数S(n，k)的递归关系式:

S(n，k)＝S(n－1，k－1)+k·S(n－1，k) (n〉k，k≥1)

S(n，k)＝0 (n〈k)或(k＝0〈n)

S(n，k)＝1 (k=1)或(k＝n)

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

long s(int n, int k)

{

if(n < k || k == 0) //边界条件

return 0;

else if(k == 1 || k == n) //边界条件

return 1;

else if(n > k && k > 0)

return s(n -1, k -1) + k \* s(n - 1,k); //递归方程

}

int main()

{

int N,K;

cin >> N >> K;

cout << s(4,3) << endl;

system("pause");

return 0;

}

[输出结果]

4 3 // 输入N,K

6 //输出划分的个数

9. [解析] 递归是从函数(过程)自身出发来达到边界条件，在通过边界条件的递归调用过程中，系统用堆栈把每次调用的中间结果（局部变量和返回地址）保存起来，直至求出递归边界值。然后返回调用函数。返回的过程中，中间结果相继出栈恢复递归算法的效率往往很低，费时和费内存空间。对求解某些问题时,我们希望用递归算法分析问题,用非递归算法具体求解问题。这就需要把递归算法转换为非递归算法。

把递归算法转化为非递归算法有如下三种基本方法：

(1) 通过分析，跳过分解过程，直接用循环结构的算法实现求值过程。

(2) 自己用栈模拟系统的运行时栈,通过分析只保存必须保存的信息,从而用非递归算法替代递归算法。

(3) 利用栈保存参数,由于栈的后进先出特性吻合递归算法的执行过程,因而可以用非递归算法替代递归算法。

[参考程序]

递归算法：

#include <iostream> //预编译命令

using namespace std;

int AKM(int,int);

int main()

{ int m,n;

cin>>m>>n;

cout<<AKM(m,n)<<endl;

system("pause");

return 0;

}

int AKM(int m,int n)

{

int k;

if (m==0)

return n+1;

else

{

if (n==0)

return(AKM(m-1,1)); //递归方程

else

return(AKM(m-1,AKM(m,n-1))); //递归方程

}

}

非递归算法：这里用循环结构的算法实现求值过程。

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

#define M 3000

int akm[M][M];

int Ackerman(int m, int n)

{

int i, j;

memset(akm, 0, sizeof(akm));

for (j = 0; j < M; j ++)

akm[0][j] = j + 1;

for (i = 1; i < M; i ++)//对第一个参数进行循环

{

akm[i][0] = akm[i - 1][1];

for (j = 1; j < M; j ++)//对第二个参数进行循环

akm[i][j] = akm[i - 1][akm[i][j - 1]];

}

return akm[m][n];

}

int main()

{

int S,T;

cin >> S >> T;

cout << Ackerman(S,T) << endl;

system("pause");

return 0;

}

[输出结果]

3 4 // 输入m,n的值

125 // 输出Ack(m，n)的值

10. [解析] 根据整数转化为二进制数留整取余的办法转换为递归函数。

[参考程序]

#include<iostream> //预编译命令

using namespace std;

int \*b;

int i;

int binary(int a)

{

if(a!= 0)

{

b[i] = a %2; //数组b中存放转换后的结果

i++;

binary(a/2); //递归函数

}

}

int main()

{

b = new int[32]; //申请新数组b存放计算后得到的二进制

memset(b,0,sizeof(b)\*32); //数组的初始化

int N;

i = 0;

cin >> N; //输入需要转化的整数

binary(N);

for(int j = i- 1; j >=0; j --) //逆序输出

cout << b[j];

cout << endl;

system("pause");

return 0;

}

[输出结果]

7 // 输入要转换的整数

111 // 输出转换的结果

11. [解析] 该题的通项类似于幂次，用递归函数求解。用t保存当前通项的值，则后一项是t/x, funcb(x,t)表示第一项为t的等比数列的求和结果，则funb(x,t) = t + funb(x,t/x)。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

double funb(double x,double t) //第一项为t的等比数列的求和结果

{

if(t<=0.000001)

return 0.000001; //递归停止条件

else

return t+ funb(x,t/x); //递归方程

}

int main()

{

int x;

cin >> x;

cout << funb(x,1) << endl;

system("pause");

return 0;

}

[输出结果]

5 // 输入x的值

1.25 // 输出求和的结果

12. [解析] 根据给定的公式，找到递归的边界条件和递归方程。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

int fun(int n)

{

if(n==1||n==0) return 1; //递归的边界条件

return fun(n-1)+fun(n-2); //递归方程

}

int main()

{

int N;

cin >> N;

cout << fun(N) << endl;

system("pause");

return 0;

}

[输出结果]

5 // 输入N的值

8 // 输出F(n)的值

13. [解析] 根据给定的公式，找到递归的边界条件和递归方程。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

double hfun(int n,double x)

{

double a,b;

if(n == 0) return 1; //递归边界条件

if(n == 1) return 2 \* x; //递归边界条件

a = 2\*x; b = 2\*(n-1);

return a \* hfun(n-1,x)-b\*hfun(n-2,x); //递归方程

}

int main()

{

int N;

double X;

cin >> N >> X;

cout << hfun(N,X) << endl;;

system("pause");

return 0;

}

[输出结果]

4 3 // 输入N,X的值

876 // 输出Hn(x)的值

14. 请用递归算法，求勒让德多项式的值，递归公式为：  
 1 当n=0时

P(x)= x 当n=1时

((2x-1)Pn-1(x)x-(n-1)Pn-2(x))/n 当n>1时

[解析] 根据给定的公式，找到递归的边界条件和递归方程。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

double pfun(int n,double x)

{

double a,b;

if(n==0) return 1; //递归边界条件

if(n==1) return x; //递归边界条件

a=(2\*x-1)\*x; b=n-1;

return (a\*pfun(n-1,x)-b\*pfun(n-2,x))/n; //递归方程

}

int main()

{

int N,X;

cin >> N >> X;

cout << pfun(N,X)<< endl;

system("pause");

return 0;

}

[输出结果]

4 5 // 输入N,X的值

18778.5 // 输出P(x)的值

15. [解析] 该函数是等比数列的求和。设当前通项是t，则后一项是t\*x，函数f(t,x,n)表示当前项为t，公比为x,总项数是n的求和函数，易知递归公式是f(t,x,n) = t+f(t\*x,x,n-1)，则原问题转换为第一项为X，公比是X，总项数是N项的等比数列求和。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

double f(double t,double x,int n)

{

if(n>=1)

return t + f(t\*x,x,n-1);

else

return 0.0;

}

int main()

{

int X,N;

cin >> X >> N;

//第一项为X，公比是X，总项数是N项的等比数列求和

cout << f(X,X,N) << endl;

system("pause");

return 0;

}

[输出结果]

4 5 // 输入N,X的值

1346 // 输出Fn(x)的值

16. [解析] 玩九连环你就会发现存在这样一个规律：

（1）第 1 环可以自由上下

（2）而上/下第 n 环时（n>1),则必须满足：

（a）第 n-1 个环在架上

（b）前 n-2 个环全部在架下

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

int times = 0;

void off(int);

//装环过程

void on(int n)

{

if(n == 1) //当只有一个环的时候直接装上就可以

{

cout << "1 on" << endl;

times ++;

return;

}

if(n == 2) //当要装两个环的时候需要先装第1个环，再装第二个环

{

cout << "1 on" << endl;

times ++;

cout << "2 on" << endl;

times ++;

return;

}

//当要装n个环的时候，需先装上n-1个环，然后卸掉n-2个环，再装上第n个环，最后再将n-2个环装上

on(n - 1);

off(n - 2);

cout << n << "on" << endl;

times ++;

on(n - 2);

}

//卸环过程

void off(int n)

{

if(n == 1) //只有1个环的时候，直接卸掉就可以

{

cout << "1 off" << endl;

times ++;

return;

}

if( n == 2) //当有两个环的时候，先卸掉第2个环，再卸掉第1个环

{

cout << "2 off" << endl;

times ++;

cout << "1 off" << endl;

times ++;

return;

}

//当有n个环的时候，需卸掉n-2个环，在卸掉第n个环，再将n-2个环装上，卸掉n-1个环

off(n - 2);

cout << n << "off" << endl;

times ++;

on(n - 2);

off(n - 1);

}

int main()

{

int n;

cin >> n;

off(n);

cout << times << endl;

system("pause");

return 0;

}

[输出结果]

2 //输入N

2 off //拆下第N个环的过程

1 off

2 //总共的步骤

17. [解析]

可得递推关系

6 当n=6

f(n) = 7 当n=7

8 当n=8

经过分析就可以得出：

（1）按n=4的方法将1个小正方形分成4个，则增加了3个正方形。

（2）以n=6为基础，由（1）可以得出n=9，12，15，⬝⬝⬝⬝⬝⬝。

（3）以n=7为基础，由（1）可以得出n=10，13，16，⬝⬝⬝⬝⬝⬝。

（4）以n=8为基础，由（1）可以得出n=11，14，17，⬝⬝⬝⬝⬝⬝。

由此可以得出如下的递归算法：

void devide(int i);

{

if (n>8)

{

分解成四小块；

对于其中一块devide(n-3)；

}

else

switch（n)

{

case 6: 按n=6分割；

case 7: 按n=7分割；

case 8: 按n=8分割；

};

}

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

void divide(int i)

{

if(i > 8)

{

cout << "分解成3小块钢板" << endl;

divide(i - 3);

}

else

{

switch(i)

{

case 8: cout << "分解成1个大的和7个小正方形钢板" << endl; break;

case 7: cout << "分解成3个较大的和4个较小的正方形钢板" << endl; break;

case 6: cout << "分解成1个大的正方形加5个小正方形钢板" << endl; break;

case 4: cout << "分解成4个小正方形钢板" << endl;break;

default: cout << "无解" << endl;

}

}

}

int main()

{

int n;

cin >> n;

divide(n);

cout << endl;

system("pause");

return 0;

}

[输出结果]

9 //输入N

分解成3小块钢板 //分割的结果

分解成1个大的正方形加5个小正方形钢板 //分割的结果

# 数组

1.(1) [解析] 数组元素是组成数组的基本单元。数组元素也是一种变量， 其标识方法为数组名后跟一个下标。 下标表示了元素在数组中的顺序号。数组元素的一般形式为： 数组名[下标] 其中的下标只能为整型常量或整型表达式。如为小数时，C++编译将自动取整。例如，a[5],a[i+j],a[i++]都是合法的数组元素。 数组元素通常也称为下标变量。必须先定义数组， 才能使用下标变量。在Ｃ++语言中只能逐个地使用下标变量， 而不能一次引用整个数组。 例如，输出有10 个元素的数组必须使用循环语句逐个输出各下标变量：  
for(i=0; i<10; i++)　　cout<< a[i]; 而不能用一个语句输出整个数组，下面的写法是错误的： cout<< a;

[答案] C

(2) [解析] C++中，引用数组元素的下标是整型常量或者是整型表达式，且应该使用中括号，小括号是非法的。所以A是错误的。申明一个数组的长度必须用整型常量，所以B,C是错误的，这两个选项数组长度n都是整型变量。D使用了宏定义，定义了常量SIZE,故选D。

[答案] D

(3) [解析] 因为定义的数组包含10个元素，对数组元素引用的小标是从0到9，显然a[10]超出了数组的范围，所以A是错误的引用。因数组下标的数据类型必须是整型，所以B是错误的。数组引用应使用中括号，小括号是非法的，所以C也是错误的，故正确答案是D，即引用a[0]。

[答案] D

(4) [解析] 对数组元素的初始化可以通过以下方法实现：

① 在定义数组时对数组元素赋以初值，将数组元素的初值依次放在一个大括号中，如：int a[5] = {1,2,3,4,5};经过上面的定义和初始化后，a[0] = 1,a[1] = 2,a[2] = 3,a[3] = 4,a[4] = 5。

② 初始化时只给一部分元素赋值

例如：int a[5] = {1,2};定义数组a有5个元素，但大括号内只提供2个初值，没有初始化的元素值为0，即a[0] = 1,a[1] = 2,a[2]= 0,a[3] = 0,a[4] = 0.故A是正确的。

③ 在对全部数组元素进行初始化时，可以不指定数组长度，初值的个数即为数组的长度。C实际上只定义了长度为1的数组，是没有实际意义的。

B没有给数组赋初值，是错误的，D选项10\*1是表达式，不是数，也不符合初始化定义。故答案是A。

[答案] A

(5) [解析] 二维数组元素的表示形式为：数组名[下标][下标]，下标可以是整型常量或者是整型表达式，其范围是0~常量表达式-1。故A选项是错误的，a[2][4]第二维的下标超过了范围，二维数组下标不能使用逗号表达式，故B是错误的。数组下标应使用中括号，而不能用小括号，故C是错误的，D选项是正确的，元素a[0]其实是一个一维数组的头指针，包括4个元素，a[0][0],a[0][1],a[0][2],a[0][3],二维数组a其实可以看成是一个一维数组，包含元素a[0],a[1],a[2]，\*a[0]是指元素a[0][0]的值。

[答案] D

(6) [解析] 可以用以下方法对二维数组元素初始化：

① 分行给二维数组赋初值

② 可以将所有数据写在一个大括号内，按数组排列的顺序对各个元素赋初值

③ 可以对部分元素赋初值，没有初始化的元素值为0

④ 如果对全部数组元素赋初值，则定义数组时对第一维的长度可以不指定，但第二维的长度不能省略。

故A选项是错误的，因为没有指定第二维的长度，C定义了一个2\*4的二维数组，但初始化数组的行数是3行超过了范围，所以是错误的。B定义了一个2\*3的数组，是正确的，D中初始化列表有一行是空的，这在C语言中是不允许的，所以是错误的。

[答案] B

(7) [解析] A选项定义了一个2\*3的二维数组，并给数组的所有元素赋初值为0。 B指定了数组的列长度，并给数组部分元素赋初值，C选项定义了2\*3的二维数组，但初始化数组的行数是行，超过了行的范围，故是错误的。D选项指定了数组的列长度是3，初始化了2行3列的数组元素。

[答案] C

(8) [解析] int a[3][4]定义了一个3\*4的二维数组，并给数组的所有元素赋初值为0。

[答案] D

(9) [解析] int a[][4]初始化了一个4列的数组，初始化数组的行数是1行，故a是一个1\*4的二维数组，且数组的每个元素都能得到初值0。

[答案] D

(10) [解析] 对于元素a[i][j],其前面有i行，本行前面有j列，所以其前面的元素个数为i\*m+j，故a[i][j]在数组中的位置是i\*m+j+1。

[答案] D

(11) [解析] 根据二维数组的列数是3，即二维数组a每一行有3个元素，而赋初始值有7个元素，所以其行数应为3。

[答案] B

(12) [解析] 对数组赋初值时，初值的个数不能超过数组的大小。因为B中定义的数组y的大小是5，而赋初值时包含了6个值，所以B定义是非法的。

[答案] B

(13) [解析] 定义了一个3\*3的数组，k取值范围是0~2，当k = 0时，a[k][2-k] 指向元素a[0][2] = 3,当k=1时，a[k][2-k] 指向元素a[1][1] = 5, 当k = 2时，a[2][0] 指向元素a[0][2] = 7。

[答案] A

(14) [解析] 字符数组初始化的时候需要注意以下几点：

① 如果字符个数大于数组长度，系统会提示用户语法错误；   
② 用字符串初始化字符数组时，系统会在字符数组的末尾自动加上一个字符'\0'。因此，要考虑数组的长度比实际字符的个数大1；   
③ 用字符串初始化一维字符数组时，可以省略花括号｛｝。

A选项s的长度是4，因为用字符串初始化字符数组时，系统会在字符数组的末尾自动加上一个字符'\0'。因此，要考虑数组的长度比实际字符的个数大1。B选项用字符‘A’,‘B’,‘C’初始化字符数组s，s的长度是3,C选项用空格串初始化数组s，故s的长度是2.

D选项初始化数组中有字符6个，超过了字符数组最大能容纳的字符个数

[答案] D

(15) [解析] 字符串以'\0'做为字符串的结束符，故字符数组中的元素是ab.

[答案] B

(16) [解析] 根据二维数组的列数是3，即二维数组a每一行有3个元素，而赋初始值有4个元素，所以其行数应为2。二维数组初始化过程中，指定值的位置按初始化值进行赋值，未指定值的位置赋初值为0.该数组在这种初始化方式中所有元素都能得到初始0，故A选项是错误的，C选项int a[0][3]二维数组的行数是0行，是不正确的，应该是等价于 int a[1][3] = {0};D也是不正确的，这里所有元素都能得到初值0.

[答案] B

(17) [解析] A选项是不正确的，因为gets函数的参数是一个，表示要输出的字符数组名，这里有两个参数，是错误的；D选项中第一条gets(“a”)后面没有分号，是错误的；C选项输入提示符的方向不正确，该题正确答案是B。

[答案] B

(18) [解析] strcpy(a,b);将字符串b(第一个‘\0’之前的所有元素)复制到字符串a，这里必须保证字符串a的长度是大于等于字符串b的长度，strcpy(a,b)执行之后，char a[7] = “ABCdef”,则a[5] = f,故选D。

[答案] D

(19) [解析] \0表示字符串的结束，故字符串c只有3个字符：\t,\v,\\。

[答案] B

(20) [解析] 字符串比较只能用字符串比较函数strcmp,故D是正确的，strcpy(a,b)是将字符串b复制给字符串a

[答案] D

(21) [解析] 比较字符串a,b用字符串比较函数strcmp(a,b)，当a和b相等时strcmp(a,b) = 0；当a>b时，strcmp(a,b) > 0;当a < b时,strcmp(a,b) <0，故D是正确的，B选项中a>b或者a < b, strcmp(a,b)均是非0值，C选项中如果strcmp(b，a)>0是真值，则b>a。

[答案] D

(22) [解析] 字符串比较的规则如下：字符串比较的规则与其他语言中的规则相同,即对两个字符串自左至右逐个字符相比较,,(按ASCII码值进行比较,),直到出现不同的字符或遇到 '\0 '为止.如果全部字符相同的话.则认为相等,若出现不相同的字符串,则以第一个不相同的字符比较结果为准。按照该规则，字符串个数并不能做为字符串比较的标准，所以A,B是错误的。

C前一个串末尾有一个空格，后一个没有，故不是相等的，所以C是错误的。

D中字符串“That”和“The”前两个字符是相等的，第三个字符“That”中的a小于“The”中的e，所以字符串“That”小于字符串“The”。故D是正确的。

[答案] D

(23) [解析] 不可以采用如下的赋值方式：

char a[6] = “happy”;

char b[5] = “nice”

a= b;

上面赋值语句是错误的，C++字符串赋值要用strcpy(a,b);

[答案] C

(24) [解析] 上述代码首先对字符数组ch赋初值“12ab56”

第一次执行for语句：当i = 0时，ch[0] = ‘1’, 满足ch[i]>'0'&&ch[i]<='9' , 执行语句 s=10\*s+ch[i]-‘0'，s = 10\*0+‘1’-‘0’= 1；

第二次执行for语句：当i= 2时，ch[0] = ‘a’,不满足ch[i]>'0'&&ch[i]<='9',结束循环，故s = 1。

[答案] A

(25) [解析] 该题的核心代码在for语句.

第一次执行for语句： k = 2,str[2] = ‘S’,switch语句中执行default:输出S,

第二次执行for语句： k=3，str[3] = ‘W’, 执行default:输出W,

第三次执行for语句： k=4, str[4] = ‘L’, 执行case ‘L': continue;跳出该语句，第四次执行for语句： k=5, str[5] = ‘I’, 执行“case ‘I' : ++k； break ”； k=6,遇到break,跳出for循环。

执行 putchar(‘\*')，输出\*，故最终输出的结果是SW\*。

[答案] B

2. (1) [解析] C/C++中多维数组都是以行的形式连续地分布在内存中的，举个例子： int a[2][2]; 则其在内存中的表现方式为：

a[0][0], 低地址

a[0][1],

a[1][0],

a[1][1], 高地址

[答案] 按行连续存放

(2) [解析] 数组x[3][5]定义了3\*5的二维数组，其中行坐标的范围是0~2,列坐标的范围是0~4。

[答案] 2，4

(3) [解析] 二维数组初始化过程中若只给部分元素赋初值，其他为赋初值的元素均初始化为0。数组a所有元素的值如下：

int a[3][4] = {{1,2,0},

{0,0,0},

{4,6,8,10}}

易知a[1][2]=0,a[2][1]=6。

[答案] 0，6

(4) [解析] 注意：① \这个操作符是转义符， 表示将后面的特殊字符作为普通字符处理

(2) \本身也是个特殊字符，要想在字符串中包括\也要对其使用转义

(3) 有些转义是固定的：\n回车换行，\\反斜杠'\'，\" 引号，\0 转义的字符串结尾符号0，\t制表符

字符串”ab\n\\012\\”实际上是：ab

\012\

加上换行符总长度是8.

[答案] 8

(5) [解析] 第一次执行while: i = 1，x[1]! = ‘\0’,满足条件，且x[0] == ‘t’,输出‘h’

第二次执行while: i = 2，x[2]! = ‘\0’,满足条件，但x[1] !=‘t’,故不输出

第三次执行while: i = 3，x[3]! = ‘\0’,满足条件，但x[2] != ‘t’,故不输出

第四次执行while: i = 4，x[4]! = ‘t’,满足条件，但x[3] != ‘t’,故不输出

第五次执行while: i = 5，x[5]! = ‘\0’,满足条件，且x[4] == ‘t’,输出e

后面的几次循环中，均没有字符为t，故不输出任何字符。所以本题输出的结果是he

[答案] he

(6) [解析] C++输入字符串主要有两种方式：

方式一：gets(s1)；//读取字符串直到换行符或EOF停止

方式二：getline(cin, 1); //读取整行输入

[答案] gets(s1)

(7) [解析] :字符串复制不能用赋值操作符=，必须用字符串复制函数strcpy(s2,s2)。

[答案] strcpy(s2,s1);

(8) [解析] strcat (字符串1，字符串2)

【1】头文件  
【2】原型:   char\* strcat(char \* str1,char \* str2);   
【3】功能:   把字符串str2接到str1后面,str1最后的'\0'被取消   
gets（字符指针）

【1】头文件：[stdio.h](http://baike.baidu.com/view/538727.htm)（c中），c++不需包含此头文件

【2】原型：char \*gets( char \*buffer );

【3】功能：从stdin流中读取字符串，直至接受到换行符或EOF时停止，并将读取的结果存放在buffer指针所指向的字符数组中。换行符不作为读取串的内容，读取的换行符被转换为null值，并由此来结束字符串。

【4】注意：本函数可以无限读取，不会判断上限，所以程序员应该确保buffer的空间足够大，以便在执行读操作时不发生溢出。如果溢出，多出来的字符将被写入到缓冲区后面的内存位置，这将破坏一个或多个不相关变量的值。

[答案] #include<string>,#include< stdio .h>

(9) [解析] C++中,数组元素的下标总是从0开始,如果一个数组有n个元素,则第一个元素的下标是0,最后一个元素的下标是n-1。且一个数组中的所有元素具有相同的[数据类型](http://baike.baidu.com/view/675645.htm)

[答案] 0，数据类型

(10) [解析] 字符串是以\0结束的，sizeof是测数组长度的，即所有的字符长度之和包括\0，但strlen是测字符串长度的，它是只针对有效字符串，也就是遇\0而止，\0并不算在其中。故sizeof(a) = 1,strlen(a) = 0。

[答案] ’\0’，1

3. (1) [解析] 只有当i == j时，i/j 为1；当i < j或j > i时，i/j均为0。

[答案]

10000

01000

00100

00010

00001

(2) [解析] 第一次执行while语句：i = 0 ,a[0] = ‘a’,b[0] = ‘a’,c[0] = ‘A’

第二次执行while语句：i = 1 ,a[1] = ‘b’,b[1] = ‘q’,c[0] = ‘Q’

第三次执行while语句：i = 2 ,a[2] = ‘m’,b[1] = ‘i’,c[0] = ‘M’

第三次执行while语句：i = 2,a[3] = ‘\0’,退出while

[答案] AQM

(3) [解析] 第一次执行while语句：s[0] == ‘A’,i = 1;

第二次执行while语句：s[1] == ‘a’,s[1] = ‘z’+‘a’ -‘a’ = ‘z’ ,i = 2;

第三次执行while语句：s[2] == ‘b’,s[2] = ‘z’+‘a’ -‘b’ = ‘y’ ,i = 3;

第四次执行while语句：s[3] == ‘D’,i = 4;

第五次执行while语句：s[5] == ‘\0’,退出

故s字符数组中含有4个字符，AzyD

[答案] AzyD

(4) [解析] 第一次执行for循环：i = 3,c = s[0][3]+s[1][3] – 2\*'0' = '0'+'8'– 2\*'0'=8

s[0][3] = 8+'0' = 8

第二次执行for循环：i = 2,c = s[0][2]+s[1][2] – 2\*'0' = '8'+'9'– 2\*'0'=17

s[0][2] = 7+'0' = 7

第三次执行for循环：i = 1,c = s[0][1]+s[1][1] – 2\*'0' = '9'+'1'– 2\*'0'=10

s[0][1] = 0+'0' = 0

第四次执行for循环：i = 0,c = s[0][0]+s[1][0] – 2\*'0' = '8'+'9'– 2\*'0'=17

s[0][1] =7+'0' = 7

故s[0] = 7078,s[1]没有改变，是9198。

[答案] 7078

9198

(5) [解析] 该程序是实现字符的右移和插入操作。

第一次执行for循环时，i = 5，k = 5,第0~4个整数后移一位，

第0位a[0] = 5,数组变成57 4 8 9 1

第二次执行for循环时，i = 4，k = 5,第0~4个整数后移一位，

第0位a[0] = 1,数组变成157 4 8 9

第三次执行for循环时，i = 3，k = 5,第0~4个整数后移一位，

第0位a[0] = 9,数组变成9157 4 8

第四次执行for循环时，i = 2，k = 5,第0~4个整数后移一位，

第0位a[0] = 8,数组变成89157 4

第五次执行for循环时，i = 1，k = 5,第0~4个整数后移一位，

第0位a[0] = 4,数组变成489157

第六次执行for循环时，i = 0，k = 5,第0~4个整数后移一位，

第0位a[0] = 7,数组变成748915

[答案]

574891

157489

915748

891574

489157

748915

4. (1) [解析] C++中,数组元素的下标总是从0开始,如果一个数组有n个元素,则第一个元素的下标是0,最后一个元素的下标是n-1。若a[i]表示数组a中的第i个元素的话，i的范围是0~2，i<3,for循环的条件有误。

[答案]

 int  main ( )

      { int a[3]={3\*0} ； //改为｛0，0，0｝

         int i；

         for (i=0； i<4； i++) cin>>a[i]； //改为for(int i = 0; i<3; i ++)

         for (i=0； i<4； i++) cout<<a[i]； //改为for(int i = 0; i<3; i ++)

      }

(2). [解析] 在Ｃ++语言中只能逐个地使用下标变量， 而不能一次引用整个数组。 例如，输出有10 个元素的数组必须使用循环语句逐个输出各下标变量：  
for(i=0; i<10; i++)　　cout<< a[i]; 而不能用一个语句输出整个数组，下面的写法是错误的： cout<< a。此外，该题还有一个错误就是cin的插入符的方向不正确。

[答案]

 int main ( )

      { int a[3]={1，2，3} ， i；

        cin<<a；//改为 for(i = 0; i < 3; i ++) cin >> a[i];

         for (i=0； i<3； i++) cout<<a[i]；

      }

(3). [解析] strlen是测字符串长度的，它是只针对有效字符串，也就是遇\0而止，\0并不算在其中。但字符串的下标总是从0开始。如果一个数组有n个元素,则第一个元素的下标是0,最后一个元素的下标是n-1。要实现字符串逆置，第i个元素是和第n-1-i个元素交换。此外，字符串结束应该加‘\0’结束符

[答案]

main ( ) //改为int main()

      { char s[80]，t[200]； int i，sl ；

         gets(s)； sl = strlen(s)；

         for (i=0； i< s1；i++) t[i]=s[sl-i]； //改为 t[i] = s[s1-i - 1]

         for (i=0； i < s1; i++) t[sl+i]=s[i]；

//循环结束加t[sl + i] = ‘\0’

         puts(t)； // 加“return 0;”

      }

(4). [解析] 该题是选择排序，每次从j+1到n-1中找到最小的元素。整型变量p记录第j小的元素当前的位置，若在j+1~n-1中找到a[i]比a[p]还小的，则将p指向i，并将a[p]和第j小元素a[j]交换。

[答案]

main ( ) //改为int main()

      { int a[100]， i， j， p， t， n=20 ；

         for (j = 0； j<n； j ++) cin>> a[j]；

         for (j = 0；j< n - 1 ；j++)

         { p = j；

            for ( i=j+1； i< n - 1；i++ )

                if (a[p]>a[i]) t=i； //t改成p

            if (p!=j)

            { t = a[j]； a[j] = a[p]； a[p]=t； }

         }

         for ( j = 0； j<n； j ++ ) cout<< a[j]； // 加“return 0;”

      }

(5). [解析] 该题是从原串找出现子串的次数，主要思想是从原串的第i个位置开始，子串从第0个位置开始，原串的第i~i+len位置都和子串0~len位置相等，其中len是子串长度，就说子串在原串中出现了一次；当前位置不相等或者已经完全匹配，则从原串的第i+1个位置起继续查找。

[答案]

 int main ( )

      { char substr[80]，str[80]；

         int i，j，k，num=0；

         gets(substr)； gets(str)；

         for (i = 0， str[i]， i++) // 改成 for (i = 0； str[i]； i++)

             for (j=i，k=0；substr[k]==str[j]；k++，j++)

                if (substr[k+1]!='\0') { //未到子串字符串结果，则继续比较

                   num++； //改成；

                   break； //删除

                }

//新增

else //已到字符串结尾，说明子串在原串中出现了一次。跳出循环。

{

num++;

break;

}

         cout<< num；

      }

5. (1) [解析] 该题求对角线元素的和，对角线元素a[i][j]满足的条件是i==j。

[答案]

【1】 i==j

【2】a[i][j]

(2) [解析] 将十进数转换成二进制数，一般采用除二取余法.如果用一个数组b来存放二进制数，可以依次把所得的余数存入b[0]，b[1]，……，b[n]，最后按b[n]，b[n-1]，……，b[1]，b[0]的顺序输出这些余数，就得到了相应的二进制数.

[答案]

【1】 n % base

【2】 n / base

【3】j = i; j >=1; j --

(3) [解析] 该题的思想就是让最大值最小值指向数组的首元素，然后不断的扫描数组的其他元素，找到比当前最大值大或者比最小值小的就将它赋给最大值或最小值，并记录位置。最后将最大值和最小值的对调。

[答案]

【1】j = i;

【2】k = i;

【3】i = a[j];a[j] = a[k];a[k] = i;

(4) [解析] 简单插入排序算法的基本思想使将数组处理 n-1 次，第 k 次处理是将第 k 个元素插入到目前的位置。第 k 次的元素是这样插入的 : 在第 k 次处理时，前面的元素 a[0]，a[1]，…，a[k-1] 必定已排成了升序，将 a[k] 与 a[k-1]，a[k-2]，…a[0] 逐个比较（由后向前），若有 a[j]< a[k] 维持原位不变。否则 a[j] 插入到a[k] 之后。

[答案]

【1】i – 1;

【2】a[j + 1] = a[j];

【3】a[j + 1] = k;

(5) [解析] 合并法排序（将两个有序数组A、B合并成另一个有序的数组C，升序）

基本思想：

① 先在A、B数组中各取第一个元素进行比较，将小的元素放入C数组；

② 取小的元素所在数组的下一个元素与另一数组中上次比较后较大的元素比较，重复上述比较过程，直到某个数组被先排完；

③ 将另一个数组剩余元素抄入C数组，合并排序完成。

[答案]

【1】a[i] > b[j]

【2】j == 5&&i<3

【3】i == 3&&j<5

(6) [解析] 该算法的思想是先将a的第0列，第1列分别放到b的第1列，第2列中，然后将a的第2列放到b的第0列中，最后输出b的元素。

[答案]

【1】b[i][j +1] = a[i][j];

【2】i = 0; i <2; i++

【3】cout << endl;

(7) [解析] 该题就是字符串查找的简单程序。从原串的第i(0~len,len是原串长)个位置查看该位置的字符串是否等于要查找的字符，相等的话返回i，否则i++,若一直到原串结束符都没找到相等的，就返回-1。

[答案]

【1】strlen(t)

【2】t[k] == c;

(8) [解析] 该题的思路类似于第七题。

[答案]

【1】i = 0

【2】i<=9

(9) [解析] 该题实现矩阵的乘法。若矩阵a是m\*n的二维数组，矩阵b是n\*k的二维数组，则a\*b 是m\*k的二维数组，只需将a[i][j]和b[j][s]相乘后求和得到，其中i的范围是0~m-1,j的范围是0~n-1，s的范围是0~k-1。

[答案]

【1】s=0;

【2】a[i][k] \* b[k][j];

【3】cout << endl;

(10) [解析] 该题是字符串连接，先找到字符串a的结束符\0所在的位置，然后从该位置起，将b的内容依次复制给它，最后在a的末尾加上结束符\0。

[答案]

【1】b[j] != ‘\0’

【2】a[i] = ‘\0’

6. (1) [解析] 用a数组存放投票结果，b数组存放候选人编号，然后用switch语句对投票结果进行统计，最后输出得票最多的，当选为厂长。

[参考程序]

#include<iostream> //预编译命令

using namespace std;

int main()

{

int a[1000];//a数组存放投票结果

int b[6] = {0};//b数组存放候选人编号

for(int i = 0; i < 1000; i ++)

{

switch(a[i]) //对投票结果进行统计

{

case 1 : b[0]++; break;

case 2 : b[1]++; break;

case 3 : b[2]++; break;

case 4 : b[3]++; break;

case 5 : b[4]++; break;

case 6 : b[5]++; break;

}

}

int max = b[0], k = 0;

for(int i = 1; i < 6; i ++) //输出投票最多的候选人

{

if(b[i] > max)

{

max = b[i];

k = i;

}

}

(2) [解析] 数组a中存放输入的10个字符，从数组最后一位到第四位，依次往后移一位，[然后将下标为4的元素赋值为@。这里需要注意的两个地方是](mailto:然后将下标为4的元素赋值为@。这里需要注意的两个地方是)：

① 因为需要插入元素，所以如果是动态定义数组的话，需要多申请一个空间。即当原始数组大小是N时，需要定义N+1的空间。如果是静态定义数组的话，则该题意中原数组有10个字符，则需至少需要定义数组char a[11];

② 移动元素时，应该是从a[9]开始往后移一位，然后a[8]…直到a[4]，这样前面元素移出后留下的空间正好可以留给后面的元素。移动过程不能反过来从a[4]移到a[9]。因为如果采用这样的方式，当a[4]移动到a[5]时，它会覆盖元素a[5]，再想移动元素a[5]时，已经不是原来的a[5]了。所以，总结来说，移动过程如果是后移，则从最后一个元素开始移，如果是前移，则是从最前方的元素开始移。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

int main()

{

int N;

cin >> N;

int i;

char \* a = new char[N+1];//这里要插入元素，所以多申请了一个空间

for(i = 0; i < 10; i ++)

cin >> a[i];

for(i = 9; i >= 4; i --) //元素后移

a[i+1] = a[i];

a[4] = '@';

for(i = 0; i < 11 ; i ++) //输出数组中的现有元素

cout << a[i] << ' ';

system("pause");

return 0;

}

[输出结果]

10 //输入N

a b c d e f g h i j //输入10个整数

a b c d @ e f g h i j //输出插入字符后的数组元素

(3) [解析] 首元素和最后一个元素交换，第二个元素和倒数第二个元素交换，第i个元素和第n-1-i个元素交换，直到第n/2个元素交换完毕，就实现了逆序重排。

[参考程序]

整型数组：

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

void invert(int a[], int n)

{

for(int i = 0; i < n / 2; i ++) //交换过程，第i个元素和第n-1-i个元素交换

{

int temp = a[i]; //交换的过程使用中间变量temp

a[i] = a[n - 1 -i];

a[n - 1 - i] = temp;

}

}

int main()

{

int N,i;

cin >> N;

int \*a = new int[N];

for(i = 0; i < N; i ++)

cin >> a[i];

invert(a,N);

for(i = 0; i < N; i ++)

cout << a[i] << ' ';

system("pause");

return 0;

}

字符数组：//逆序过程采用递归。

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

//相当于交换后n-1长度的字符串，然后输出第一个字符

void inversion( char \*str) {

if(\*str != '\0')

{

inversion(str + 1);

cout << \*str;

}

}

int main()

{

char ch[256] = {0};

cin >> ch;

inversion(ch);

cout << endl;

system("pause");

return 0;

}

[输出结果]

5 //输入N

8 6 5 4 1 //输入5个整数

1 4 5 6 8 //输出逆序后的结果

(4) [解析] 从数组的首元素开始查看该元素是不是要删除的字符，若不是，则往后继续查看。找到要删除的元素后，将其后的元素均往前移一位。该题要注意的是元素前移，具体如何实现前移读者可以参考本章习题第2题的解析。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

int main()

{

int i;

char ch[256] = {0};

ch[0] = 'a';

char temp = 's'; //用temp存放要删除的元素

for(i = 1; i < 26; i ++ )

ch[i] = 'a' + i; //顺序存放a到z

for(i = 0; i < 26; i ++)

{

if(ch[i] == 's') //找到要删除的元素

ch[i]= ch[i + 1];//后面的元素均往前移一位

}

for(i = 0; i < 26; i ++ ) //删除元素后，输出数组的元素看是否删除成功

cout << ch[i];

cout << endl;

system("pause");

return 0;

}

[输出结果]

abcdefghijklmnopqrtuvwxyz //删除后的结果

(5) [解析] 假设数组a有n个元素，则a[1]的值最终要存放在a[n]中， 反之a[n]的值最终要存放在a[1]中，通过交换二者的赋值可实现这一目的.类似地，a[2]应与a[n-1]交换，a[3] 应与a[n-2]交换，如此进行下去直到每对要交换的元素对的值都交换过为止.程序中使用变量i和j存放当前要交换赋值的两个数组元素的下标，开始时，i=1，j=n，每交换赋值一次，则i增加1，同时j减少1，当i≥j时，交换结束.

[参考程序]

#include <iostream> //预编译命令

using namespace std;

void reverse(int \*a, int N) //交换过程

{

for(int i = 0, j = N -1; i < j; i++, j --)

{

int temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

int main()

{

int n;

cin >> n;

int \*a = new int[n];

for(int i = 0; i < n; i ++)

cin >> a[i];

reverse(a,n);

for(int i = 0; i < n; i ++)

cout << a[i] << ' ';

cout << endl;

system("pause");

return 0;

}

[输出结果]

6 //输入n

1 8 9 7 6 10 //输入n个整数

10 6 7 9 8 1 //输出逆序后的数组元素

(6) [解析] 程序中设置一个变量max，先将第一个元素的值存入max，然后从第二个元素开始逐个与max比较，如果max比当前的元素小，则把max的值变成当前元素的值，并记录其下标.这样当所有元素都与max比较完后，max的值即是所有元素中的最大值。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

void getMax(int a[], int n)

{

int k = 0;

int max = a[0]; //第一个元素的值存入max

for(int i = 0; i < n; i ++)

{

//如果max比当前的元素小，则把max的值变成当前元素的值，并记录其下标

if(a[i] > max)

{

k = i;

max = a[i];

}

}

cout << "最大值是" << max << " 位置是第" << k + 1 << "个元素" <<endl;

}

int main()

{

int N,i;

cin >> N;

int \*a = new int[N];

for(i = 0; i < N; i ++)

cin >> a[i];

getMax(a,N);

system("pause");

return 0;

}

[输出结果]

6 //输入n

8 29 78 34 23 65 //输入n个整数

最大值是78 位置是第3个元素 //输出最大值和最大值所在的位置

(7) [解析] 将十进数转换成二进制数，一般采用除二取余法.如果用一个数组b来存放二进制数，可以依次把所得的余数存入b[0]，b[1]，……，b[n]，最后按b[n]，b[n-1]，……，b[1]，b[0]的顺序输出这些余数，就得到了相应的二进制数.

[参考程序]

#include<iostream> //预编译命令

using namespace std;

int main()

{

int \*b = new int[32]; //申请新数组b存放计算后得到的二进制

memset(b,0,sizeof(b)\*32); //数组的初始化

int N;

int i = 0;

cin >> N; //输入需要转化的整数

while(N!= 0)

{

b[i] = N %2; //数组b中存放转换后的结果

i++;

N = N/2;;

}

for(int j = i - 1; j >=0; j --) //逆序输出

cout << b[j];

cout << endl;

system("pause");

return 0;

}

[输出结果]

7 //输入要转换的整数N

111 //输出转换后的二进制数

(8) [解析] 为了设计出由计算机输出1到N的全部排列程序，就必须寻找不同排列之间的规律.通过观察N=5(参见本例的运行结果)的排列情况，可以发现，如果把每个排列看作一个自然数，则所有排列对应的数是按从小到大的顺序排列，从当前的排列产生下一个排列时必然会造成某一位置上的数字变大，这一位置显然应该尽量靠右，并且在它左边位置上的数字保持不变，这就意味着这一位置变成的数字来自于它的右边， 并且变大的幅度要尽可能小，也就是说在它右边如有几个数同时比它大时，应该用其中最小的来代替它.由于这一位置是满足上述条件的最右边的一位，所以在它右边的所有数字按逆序排列，即在这些数字的右边没有一个大于它的数.程序中先从右至左找到第一个位置，该位置上的数比它右边的数小，这个位置就是所要找的满足上述条件的位置，然后再从右到左找到第一个比该位置上的数字大的数字所在的位置，将这两个位置上的数字交换，再将该位置右边的所有元素颠倒过来，即将它们按从小到大的顺序排列，就得到了下一个排列.

[参考程序]

#include<iostream> //预编译命令

using namespace std; //使用名字空间

int main()

{

int n;

cin >> n;

int a[9],s = 1, i,j,k,l,m,temp,t;

for(i = 2; i <= n; i ++) //计算出总的排列数目

{

s = s \* i;

}

cout << s << endl;

for(i = 0; i < n ; i ++) //找到排列的最小的值，数字从小到大

a[i] = i + 1;

for(i = 0; i < n; i ++) //输出该排列

cout << a[i] << ' ';

cout << endl;

//先从右至左找到第一个位置，该位置上的数比它右边的数小

for(i = 1; i < s; i ++)

{

j = n - 1;

while(a[j] < a[j - 1])

j --;

for( k = n -1; k >= j ; k --)

{//从右到左找到第一个比该位置上的数字大的数字所在的位置

if(a[k] > a[j -1])

{//将这两个位置上的数字交换

temp = a[j-1];

a[j -1] = a[k];

a[k] = temp;

break;

}

}

//将该位置右边的所有元素颠倒过来

for(l = j , m = n -1; l < m; l ++, m --)

{

temp = a[l];

a[l] = a[m];

a[m] = temp;

}

for(t = 0; t < n; t ++)

cout << a[t] << ' ';

cout << endl;

}

system("pause");

return 0;

}

[输出结果]

3 //输入N

6 //排列个数

1 2 3 //所有排列方案

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

(9) [解析] 对于一个数，它的幂是无穷多个的，但是末尾三位数只有1000种。这表明当第一次重复出现大于等于1000的末尾三位数时，这就是我们要求的M和N了.

[参考程序]

#include<iostream> //预编译命令

using namespace std;

int main()

{

int k,i;

cin >> k; //如果K等于1则退出

long int a[1000] = {0},t,flag = 0;

//当K大于1000时我们只要对它的末尾三位数进行幂运算

if( k >= 1000)

{

k %= 1000;

a[k] ++;

flag = 1;

}

for(t = k, i = 2; ; i ++)

{

k = t \*k;

if(flag == 1 || k >= 1000)

{

k %= 1000;

a[k] += i;

if(a[k] > i) // //末尾三位数出现了第二次，退出循环

break;

}

}

cout<< a[k] << endl;

system("pause");

return 0;

}

[输出结果]

2 //输入K

120 //输出结果

(10) [解析] 从八位选手的循环比赛表中可以看出，这是一个具有对称性的方阵，可以把方阵一分为四来看，那么左上角的4\*4的方阵就是前四位选手的循环比赛表，而右上角的4\*4的方阵就是后四位选手的循环比赛表，它们在本质上是一样的，都是4个选手的循环比赛表，所不同的只是选手编号不同而已，将左上角中方阵的所有元素加上4就能得到右上角的方阵.下方的两个方阵表示前四位选手和后四位选手进行交叉循环比赛的情况，同样具有对称性，将右上角方阵复制到左下角即得到1，2，3，4四位选手和5，6，7，8四位选手的循环比赛表，根据对称性， 右下角的方阵应与左上角的方阵相同.这样，八名选手的循环比赛表可以由四名选手的循环比赛表根据对称性生成出来.同样地， 四名选手的循环比赛表可以由二名选手的循环比赛表根据对称性生成出来，而两名选手的循环比赛表可以说是已知的，这种程序设计方法叫做分治法，其基本思想是把一个规模为n的问题分成若干个规模较小的问题，使得从这些较小问题的解易于构造出整个问题的解.  
程序中用数组a记录n名选手的循环比赛表， 整个循环比赛表从最初的1\*1的方阵按上述规则生成出2\*2 的方阵， 再生成出4\*4 的方阵，……直到生成出整个循环比赛表为止.变量m表示当前方阵的大小，也是要生成的下一个方阵的大小的一半.

[参考程序]

#include<iostream> //预编译命令

#include <iomanip>

#include<cmath>

using namespace std;

//安排赛事进行填制表格

void arrangement(int n, int N, int k, int a[100][100])

{

for(int i = 1; i <= N; i ++)

a[1][i] = i;// a[i][j]代表第i号选手在第j天遇到的对手

int m = 1;

for(int s =1; s <= k; s++)//把问题分成k部分

{

N /= 2;

for(int t = 1; t <= N; t ++)

{

for(int i = m + 1; i <= 2 \* m; i ++)//i控制行

{

for(int j = m + 1; j <= 2 \* m; j ++) //j控制列

{//右下角的值等于左上角的值

a[i][j + (t - 1) \* m \* 2] = a[i - m][j + (t - 1) \* m \* 2 - m];

a[i][j + (t - 1) \* m \* 2 - m] = a[i - m][j + (t - 1) \* m \* 2];

}

}

}

m \*= 2;

}

}

//输出到屏幕

void print(int n, int a[100][100])

{

cout <<"-------" << "循环赛日程表" << "-------"<< endl;

cout << "日期";

for(int p = 1; p < n ;p ++)

cout << setw(4) << setfill('-') << p;

cout << endl;

for(int i = 1; i <= n; i ++)//i控制行

{

cout << setw(3) << setfill(' ') << a[i][1] << "|";

for(int j = 2; j <= n; j ++)//j控制列

cout << setw(4) << setfill(' ') << a[i][j];

cout << endl;

}

}

int main()

{

int k,n = 1;

cout << "请输入k" << endl;

cin >> k;

int a[100][100];

for(int i = 1; i <= k; i ++)

n \*= 2;

cout << "参赛人数" << n << endl;

int N = n;

arrangement(n,N,k,a);

print(n,a);

system("pause");

return 0;

}

[输出结果]

请输入k： //显示

3 //输入k

参赛人数8

------循环赛日程表-------

日期---1----2----3----4---5---6---7

1| 2 3 4 5 6 7 8

2| 1 4 3 6 5 8 7

3| 4 1 2 7 8 5 6

4| 3 2 1 8 7 6 5

5| 6 7 8 1 2 3 4

6| 5 8 7 2 1 4 3

7| 8 5 6 3 4 1 2

8| 7 6 5 4 3 2 1

(11) [解析] 首先问题中对元，角，分的抽取是相对独立的，因为金额为元的货币化为角后抽取是没有意义的，同样金额为角的货币化为分后抽取也是没有意义的，这样就只要考虑从0到9分(元和角一样)之间的金额抽取问题即可.对某一金额k(0<=k<=9)，将所有的由1，2，5三种面值组成金额k的方案全部列举一遍，再对每种组成方案，找出所有的不可抽取的钱币数，这些钱币数就是从金额k中无法抽取的钱币数.  
程序中用数组work存放每一金额k中能够抽取和无法抽取的钱币数的情况，用work[i，j]=1，表示从i中能够抽取j， 用work[i，j]=0表示从i中无法抽取j.

[参考程序]

#include<iostream> //预编译命令

#include <iomanip> //格式化输出头文件

using namespace std; //使用名字空间

int main()

{

int array[10];

int value,i,j,k,n,x,y,z,total;

int temp[10];

int work[10][10];

for(i = 0; i < 10; i ++)

{

for(int j = 0; j < 10; j ++)

work[i][j] = 1;

}

for(value = 0; value < 10; value ++)

{

for(x = 0; x <= value / 5; x ++) //x列举面值为5的钱币数

{

for(y = 0; y <= (value - 5 \* x) / 2; y ++)//y列举面值为2的钱币数

{

z = value - 5 \* x - 2 \* y;//z表示面值为1的钱币数

for(i = 0; i < 10; i ++)

{

temp[i] = 0;

}

for(i = 0; i <= x; i ++)

{

for(j = 0; j <= y; j ++)

{

for(k = 0; k <= z; k ++)

{

temp[5 \* i + 2 \* j + k] = 1;

}

}

}

for( i = 0; i < 10; i ++)

{

if(temp[i] == 0)

work[value][i] = 0;

}

}

}

}

cout << "Input the value :" << endl;

cin >> value;

i = value / 100;

j = (value / 10) % 10;

k = value %10;

cout << "The value that is able to take from $" << i << '.' << j << k << " are" << endl;

total = 0;

for(n = 1; n <= value; n ++)

{

x = n / 100;

y = (n / 10) % 10;

z = n % 10;

if((work[i][x] ==1) && (work[j][y] == 1) && (work[k][z] == 1))

{

total = total + 1;

cout << setw(6) << n ;

if(total % 10 == 0)

cout << endl;

}

}

system("pause");

return 0;

}

[输出结果]

Input the value: //显示

527 //总钱数

The value that is able to take from $5.27 are // 一定可以抽出的钱币值

2 5 7 20 22 25 27 500 502 505

507 520 522 525 527

(12) [解析] 用二维数组存放该柱状图，因为月份有超过10的两位整数，所以数组的第一列，第二列存放月份，后12列从第12行开始依次存放产值。该题需要注意的地方是

① \*号是字符型的，所以将数组设成字符型的，月份是10月以后时，应该用两个字符来存储，所以这里用数组的第1列，第二列来存储月份(该题中数组设为从第1列开始)。其中matrix [1][1] = 1, matrix [1][2] = 2, matrix [2][1] = 1, matrix [2][2] = 1, matrix [3][1] = 1, matrix [3][2] = 0,则设i是月份，则当 i <= 3时，matrix[i][1] = (13 - i)/10 + '0'，matrix[i][2] = (13 - i)%10 + '0'，当i属于1~9时，matrix[i][1] = (13- i) + '0'。

② 打印产量\*时，应该从第12行开始打印，一直打印到12-a[i]+1行，其中a[i]代表第i个月的产量。还需注意的一点是第i个月的产量对应的是第i+2列，因为数组的前两列存储的是月份。

③ 横坐标的月份只需格式输出即可。

[参考程序]

#include <iostream> //预编译命令

#include<iomanip>

using namespace std;

int main()

{

char\*\* matrix = new char\*[13 \* 16];

int \* a = new int[13];

int i,j;

for(i = 1; i < 13; i ++ ) //输入12个月的产量

{

cin >> a[i];

}

for (i = 1; i < 13; ++i) //初始化二维数组

{

matrix[i] = new char[13];

for (int j = 1; j < 16; ++j)

matrix[ i ][ j ] = ' '; //

}

for(i = 1; i < 13; i ++) //矩阵的第一列，第二列初始化为月份

{

if( i <= 3)//月份超过10，需占两列

{

matrix[i][1] = (13 - i)/10 + '0';

matrix[i][2] = (13 - i)%10 + '0';

}

else //月份是1~9

{

matrix[i][1] = (13- i) + '0';

}

}

for(i = 1; i < 13; i++) // 初始化第i个月的产值

{

int j = i + 2;

for(int k = 12; k >= 13 - a[i]; k --)// 从第12行开始，依次存放产值

{

matrix[k][j] = '\*';

}

}

for(i = 1; i < 13; i ++) //打印柱状图

{

for(j = 1; j <= 2; j ++)

{

cout << matrix[i][j];

}

for(j = 3; j < 16; j ++)

{

cout <<setw(4)<< matrix[i][j] << ' ';

}

cout << endl;

}

//打印横坐标

cout << setw(7) << "一" << setw(5) << "二" << setw(5) << "三"<< setw(5) << "四"<< setw(5) << "五" << setw(5) << "六"<< setw(5) << "七"

<< setw(5) << "八" << setw(5) << "九" << setw(5) << "十"<< setw(5) << "十"<< setw(5) << "十"<< endl;

cout << setw(7) << "月" << setw(5) << "月" << setw(5) << "月"<< setw(5) << "月"<< setw(5) << "月" << setw(5) << "月"<< setw(5) << "月"

<< setw(5) << "月" << setw(5) << "月" << setw(5) << "月"<< setw(5) << "一"<< setw(5) << "二"<< endl;

cout << setw(57) << "月" << setw(5) << "月" << endl;

system("pause");

return 0;

}

[输出结果]

2 3 1 6 5 7 9 10 8 5 3 1//输入12个月的产量

//输出图案

(13) [解析] 用结构体定义蔬菜，包含两个数据成员，名称和价格。初始化蔬菜名称和价格后，按价格对蔬菜从高到低排序，输出排序后的结果。

[参考程序]

#include <iostream> //预编译命令

#include <string>

using namespace std;

struct vegetable //蔬菜的结构体

{

string name;

float price;

};

void input(vegetable \*a,int n) //输入蔬菜的名称和价格

{

for(int i = 0; i < n; i ++)

{

cin >> a[i].name;

cin >> a[i].price;

}

}

void sort(vegetable \*a,int n) //按照价格对蔬菜进行排序

{

for(int i = 0; i < n; i ++)//冒泡排序

{

for(int j = 0; j < n-1- i; j ++)

{

if(a[j].price < a[j+1].price)

{

vegetable temp = a[j];

a[j] = a[j + 1];

a[j+1] = temp;

}

}

}

}

void output(vegetable \*a, int n) //输出排序后的结果

{

for(int i = 0; i < n; i ++)

cout << a[i].name << ' ' << a[i].price << endl;

}

int main()

{

int N;

cin >> N;

vegetable \*matrix= new vegetable[N];

for(int i = 0; i < N; i ++)

{

matrix[i].name = '\0';

matrix[i].price = 0;

}

input(matrix,N);

sort(matrix,N);

output(matrix,N);

system("pause");

return 0;

}

[输出结果]

13 //输出N

jiezi 3.00 //输入N种蔬菜的名称和价格

fanqie 2.20

lajiao 4.80

tudou 2.60

doujiao 2.00

wandou 1.60

donggua 4.20

jiucai 1.60

shizijiao 3.60

yuanbaicai 0.40

youcai 0.60

lajiao 4.8 //输出按价格从高到低排序后的结果

donggua 4.2

shizijiao 3.6

jiezi 3

tudou 2.6

fanqie 2.2

doujiao 2

huanggua 2

wandou 1.6

jiucai 1.6

qincai 1

youcai 0.6

yuanbaicai 0.4

(14) [解析] 对商品定义结构体数组，包括商品的数量和价格。对商店定义结构体，商店里存放的是商品数组和营业总额，首先对商品的价格进行初始化，再对商店的所有商品初始化，然后计算营业总额。

[参考程序]

#include <iostream> //预编译命令

using namespace std;

struct product

{

int num;

float price;

};

struct store

{

product p[3];

float amount;

};

int main()

{

int i,j;

store s[4];

for(i = 0; i < 4; i ++) //商品价格初始化

{

s[i].p[0].price = 12.5;

s[i].p[1].price = 6.8;

s[i].p[2].price = 4.3;

}

//商店的商品及数目进行初始化

s[0].p[0].num = 58;

s[0].p[1].num = 64;

s[0].p[2].num = 83;

s[1].p[0].num = 126;

s[1].p[1].num = 37;

s[1].p[2].num = 63;

s[2].p[0].num = 194;

s[2].p[1].num = 81;

s[2].p[2].num = 112;

s[3].p[0].num = 206;

s[3].p[1].num = 120;

s[3].p[2].num = 101;

//计算营业总额

for(i = 0; i < 4; i++)

{

s[i].amount = 0.0;

for(j = 0; j < 3; j++)

{

s[i].amount += (s[i].p[j].price \* s[i].p[j].num);

}

}

//输出营业总额

for(i = 0; i < 4; i ++)

cout << "第" << i + 1 << "商店的营业总额 " << s[i].amount << endl;

system("pause");

return 0;

}

[输出结果]

第1商店的营业总额 1517.1

第2商店的营业总额 2097.5

第3商店的营业总额 3457.4

第4商店的营业总额 3825.3

(15) [解析] 用结构体定义学生，包括学号，姓名，性别，年龄，成绩。先输入学生信息，按照性别统计男，女人数。用strcmp比较性别，若strcmp(stu.gender,“male”)，返回为0时说明该学生是男生，否则是女生。然后计算所有学生的总年龄，总成绩，求平均后得到平均年龄，平均成绩，然后将低于平均成绩的学生打印出来。

[参考程序]

#include <iostream>

using namespace std;

struct Student //定义学生的结构体

{

char num[10]; //学号

char name[20]; //姓名

char gender[7]; //性别

int age; //年龄

float grade; //成绩

};

int main()

{

Student stu[10];

int sum\_age = 0;

float sum\_grade = 0;

for(int i = 0; i < 10; i ++) //输入学生信息

{

cout << "学号： " ;

cin >> stu[i].num ;

cout << "姓名：";

cin >> stu[i].name;

cout << "性别：";

cin >> stu[i].gender;

cout << "年龄：";

cin >> stu[i].age;

cout << "成绩：";

cin >> stu[i].grade;

}

int count = 0;

for(int i = 0; i < 10; i ++) //按照性别统计学生人数

{

if(!strcmp(stu[i].gender,"male"))

count ++; //男生人数

}

for(int i = 0; i < 10; i ++) //计算所有学生的总年龄，总成绩

{

sum\_age += stu[i].age;

sum\_grade += stu[i].grade;

}

float avg\_age =(float) sum\_age / 10;//计算平均年龄

float avg\_grade = sum\_grade / 10; //计算平均成绩

cout << "男生人数是：" << count << endl;//输出男生人数

cout << "女生人数是：" << 10 - count << endl;//女生人数是总人数-男生人数

cout << "低于平均分的学生如下：" << endl;

for(int i = 0; i < 10; i ++)//打印出低于平均成绩的学生信息

{

if(stu[i].grade < avg\_grade)

cout <<"学号是：" << stu[i].num << "姓名： " << stu[i].name << "性别 " << stu[i].gender <<"年龄： " << stu[i].age << "成绩： " << stu[i].grade <<endl;

}

system("pause");

return 0;

}

(16) [解析] 选择法就是先将10个数中最小的数与a［0］对换；再将a［1］到a［9］中最小的数与a［1］对换……每比较一轮，找出一个未经排序的数中最小的一个。共比较9轮

[参考程序]

#include <iostream>

using namespace std;

void Selectsort(int a[], int n)

{

for(int i = 0; i < n - 1; i ++)

{//将a[i+1]~a[n]中的数和a[i]比较，若有比a[i]小的，和a[i]交换

int k = i;

for(int j = i + 1; j < n; j ++)

{

if(a[k] > a[j])

k = j;

}

int temp =a[i];

a[i] = a[k];

a[k] = temp;

}

}

int main()

{

int i,n;

cout << "请输入一个整数：";

cin >> n;

int \*a = new int[n];

cout << "请输入" << n << "个你要排的数 ：";

for(i = 0; i < n; i ++)

cin >> a[i];

Selectsort(a,n);

cout << n << "个数排序后从小到大的顺序是：" << endl;

for(i = 0; i < n; i ++)

cout << a[i] << ' ';

cout << endl;

system("pause");

return 0;

}

[输出结果]

请输入一个整数：10 //输入N

请输入10个你要排的数： 10 9 8 7 6 5 4 3 2 1 //输入N个整数

10个数排序后从小到大的顺序是： //输出排序后的结果

1 2 3 4 5 6 7 8 9 10

(17) [解析] 该题需要注意的一点是题目中指出是已排好序的数组，但未指明是从小到大排还是从大到小排，所以需要分情况讨论。

① 原数组从小到大排：用a[0] <= a[1]判断。从数组的第0个元素开始找，如果找到元素a[i]大于要插入的整数num，则从a[n-1]~a[i]依次后移一位，将num插入到原a[i]位置。如果数组末尾a[n-1]都未找到比num大的，则将num赋给a[n]即可。

(2) 原数组从大到小排：用a[1] <= a[0]判断。从数组的第0个元素开始找，如果找到元素a[i]小于要插入的整数num，则从a[n-1]~a[i]依次后移一位，将num插入到原a[i]位置。如果数组末尾a[n-1]都未找到比num小的，则将num赋给a[n]即可。

[参考程序]

#include <iostream> //预编译命令

using namespace std;

int main()

{

int n,num;

int k = 0;

cout << "请输入一个整数：";

cin >> n;

int \*a = new int[n+1]; //要插入元素，所以多定义一个空间

cout << "请输入" << n << "原先排好序的数 ：";

int i = 0;

for(i = 0; i < n; i ++)

cin >> a[i];

cout << "请输入要插入的整数：";

cin >> num;

if(a[1] <= a[0]) // 原数组从大到小排

{

for(i = 0; i < n; )

{

if(a[i] > num)//若该元素小于当前数组元素，则数组后移

i ++;

else

break;

}

k = i;

//若未到数组最后一个元素，则需要后移该位置后的所有数组元素

if(k != n)

{

for(int j = n -1; j >= k; j --)

a[j + 1] = a[j];

a[k] = num;//在该位置插入元素

}

else

a[n] = num;//否则末尾元素插入该元素

}

else //原数组从小到大排

{

for(i = 0; i < n; )

{

if(a[i] < num) //若该元素大于当前数组元素，数组后移

i ++;

else

break;

}

k = i;

if(k != n )//若要插入的位置未到数组末尾，则该位置后的所有元素后移

{

for(int j = n -1; j >= k; j --)

a[j + 1] = a[j];

a[k] = num;//在该位置插入该元素

}

else

a[n] = num;//否则在数组末尾插入该元素

}

cout << n + 1<< "个排好序的数是：" << endl;

for(i = 0; i < n+1; i ++)//输出排好序的数组

cout << a[i] << ' ';

system("pause");

return 0;

}

[输出结果]

请输入一个整数：5 //输入N

请输入5个你要排的数： 10 9 8 7 6 //输入N个排好序的整数

请输入要插入的整数：4 //输入要插入的数

6个排好序的数是： 10 9 8 7 6 4 //输出排序后的结果

(18) [解析] 该题只需定义好不同类型字符的计数器，初始化为0。然后从文章的首字符进行判断，遇到某种类型的字符，该类型字符的计数器加1，一直到文章末尾。

[参考程序]

#include <iostream> //预编译命令

using namespace std;

int main()

{

char a[3][80];

int b\_letter = 0,s\_letter = 0,c\_char = 0,digit = 0,space = 0,other = 0;

cout << "请输入三段文字 :" << endl;

for(int i = 0; i < 3; i ++)

{

for(int j = 0; (a[i][j] = getchar()) != '\n'; j ++)

{

if(a[i][j] > 128) //中文字符

{

c\_char ++;

j ++;//一个汉字有两个字节

}

else if(a[i][j] >= 'a' && a[i][j] <= 'z')// 小写字母

s\_letter ++;

else if(a[i][j] >= 'A' && a[i][j] <= 'Z') //大写字母

b\_letter ++;

else if(a[i][j] >= '0' && a[i][j] <= '9') //数字

digit ++;

else if(a[i][j] == ' ')//空格

space ++;

else

other ++; //其他字符

}

}

cout << "统计结果如下：" << endl;

cout << "英文大写字母的个数是：" << b\_letter << endl;

cout << "英文小写字母的个数是：" << s\_letter << endl;

cout << "中文字符的个数是：" << c\_char << endl;

cout <<"数字的个数是：" << digit << endl;

cout << "空格的个数是 :" << space << endl;

cout << "其他字符的个数是：" << other << endl;

system("pause");

return 0;

}

[输出结果]

请输入三段文字：

happy 12 middle school

it is a beautiful school

Mr zhuang is a good teacher

统计结果如下：

英文大写字母的个数是：2

英文小写字母的个数是：57

中文字符的个数是：0

数字个个数是：2

空格的个数是：12

其他字符的个数是：0

(19) [解析] 用二维数组a存放某一学生的某门科目的成绩，然后按学生求该学生所有科目的平均成绩，按科目求该科目的所有学生的平均成绩

[参考程序]

#include <iostream> //预编译命令

#include<iomanip>

using namespace std;

int main()

{

float a[4][6];

memset(a,0,sizeof(a));

for(int i = 0; i < 3; i ++) //输入3个学生4门科目的成绩

{

for(int j = 0; j < 4; j ++)

cin >> a[i][j];

}

cout << setw(8) << "语文" << setw(8) << "数学" << setw(8) << "英语" << setw(8) << "计算机" << setw(8) << "总分" <<setw(8) << "平均分" << endl;

for(int i = 0; i < 3; i ++)

{

for(int j = 0; j < 4; j ++)

{

cout <<setw(8) << a[i][j];

a[i][4] += a[i][j]; //a[i][4]存放学生i的所有科目的成绩总和

}

a[i][5] = a[i][4] / 4; //a[i][5]存放学生i的平均成绩

cout <<setw(8) << setprecision(4) << a[i][4]<<setw(8) << setprecision(4)<< a[i][5]; //指定小数位数进行格式化输出

cout << endl;

}

for(int j = 0; j < 4; j ++)

{

for(int i = 0; i < 3; i ++)

{

a[3][j] += a[i][j];//a[3][j]存放j门课所有学生的成绩总和

}

a[3][j] = a[3][j]/ 3; // a[3][j]存放j门课所有学生的平均成绩

cout <<setw(8) << setprecision(4) << a[3][j];

}

cout << endl;

system("pause");

return 0;

}

(20) [解析] 该题考察字符串string的基本操作。

[参考程序]

① #include<iostream> //预编译命令

#include<cstring> //string的头文件

using namespace std; //使用名字空间

int main()

{

string s1;

getline(cin,s1); //读入字符串

int i;

for(i = 0; s1[i]!= '\0'; i ++); //字符串以'\0'结束，当s1[i] = '\0',i为数组长度

int length = i;

cout << length << endl;

system("pause");

return 0;

}

[输出结果]

Happy day //输入字符串

9 //输出字符串长度

②

#include<iostream> //预编译命令

using namespace std;

int main()

{

string s1,s2;

getline(cin,s1);

s2 = s1; //string中支持赋值操作符'='

cout << s2;

cout << endl;

system("pause");

return 0;

}

[输出结果]

Happy day //输入s1字符串

Happy day //输出s2字符串

③

#include<iostream> //预编译命令

#include<cstring>

using namespace std;

int main()

{

string s1,s2;

getline(cin,s1);

getline(cin,s2);

s1 += s2; //string支持串连接操作'+='，结果存放到s1中

cout << s1;

system("pause");

return 0;

}

[输出结果]

Happy day //输入字符串s1

Happy life //输入字符串s2

Happy day Happy life //输出连接后的字符串s1

④

#include<iostream> //预编译命令

using namespace std;

int main()

{

string s;

char c;

getline(cin,s); //输入字符串

cin >> c; //输入要查找的字符

int i;

for(i = 0; s[i] != '\0'; i ++) //从字符串的第0个位置开始查找

{

if(s[i] == c)//若该位置的字符等于要查找的字符，输出

{

cout << "字符" << c << "在串中的位置是" << i + 1 << endl;

break; //跳出

}

}

if(s[i] == '\0') //若比较到字符的结束符'\0',则说明未找到符合条件的

cout << -1 << endl;

system("pause");

return 0;

}

[输出结果]

Happyday //输入字符串s

d //输入要查找的字符d

字符d在字符串中的位置是6 //输出字符d在字符串中的位置

⑤

#include<iostream> //预编译命令

using namespace std;

int main()

{

string s1,s2;

getline(cin,s1); //输入字符串s1

getline(cin,s2); //输入字符串s2

int i = 0;

int result;

while((s1[i] == s2[i])&& (s1[i] != '\0')) i ++; //两字符串在i位置相等

if(s1[i] == '\0' && s2[i] == '\0') //比较到字符串末尾

result = 0; //两字符串相等

else

result = s1[i] - s2[i]; //字符串支持运算符'-'

cout << result << endl;

system("pause");

return 0;

}

[输出结果]

Happy day //输入字符串s1

Happy life //输入字符串s2

-8 //输出相减结果

(21) [解析] 程序设计时需注意以下几点：

① 用字符串变量str存放句子，用容器strArray存放单词。

② i=0开始，依次判断时str[i]是否为空格，如果是，则i+1，继续判断，否则，令k = i(k用来记录单词的起始位置)，且str[i]如果是大写字母，将其转换为小写字母，i+1，继续往后判断单词的大小写，直到str[i]为空格。此时从k位置开始截取长度为i-k的子串。该子串代表新出现的单词。

③ 找到一个新出现的单词并不立即放到容器中，而是与容器中的单词比较，若和容器中的其中一个单词相同，则不做插入操作，否则，将该单词插入到容器中。

④ 容器的大小即为单词的个数

[参考程序]

#include<iostream> //预编译命令

#include<vector>

#include<string>

using namespace std;

#define N = 2000;

int main()

{

string str;

getline(cin,str);

vector<string> strArray;

int i,j,k;

vector<string> :: iterator it;

string sub;

for(i = 0; str[i] != '\0'; i ++);

int length = i; //计算句子的长度，存放在length中

i = 0;

while(i < length)

{

while(i < length && str[i] == ' ')

i = i + 1; //若当前字符为空格，则后移

if(i < length)

{

k = i; //找到单词的开始位置

while(i < length && str[i] != ' ')

{

if(str[i] >= 'A' && str[i] <= 'Z') //大写字母转换为小写字母

{

str[i] = str[i] + 32;

}

i ++;

}

sub = str.substr(k,i-k);//i为单词的结束位置，k为单词的起始位置

for( it = strArray.begin(); it != strArray.end()&& it < strArray.end(); )

{//单词sub和容器中的单词比较

if(sub != \*it)

{

it ++;//若不相等，比较容器中的下一个元素

}

else if(sub == \*it)//找到相等的就退出

{

break;

}

}

if(it == strArray.end()) //到容器末尾还未找到相等的，则插入

{

strArray.push\_back(sub);

}

}

}

cout << strArray.size() << endl;//输出容器的大小，也就是单词的个数

system("pause");

return 0;

}

[输出结果]

Life in the school is beautiful teachers in THE school is nice //输入句子

9 //统计单词个数

(22) [解析] 该题将整数转换为字符串来做。利用串比较将所有字符串从大到小排序，最后做一个串连接即可。程序设计时需要注意以下几点：

① 将数据类型设置为string, 依次读取n个正整数，将其放入容器strArray中。

② 按照容器中的sort函数对容器中的字符串从小到大进行排序。

③ 对容器中的元素按照逆序连接后输出即可。

[参考程序]

#include <iostream> //预编译命令

#include <cstring> //字符串的头文件

#include <vector> //容器的头文件

using namespace std; //使用名字空间

int main()

{

int n;

cin >> n; //输入整数个数

vector<string> strArray; //容器strArray中存放所有字符串

string str;

for(int i = 0; i < n; i ++)

{

cin >> str; //输入整数，用字符串存储

strArray.push\_back(str); //插入到容器中

}

str = '\0';

//对容器中的元素进行排序，默认是从小到大排序的

(strArray.begin(),strArray.end());

for(int i = n - 1; i >=0; i --)

str += strArray.at(i);//逆序进行串连接

cout << str << endl;

system("pause");

return 0;

}

[输出结果]

3 //输入整数个数n

13 312 343 //输入n个整数

343312213 //输出连接后的结果

(23) [解析] 先来看个统计的方法：

假设一共有n=a+b条直线（即n条直线分成2组，分别为a条和b条）

则总的交点数= a内的交点数＋b内的交点数 ＋a，b之间的交点数

我们来分析加入第N条直线的情况(这里以N=4为例)：

（分类方法：和第N条直线平行的在a组，其余在b组）

① 第四条与其余直线全部平行 => 0+4\*0+0=0；

② 第四条与其中两条平行，交点数为0+(n-1)\*1+0=3;

③ 第四条与其中一条平行，这两条平行直线和另外两点直线的交点数为(n-2)\*2=4,而另外两条直线既可能平行也可能相交，因此可能交点数为：

0+（n-2）\*2+0=4 或者 0+(n-2)\*2+1=5

④ 第四条直线不与任何一条直线平行，交点数为：

0+(n-3)\*3+0=3 或0+ (n-3)\*3+2=5 或0+ (n-3)\*3+3=6

即n=4时，有0个，3个，4个，5个，6个不同交点数。

从上述n=4的分析过程中，我们发现：m条直线的交点方案数

=（m-r）条平行线与r条直线交叉的交点数+ r条直线本身的交点方案

=（m-r）\*r+r条之间本身的交点方案数（0<=r<m）

[参考程序]

#include <iostream> //预编译命令

using namespace std;

#define MAX (20 \* 19 / 2 + 1)

int ans[MAX] = {0};

void intersection(int n, int r)

{

int i;

if (n == 0)

ans[r] = 1;

else

for (i = 1; i <= n; i++)

intersection(n - i, r + (n - i) \* i); } //递归方程

int main()

{

int i;

int N;

cin >> N;

intersection(N, 0);

for (i = 0; i < MAX; i++)

if (ans[i])

cout << i << endl;

system("pause");

return 0;

}

[输出结果]

5 //输入5，表示5条直线

0 //输出可能的交点数

4

6

7

8

9

10

(24) [解析] 递归是从函数(过程)自身出发来达到边界条件，在通过边界条件的递归调

用过程中，系统用堆栈把每次调用的中间结果（局部变量和返回地址）

保存起来，直至求出递归边界值。然后返回调用函数。返回的过程中，中间结果相继出栈恢复递归算法的效率往往很低，费时和费内存空间。

对求解某些问题时,我们希望用递归算法分析问题,用非递归算法具体求解问题。这就需要把递归算法转换为非递归算法。

把递归算法转化为非递归算法有如下三种基本方法：

① 通过分析，跳过分解过程，直接用循环结构的算法实现求值过程。

② 自己用栈模拟系统的运行时栈,通过分析只保存必须保存的信息,从而用非递归算法替代递归算法。

③ 利用栈保存参数,由于栈的后进先出特性吻合递归算法的执行过程,因而可以用非递归算法替代递归算法。

[参考程序]

递归算法：

#include <iostream> //预编译命令

using namespace std;

int AKM(int,int);

int main()

{ int m,n;

cin>>m>>n;

cout<<AKM(m,n)<<endl;

system("pause");

return 0;

}

int AKM(int m,int n)

{

int k;

if (m==0)

return n+1;

else

{

if (n==0)

return(AKM(m-1,1)); //递归方程

else

return(AKM(m-1,AKM(m,n-1))); //递归方程

}

}

非递归算法：这里用循环结构的算法实现求值过程。

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

#define M 3000

int akm[M][M];

int Ackerman(int m, int n)

{

int i, j;

memset(akm, 0, sizeof(akm));

for (j = 0; j < M; j ++)

akm[0][j] = j + 1;

for (i = 1; i < M; i ++)//对第一个参数进行循环

{

akm[i][0] = akm[i - 1][1];

for (j = 1; j < M; j ++)//对第二个参数进行循环

akm[i][j] = akm[i - 1][akm[i][j - 1]];

}

return akm[m][n];

}

int main()

{

int S,T;

cin >> S >> T;

cout << Ackerman(S,T) << endl;

system("pause");

return 0;

}

[输出结果]

3 4 // 输入m,n的值

125 // 输出Ack(m，n)的值

(25) [解析] 递归求解该过程，从起始点开始递归，递归方程有两个，一个路径是往右走，下一个端点值是该端点值+2；另一条路径是往上走或者往右下走，下一个端点值是该端点值+1。递归的停止条件是走过的路径条数等于N。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

#define MAX 9

int path[MAX],N;

int write(int i) //输出路径

{

int j;

for(j = 0; j < i; j ++)

cout << path[j] << "->";

cout << path[i] << endl;

}

void f(int i) //递归求解并打印路径

{

if(path[i - 1] == N) //若已经遍历到路径的终点，则打印

{

write(i - 1);

return;

}

if(path[i - 1] + 1 <= N) //若该路径端点值+1还没遍历到路径的终点

{

path[i] = path[i - 1] + 1;

f(i + 1); //则遍历路径端点值+1

}

if(path[i - 1] + 2 <= N) //若该路径端点值+2还没遍历到路径的终点

{

path[i] = path[i - 1] + 2; //则遍历路径端点值+2

f(i + 1);

}

}

int main()

{

cin >> N;

path[0] = 0;

f(1);

system("pause");

return 0;

}

[输出结果]

3 //输入N

0->1->2->3 //输出所有路径

0->1->3 //输出所有路径

0->2->3 //输出所有路径

(26) [解析] 我们知道，在中缀表达式中，表达式的求值顺序并不是按运算符的排列顺序进行的，所以在表达式求值时，一般将中缀表达式转换为后缀表达式，然后对后缀表达式进行求值运算。在后缀表达式中，剔除了所有括号，运算符按优先级从左到右排列。按照中缀变后缀的思想，将后缀表达式还原为中缀表达式，在必要的地方添加括号，就能剔除多余括号。该题的算法思想就是先将中缀表达式转换为后缀表达式，然后再将后缀表达式还原为中缀表达式即可。

[参考程序]

#include<iostream>

#include<string>

using namespace std;

#define MAXLEN 256 // 表达式的最大长度

char infix[MAXLEN] = {0}; // 中缀表达式字符串

char sufix[MAXLEN] = {0}; // 后缀表达式字符串

char stack[MAXLEN] = {0}; // 用字符数组模拟堆栈

int top = 0; // 堆栈顶计数器

// 该函数返回堆栈中的操作符优先级,'#'为起始/终止符

int getIsp(char ch)

{

switch (ch)

{

case '#': return 0;

case '(': return 1;

case '\*'： case '/': return 5;

case '+':

case '-': return 3;

case ')': return 6;

default: return -1;

}

}

// 该函数返回中缀表达式中的操作符优先级,'#'为起始/终止符

int getIcp(char ch)

{

switch (ch)

{

case '#': return 0;

case '(': return 6;

case '\*':

case '/': return 4;

case '+':

case '-': return 2;

case ')': return 1;

default: return -1;

}

}

// 该函数返回后缀表达式中的操作符优先级,注意,此时只有四则运算操作符,没有括号

int getItp(char ch)

{

switch (ch)

{

case '\*': return 3;

case '/': return 4;

case '+': return 1;

case '-': return 2;

default: return -1;

}

}

// 判断一个符是否为小写字母,根据题意,表达式中只会出现小写字母作为变量

int isAlpha(char ch)

{

return (ch >= 'a' && ch <= 'z');

}

// 将中缀表达式转换为后缀表达式,达到消除括号的目的

void in2su()

{

// inidx是中缀表达式的计数器,suidx是后缀表达式的计数器

int inidx = 0, suidx = 0;

char ch = infix[inidx++];

top = 0;

memset(sufix, 0, sizeof(sufix)); // 清空后缀表达式

infix[strlen(infix)] = '#'; // 预先填入一个结束标记

stack[top++] = '#'; // 在栈中放入一个起始标记

// 当碰到结束符时则跳出循环

while (!(ch == '#' && top > 0 && stack[top - 1] == '#'))

{

// 如果是字母(变量),则直接输出到后缀表达式中,不经过栈

if (isAlpha(ch))

{

sufix[suidx++] = ch;

ch = infix[inidx++];

}

else

{

// 如果当前操作符的优先级比栈顶的操作符优先级更高则入栈,读取下一个字符

if (getIcp(ch) > getIsp(stack[top - 1]))

{

stack[top++] = ch;

ch = infix[inidx++];

continue;

}

// 如果当前操作符的优先级比栈顶的操作符优先级更低则出栈

if (getIcp(ch) < getIsp(stack[top - 1]))

{

sufix[suidx++] = stack[--top];

continue;

}

// 如果出栈碰到左括号,则直接读取下一个字符

if (stack[--top] == '(') ch = infix[inidx++];

}

}

}

// 将后缀表达式转换为中缀表达式,在必要时添加括号

void su2in()

{

// inidx是中缀表达式的计数器,suidx是后缀表达式的计数器

// flag数组记录添加括号的位置,off是它的计数器,gap标记操作符之间是否有间隔

// lastop记录前一个操作符,初始为'#'

int inidx = 0, suidx = 0, idx = 0;

int flag[MAXLEN] = {0}, off = 0, gap = 0;

char ch = sufix[suidx++];

char lastop = '#';

char left, right;

top = 0;

memset(infix, 0, sizeof(infix)); // 清空中缀表达式

sufix[strlen(sufix)] = '#'; // 预先填入一个结束符

stack[top++] = '#'; // 将起始符入栈

// 碰到结束符则退出循环

while (!(ch == '#' && top > 0 && stack[top - 1] == '#'))

{

// 如果是变量,则直接入栈

if (isAlpha(ch))

{

stack[top++] = ch;

gap = 1;

}

else

{

// 如果前一个字符不是操作符,则必须从栈中退出变量,构建子表达式

if (gap)

{

// 如果栈中有两个以上多余的变量,则分别加入到中缀表达式中

if (top >= 2 + off)

{

right = stack[--top];

left = stack[--top];

flag[off++] = inidx;

infix[inidx++] = left; // 添加左变量

infix[inidx++] = ch; // 添加操作符

infix[inidx++] = right; // 添加右变量

}

// 否则只添加一个变量到中缀表达式中

else

{

// 如果当前操作符优先级比上一个操作符大,则必须对前一个子表达式添加括号

if (lastop != '#' && getItp(ch) > getItp(lastop))

{

for (idx = inidx; idx > flag[off-1]; idx--)

infix[idx] = infix[idx - 1];

infix[flag[off-1]] = '(';

infix[inidx + 1] = ')';

inidx += 2;

}

right = stack[--top];

infix[inidx++] = ch; // 添加操作符

infix[inidx++] = right; // 添加右变量

}

}

// 如果前一个字符也是操作符,则将其插入到已有的表达式中

else

{

// 如果当前操作符的优先级比上一次更高,则必须给上一个子表达式添加括号

if (getItp(ch) > getItp(lastop))

{

for (idx = inidx; idx > flag[off-1]; idx--)

infix[idx] = infix[idx - 1];

infix[flag[off-1]] = '(';

infix[inidx + 1] = ')';

inidx += 2;

}

// 如果只剩下最后一个子表达式,则必须从栈中插入左变量

if (top > 1 && off == 1)

{

left = stack[--top];

for (idx = inidx; idx > flag[off-1]; idx--)

infix[idx + 1] = infix[idx - 1];

infix[flag[off-1]] = left;

infix[flag[off-1] + 1] = ch;

inidx += 2;

}

// 否则不用插入变量,只要插入操作符就行了

else

{

for (idx = inidx; idx > flag[off-1]; idx--)

infix[idx] = infix[idx - 1];

infix[flag[off-1]] = ch;

inidx++;

off--;

}

}

gap = 0;

lastop = ch;

}

ch = sufix[suidx++]; // 读取下一个字符

}

}

int main()

{

gets(infix); // 读取原始表达式

in2su(); // 从中缀转换为后缀

puts(sufix); // 输出后缀表达式

su2in(); // 从后缀转换为中缀

puts(infix); // 输出中缀表达式

system("pause");

return 0;

}

# 指针

1. [解析] 在c＋＋中调用函数时有三种参数传递方式：

（1）传值调用；   
（2）传址调用（传指针）；   
（3）引用传递；

按值传递在传递的时候，实参被复制了一份，然后在函数体内使用，函数体内修改参数变量时修改的是实参的一份拷贝，而实参本身是没有改变的，所以如果想在调用的函数中修改实参的值，使用值传递是不能达到目的的，这时只能使用引用或指针传递。例如，要实现两个数值交换。   
void swap(int a int b);   
int main(){   
 int a=1 b=2   
　　swap(a b)   
}   
这样，在main()函数中的a b值实际上并没有交换，如果想要交换只能使用指针传递或引用传递。

[参考程序]

#include <iostream> //预编译命令

#include <iomanip> //格式化输出的头文件

using namespace std; //使用名字空间

void swap(int \*f1,int \*f2); //函数申明

int main()

{

int n1,n2,n3;

int \*p1,\*p2,\*p3,\*p;

cout<<"Input three integers n1，n2，n3:\n";

cin>>n1>>n2>>n3;

p1=&n1;

p2=&n2;

p3=&n3;

if(n1>n2)swap(\*p1,\*p2); //传指针实现交换

if(n1>n3)swap(\*p1,\*p3);

if(n2>n3)swap(\*p2,\*p3),

cout<<"Now the order is:\n"<<n1<<setw(8)<<n2<<setw(8)<<n3<<endl;

system("pause");

return 0;

}

void swap(int \*f1,int \*f2) //交换两整数

{

int \*w;

w=f1;

f1=f2;

f2=w;

}

[输出结果]

Input three integers n1,n2,n3 //显示

34 21 25 //输入三个整数

Now the order is:

21 25 34 //输出排序后的结果

2. [解析] 用串比较函数strcmp即可实现串比较，字符串地址是一个指针，直接做参数就可以实现交换。

[参考程序]

#include <iostream> //预编译命令

using namespace std;

char swap(char \*p1,char \*p2);

int main()

{

char str1[20],str2[20],str3[20];

cout<<"Input three lines:"<<endl;

gets(str1); //输入字符串

gets(str2); //输入字符串

gets(str3); //输入字符串

if(strcmp(str1,str2)>0)swap(str1,str2); //串比较，串交换

if(strcmp(str1,str3)>0)swap(str1,str3);

if(strcmp(str2,str3)>0)swap(str2,str3);

cout<<"Now，the order is :\n";

cout<<str1<<endl<<str2<<endl<<str3<<endl;

system("pause");

return 0;

}

char swap(char \*p1,char \*p2) //交换两个字符串

{

char p[20];

strcpy(p,p1);

strcpy(p1,p2);

strcpy(p2,p);

}

[输出结果]

Input three lines: //输入3个字符串

happy life

better day

best wisher

Now,the order is: //输出排序后的结果

best wisher

better day

happy life

3. [解析] 先输入10个整数，存放到整型数组number中。将第一个数赋给最小值min，将最后一个数赋给最大值max，然后其他元素均和min，max比较，如比min小，则和min交换，若比max大，则和max交换。

[参考程序]

#include <iostream> //预编译命令

using namespace std;

void input (int number[10])

{

int i;

cout<<"Input 10 numbers:\n";

for(i=0;i<10;i++)

cout<<number[i]<<" ";

cout<<endl;

}

void max\_min\_value(int array[10])

{

int \*max,\*min,\*p,\*array\_end;

array\_end=array+10;

max=min=array;

for(p=array+1; p<array\_end;p++)

if(\*p>\*max)max=p; //将最大数地址赋给max

else if(\*p<\*min)min=p;// 将最小数地址赋给min

\*p=array[0]; array[0]=\*min; \*min=\*p;//将最小数与第一个数交换

\*p=array[9]; array[9]=\*max; \*max=\*p;//将最大数与最后一个数交换

return;

}

void output(int array[10])//输出函数

{

int \*p;

cout<<"Now，they are:\n";

for(p=array; p<=array+9; p++)

cout<<\*p<<" ";

cout<<endl;

}

int main()

{

int number[10]={32,24,56,78,1,98,36,44,29,6};

input(number);

max\_min\_value(number); //调用交换函数

output(number);//调用输出函数

system("pause");

return 0;

}

[输出结果]

Input 10 numbers:

32 24 56 78 1 98 36 44 29 6

Now,they are

1 24 56 78 32 6 36 44 29 98

4. [解析] 右移操作从数组的末尾元素开始往后移，要实现右移m个位置，可以先实现一次右移，再实现m-1次右移，故可以用递归实现。

[参考程序]

#include <iostream> //预编译命令

#include <iomanip>

using namespace std;

void move(int array[20],int n,int m);

int main()

{

int number[20],n,m,i;

cout<<"How many numbers?";

cin>>n;

cout<<"Input number:\n";

for(i=0;i<n;i++) //输入数组中的元素

cin>>number[i];

cout<<"How many place you want to mober?";

cin>>m; //输入要右移的操作次数

move(number,n,m);//进行m次右移

cout<<"Now，they are :\n";

for(i=0;i<n;i++)//输出右移后的数组结果

cout<<setw(6)<<number[i];

cout<<endl;

system("pause");

return 0;

}

void move(int array[20],int n,int m)

{

int \*p,array\_end;

array\_end=\*(array+n-1); // array\_end指向数组的末尾

for(p=array+n-1;p>array;p--)//右移一个位置，总共要右移m个位置

\*p=\*(p-1);

\*array=array\_end;

m--;//已经右移了一位，故还需做m-1次右移操作

if(m>0)move(array,n,m); //递归执行该操作

}

[输出结果]

How many numbers?5 //输入n

Input number:

7 3 4 5 6 //输入n个整数

How many place you want to mober?3 //输入m

Now,they are

4 5 6 7 3 //输出移动后的结果

5. [解析] 该题目是约瑟夫问题，可以用环形结构实现。用指针p指向数组的首地址，每次循环计数到数组结尾时，重新赋值为0继续进行操作。

[参考程序]

#include <iostream> //预编译命令

using namespace std;

int main()

{

int i,k,m,n,num[50],\*p;

cout<<"Input number of person:n=";

cin>>n;

p=num;

for(i=0;i<n;i++)

\*(p+i)=i+1; //以1至n为序给每个人编号

i=0; //i为每次循环时的计数变量

k=0; //k为按1、2、3报数时计数变量

m=0; //m为退出人数

while(m<n-1)//当退出人数比n-1少时（即退出人数大于1时）执行循环体

{

if(\*(p+i)!=0)k++;

if((k==3))//对退出的人的编号置为0

{

\*(p+i)=0;

k=0;

m++;

}

i++;

if(i==n)i=0; //报数到尾后，i恢复为0

}

while(\*p==0)p++;

cout<<"The last one in NO."<<\*p<<endl;

system("pause");

return 0;

}

[输出结果]

Input number of person: n = 100 //输入总人数n

The lase one is No.7 //输出最后留下的人的编号

6. [解析] 该题是将一个字符串的字串复制成另一个字符串，用string的substr实现。

[参考程序]

#include <iostream>

#include <cstring>

using namespace std;

int main()

{

int m,i,L;

string str1="this a ";

string str2="reading-room";

cout<<"Which character that being to copy?";

cin>>m;

str1= str2.substr(m-1); //从第m个位置开始的子串赋给str1

cout<<str1<<endl;

cout<<endl;

system("pause");

return 0;

}

[输出结果]

Which character that being to copy?6 //输入m

ng-room //输出str1

7. [解析] 该题只需定义好不同类型字符的计数器，初始化为0。然后从文章的首字符进行判断，遇到某种类型的字符，该类型字符的计数器加1，一直到文章末尾。

[参考程序]

#include <iostream> //预编译命令

using namespace std;

int main()

{

int upper=0,lower=0,digit=0,space=0,other=0,i=0;

char \*p,s[20],x;

cout<<"Input string :";

while ((s[i]=getchar())!='\n')i++; //未到换行符，说明文字未结束

p=&s[0];

while (\*p!='\n')

{

if(('A'<=\*p)&&(\*p<='Z'))++upper; //大写字母

else if(('a'<=\*p)&&(\*p<='z'))++lower; //小写字母

else if(\*p==' ')++space; //空格

else if((\*p<='9')&&(\*p>='0'))++digit; //数字

else ++other;//其他字符

p++;//p指向下一个字符

}

cout<<"upper case :"<<upper<<endl <<"lower case: "<<lower<<endl;

cout<<"space: "<<space <<endl<<"digit: "<<digit<<endl;

cout<<"other: "<<other<<endl;

system("pause");

return 0;

}

[输出结果]

Input string:happy Day in 12 middle school //输入字符串

upper case：1 //输出统计结果

lower case：21

space：5

digit:：2

other：0

8. [解析] 该题可以用字符串函数strtol实现。long int strtol(const char \*nptr,char \*\*endptr,int base);

这个函数会将参数nptr字符串根据参数base来转换成长整型数。

[参考程序]

#include <iostream> //预编译命令

#include <string> //string的头文件

using namespace std;

int main()

{

char str[32], \*ps=str, \*ps1;

long a[16], \*p=a, \*p1, op=1;

gets(str);

while (\*ps)

{

\*p=strtol(ps,&ps1,10); 使用strtol将字符串ps转换为整型

if (ps==ps1)//不是整数范围内，则往后扫描

++ps;

else //若是整数，则将其赋给数组a

{

ps=ps1;/

p++;

}

}

p1 = a;//p1指针指向数组a的首地址

int count = p - p1;//p指针指向数组最后一个元素，相减得到元素的个数

cout <<count << endl;

for (p1=a;p1<p;++p1)//输出数组a的所有元素

cout << \*p1 << endl;

system("pause");

return 0;

}

[输出结果]

Input a string:

a123x456 17960?302tab5876

There are numbers in this line.They are:5

123 456 17960 302 5876

# 字符串

1. [解析] 该题考察字符串的基本操作，如果在字符串中提取单词及对字符进行转换(如将小写转换为大写)是该题要掌握的知识点。

[参考程序]

#include <iostream> //预编译命令

#include <string>

using namespace std;

int main()

{

string s,ID;

getline(cin,s);

int k;

if(s[0] >= 'a' && s[0] <= 'z') //串的第一个字符是小写

s[0] = s[0] - 32; //转换为大写

ID = s.substr(0,1);//提取从第0个元素开始的长度为1的子串

int i = 1;

while(s[i] != ' ' ) //遇到空格，继续往后扫描

{

i ++;

}

i = i + 1;

while(s[i] != '\0' && i < s.length()-1) //循环操作直到串结束

{

while(s[i] != ' ' && s[i] != '\0' )

{

while(s[i] != ' ' && s[i - 1] == ' ')//遇到中间名或者姓

{

k = i;//记录开始位置

if(s[i] >= 'a' && s[i] <= 'z') //开始的第一个字母小写变大写

{

s[i] = s[i] - 32;

}

i ++;

}

i ++;

}

ID += s.substr(k,i - k);//i为结束位置，提取子串和ID进行连接

i ++;

}

cout << "登录ID为" << ID << endl;

system("pause");

return 0;

}

[输出结果]

David Beckham

登录ID为DBeckham

2. [解析] 该题可以用字符串数组和指针存储字符串。首先读入字符串，然后对字符串进行比较，对字符串进行排序，最后输出排序后的结果。

[参考程序]

#include<iostream> //预编译命令

using namespace std;

void input(string \*s,int n)

{

cout << "The array before sorting is :" << endl;

for(int i = 0; i < n; i ++)

getline(cin,s[i]); //读入字符串

}

int compare(string s1,string s2)//进行串比较

{

int i = 0;

int result;

while((s1[i] == s2[i])&& (s1[i] != '\0')) i ++;

if(s1[i] == '\0' && s2[i] == '\0')

result = 0;

else

result = s1[i] - s2[i];

return result;

}

void sort(string \*s,int n) //对字符串进行排序，这里采用冒泡排序

{

for(int i = 0; i < n - 1; i ++)

{

for(int j = n - 2; j >= i; j --)

{

if(compare(s[j],s[j+1]) < 0)

{

s[j].swap(s[j + 1]);

}

}

}

}

void output(string \*s,int n) //输出排序后的字符串

{

cout << "The array after sorting is :"<< endl;

for(int i = 0; i < n; i ++)

cout << s[i] << endl;

}

int main()

{

int N;

cin >> N;

getchar();

string \* str = new string[N];

input(str,N);

sort(str,N);

output(str,N);

system("pause");

return 0;

}

[输出结果]

8 //输入要排序的字符串个数

The array before sorting is: //排序前的字符串

PAB

5C

PABC

CXY

CRSI

7

B899

B9

The array after sorting is: //排序后的字符串

PABC

PAB

CXY

CRSI

B9

B899

7

5C

3. [解析] 可以采用顺序存储结构和链式存储结构。本题采用顺序存储结构求解，首先将数组的首元素赋给变量max，然后依次将其他元素和max比较，如比max大，则进行交换并记录下标。

[参考程序]

#include<iostream> //预编译命令

#include<cmath>

using namespace std;

void getMax(float a[], int n)

{

int k = 0;

float max = a[0]; //数组的首元素赋给max

for(int i = 0; i < n; i ++)

{

if((a[i] - max) > 1e-6) //a[i] > max

{

k = i; //记录下标

max = a[i]; //交换

}

}

cout << "最大值是" << max << " 位置是第" << k + 1 << "个元素" <<endl;

}

int main()

{

int N,i;

cin >> N;

float \*a = new float[N];

for(i = 0; i < N; i ++)

cin >> a[i];

getMax(a,N);

system("pause");

return 0;

}

[输出结果]

5 //输入实数个数N

1.23 3.34 5.87 2.34 0.78 //输入N个实数

最大值是5.87 位置是第3个元素 //输出最大值和它的位置

4. [解析] 该题将整数转换为字符串来做。利用串比较将所有字符串从大到小排序，最后做一个串连接即可。

[参考程序]

#include <iostream> //预编译命令

#include <cstring> //字符串的头文件

#include <vector> //容器的头文件

using namespace std; //使用名字空间

int main()

{

int n;

cin >> n; //输入整数个数

vector<string> strArray; //容器strArray中存放所有字符串

string str;

for(int i = 0; i < n; i ++)

{

cin >> str; //输入整数，用字符串存储

strArray.push\_back(str); //插入到容器中

}

str = '\0';

//对容器中的元素进行排序，默认是从小到大排序的

sort(strArray.begin(),strArray.end());

for(int i = n - 1; i >=0; i --)

str += strArray.at(i);//逆序进行串连接

cout << str << endl;

system("pause");

return 0;

}

[输出结果]

3 //输入n

13 312 343 //输入n个整数

343312213 //输出连接后的结果

5. [解析] 假设字符串数组str有n个元素，则str[0]的值最终要存放在str[n-1]中， 反之str[n-1]的值最终要存放在str[0]中，通过交换二者的赋值可实现这一目的.类似地，str[1]应与str[n-2]交换，str[2] 应与str[n-3]交换，如此进行下去直到每对要交换的元素对的值都交换过为止.程序中使用变量i和j存放当前要交换赋值的两个数组元素的下标，开始时，i=0，j=n-1，每交换赋值一次，则i增加1，同时j减少1，当i≥j时，交换结束.

[参考程序]

#include <iostream> //预编译命令

#include <string>

#include <cmath>

using namespace std;

void reverse(string str)

{

int i,j;

for(i = 0,j = str.length() - 1; i < j; i ++,j --)//元素交换

{

char temp = str[i];

str[i] = str[j];

str[j] = temp;

}

cout << str << endl;

}

int main()

{

string s;

getline(cin,s);

reverse(s);

system("pause");

return 0;

}

[输出结果]

it is a nice day //输入一字符串

yad ecin a si ti //输出逆序后的结果

6. [解析] 该题可以用二重循环实现，i指向原字符串数组，j指向key。当原字符串数组第i个位置起，和key的第j个字符相等，j从0到key的长度，则返回i。

[参考程序]

#include<iostream> //预编译命令

#include<cstring>

using namespace std;

int main()

{

string s;

getline(cin,s);

string key;

cin >> key;

int i,j,k;

j = 0;

for(i = 0; i < s.length();)

{

k = i;

for(; j < key.length()&& key[j] != '\0'; )//j从key的起始位置开始比较

{

if(s[i] == key[j]) //相等则i，j同时后移

{

i ++;

j ++;

}

else

{

j = 0;

i ++; //不相等i

k = i;

}

}

if(key[j] == '\0')

{

break;

}

}

cout << k + 1<< endl;

system("pause");

return 0;

}

[输出结果]

Happy day //输入字符串s

day //输入关键字key

7 //输出key在s中出现的位置

7. [解析] 本题考察的地方有以下几点：

(1) 字符串数组的申明和初始化；

(2) 字符串长度的计算；

(3) 排序算法在字符串中的应用。

[参考程序]

#include<iostream> //预编译命令

#include<string>

using namespace std;

void input(string \*s,int n) //输入

{

for(int i = 0; i < n; i ++)

getline(cin,s[i]);

}

void output(string \*s,int n) //输出

{

for(int i = 0; i < n; i ++)

cout << s[i] << endl;

}

void sort(string \*s,int n) //排序

{

for(int i = 0; i < n - 1; i ++) //冒泡排序

{

for(int j = n - 2; j >= i; j --)

{

if(s[j].length() > s[j + 1].length())

{

s[j].swap(s[j + 1]);

}

}

}

}

int main()

{

int N;

cin >> N;

getchar();

string \* str = new string[N];

input(str,N);

sort(str,N);

output(str,N);

system("pause");

return 0;

}

[输出结果]

3 //输入字符串个数N

Happy //输入排序前的字符串

Day

Happy day

Day //输出排序后的字符串

Happy

Happy day

# 文件

1. [解析] 该题读者需要掌握的地方有以下几点：

(1) 文件的打开，关闭操作

(2) 判断文件是否成功打开

(3) 文件的读写操作

(4) 结构体数组的使用和操作

[参考程序]

#include "stdio.h"

#include "stdlib.h"

struct SC{

char index[5]; /\*序号\*/

char std\_n[10]; /\*学号\*/

int scores[5]; /\*各科成绩\*/

int avg\_sc; /\*平均成绩\*/

int total\_sc; /\*总成绩\*/

};

int main(void)

{

FILE \*fp\_in, \*fp\_out;

struct SC sc;

int i;

fp\_in = fopen("std0001.txt","r"); /\*打开输入文件\*/

if(!fp\_in)

{

printf("Can't Open the file std0001.txt\n");

exit(1);

}

fp\_out = fopen("std0002.txt","wr"); /\*打开输出文件\*/

if(!fp\_out)

{

printf("Can't Open the file std0002.txt\n");

fclose(fp\_in);

exit(1);

}

while(!feof(fp\_in)) /\*文件尚为读完\*/

{

/\*从文件中读取一个学生的成绩记录\*/

fscanf(fp\_in,"%s %s %d %d %d %d %d",sc.index,sc.std\_n,&sc.scores[0],&sc.scores[1],&sc.scores[2],&sc.scores[3],&sc.scores[4]);

sc.total\_sc = 0;

for(i=0;i<5;i++) /\*计算总成绩\*/

{

sc.total\_sc += sc.scores[i]; }

sc.avg\_sc = sc.total\_sc/5; /\*计算平均成绩\*/

/\*将计算好的一个结果写入到输出文件中\*/

fprintf(fp\_out,"%s %s %d %d %d %d %d %d %d\n",sc.index,sc.std\_n,sc.scores[0],sc.scores[1],sc.scores[2],sc.scores[3],

sc.scores[4],sc.avg\_sc,sc.total\_sc);

}

fclose(fp\_in);

fclose(fp\_out);

system("pause");

return 0;

}

2. [解析] 该题读者需要掌握的知识点有如下几个方面

(1) 文件结束符的判断

(2) 文件读写一行

(3) 容器的判断重复和记数功能

[参考程序]

#include <iostream> //预编译命令

#include <fstream>

#include <string>

#include <map>

using namespace std;

int main(){

int row = 0; //记录行数

int characters = 0; //记录总字符串数

map<char, int> m; //声明一个map<char，int>容器

ifstream fin("1111.txt"); //读文件 ，读取失败则退出

if(!fin){

cout << "open file error" << endl;

exit(0);

}

string str;

while(1){

if(fin.eof()) break;

getline(fin,str); //读取文件中的一行

row++; //每读一次行数加 1

for(int i = 0; i <str.size(); i++){

char temp = str[i];

/\*

\* 这里主要说下map容器取下标操作 ，借助于map容器的特点，

\* 在map容器查找元素，如果未找到，则自动增加一个未查找到的元素，

\* 用一个char类型（键值）取下标，返回的是该键所关联的值

\* 这里说得比较含糊， 具体参考 c++ primer 书中的map容器介绍

\*/

++m[temp];

}

}

map<char,int>::iterator iter;

cout << "==============================================" << endl;

cout << "every word appears in the file: " << endl;

for(iter = m.begin(); iter != m.end(); iter++){

//打印每个字母以及个数

cout << iter->first << '\t' << iter->second << endl;

characters += iter->second;

}

cout << "==============================================" << endl;

cout << "total rows: " << '\t' << row << endl;

cout << "total characters:" << '\t' << characters << endl;

cout << "==============================================" << endl;

system("pause");

return 0;

}

[输出结果]

=====================================

every word appears in the file:

b 13

c 11

d 9

e 21

h 16

i 16

l 38

n 16

o 30

q 16

r 13

t 13

v 13

x 13

y 23

z 11

=====================================

total rows: 16

total characters: 272

=====================================

# 输入与输出流

1. [解析] 该题需要读者掌握的知识点有：

(1) 文件的读写

(2) 文件的追加

[参考程序]

#include <iostream>

#include <fstream>

using namespace std;

//fun1函数从键盘输入20个整数，分别存放在两个磁盘文件中

void fun1()

{

int a[10];

ofstream outfile1("f1.dat"),outfile2("f2.dat"); //分别定义两个文件流对象

if(!outfile1) //检查打开f1.dat是否成功

{

cerr<<"open f1.dat error!"<<endl;

exit(1);

}

if(!outfile2) //检查打开f2.dat是否成功

{

cerr<<"open f2.dat error!"<<endl;

exit(1);

}

cout<<"enter 10 integer numbers:"<<endl;

for(int i=0;i<10;i++) //输入10个数存放到f1.dat文件中

{

cin>>a[i];

outfile1<<a[i]<<" ";

}

cout<<"enter 10 integer numbers:"<<endl;

for(int i=0;i<10;i++) //输入10个数存放到f2.dat文件中

{

cin>>a[i];

outfile2<<a[i]<<" ";

}

outfile1.close(); //关闭f1.dat文件

outfile2.close(); //关闭f2.dat文件

}

//从f1,dat读入10个数，然后存放到f2.dat文件原有数据的后面

void fun2()

{

ifstream infile("f1.dat"); //f1.dat作为输入文件

if(!infile)

{

cerr<<"open f1.dat error!"<<endl;

exit(1);

}

ofstream outfile("f2.dat",ios::app);

//f2.dat作为输出文件，文件指针指向文件尾，向它写入的数据放在原来数据的后面

if(!outfile)

{

cerr<<"open f2.dat error!"<<endl;

exit(1);

}

int a;

for(int i=0;i<10;i++)

{

infile>>a; //磁盘文件f2.dat读入一个整数

outfile<<a<<" "; //将该数存放到f2.dat中

}

infile.close();

outfile.close();

}

//从f2.dat中读入20个整数，将它们按从小到大的顺序存放到f2.dat

void fun3()

{

ifstream infile("f2.dat"); //定义输入文件流infile，以输入方式打开f2.dat

if(!infile)

{

cerr<<"open f2.dat error!"<<endl;

exit(1);

}

int a[20];

int i,j,t;

for(i=0;i<20;i++)

infile>>a[i]; //从磁盘文件f2.dat读入20个数放在数组a中

for(i=0;i<19;i++) //用冒泡法对20个数排序

for(j=0;j<19-i;j++)

if(a[j]>a[j+1])

{

t=a[j];a[j]=a[j+1];a[j+1]=t;

}

infile.close(); //关闭输入文件f2.dat

ofstream outfile("f2.dat",ios::out);

// f2.dat作为输出文件，文件中原有内容删除

if(!outfile)

{

cerr<<"open f2.dat error!"<<endl;

exit(1);

}

cout<<"data in f2.dat:"<<endl;

for( i=0;i<20;i++)

{

outfile<<a[i]<<" "; //向f2.dat输出已排序的20个数

cout<<a[i]<<" ";

} //同时输出到显示器

cout<<endl;

outfile.close();

}

int main()

{

fun1(); //写文件

fun2(); //文件的追加

fun3(); //文件内容的排序

system("pause");

return 0;

}

2. [参考程序]

C语言版：

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main()

{

int dat[10000];

long t = clock(); //记录当前时间，以ms为单位

FILE \*fin, \*fout;

int i, j, sum;

fout = fopen("out.dat", "w");

for (i = 0; i < 10000; i++)

{

fin = fopen("in.dat", "r");

for (j = 0; j < 10000; j++)

fscanf(fin, "%d", &dat[j]);

sum = 0;

for (j = 0; j < 10000; j++)

sum += dat[j];

sum /= 10000;

fprintf(fout, "%d\n", sum);

fclose(fin);

}

fclose(fout);

printf("COST TIME: %dms.\n", clock() - t); //输出程序耗时

system("pause");

return 0;

}

[输出结果]

COST TIME:39969ms

C++语言版：

#include<iostream>

#include <fstream>

#include <ctime>

using namespace std;

int main()

{

int dat[10000];

long t = clock(); //记录当前时间，以ms为单位

int i, j, sum;

ofstream out("out.dat");

for (i = 0; i < 10000; i++)

{

ifstream in("in.dat");

for (j = 0; j < 10000; j++)

in >> dat[j];

sum = 0;

for (j = 0; j < 10000; j++)

sum += dat[j];

sum /= 10000;

out << sum;

in.close();

}

out.close();

cout << clock() - t << endl; //输出程序耗时

system("pause");

return 0;

}

[输出结果]

COST TIME:46859ms

# 链表

1. [解析] 定义在单链表结构中结点的数据结构，然后从表头开始查找，找到第i个结点时，返回该结点的指针，若一直到表尾还未找到第i个结点，则返回0。

[参考程序]

#include <iostream>

using namespace std;

struct NODE

{

int data;

struct NODE \*next;

}

void GetNode(NODE \*head,int i)

{

if(i < 1)

return 0;

NODE \*p = head;

int k = 0;

while(p != NULL && k < i)

{

p = p ->next;

k ++;

}

if(p == NULL)

return 0;

else

return p;

}

2. [解析] 第1步：先将 2 个表 ha 和 hb 调整为递减的表示形式。

第2步：合并2个表 ha 和 hb

[参考程序]

Inverse\_sort(LinkList llist) //逆向排序

{

PNode p,q;

p= llist->link;

llist->link=NULL;

while(p!=NULL) // 当前元素有效

{

q=p->link;

p->link= llist->link;

llist->link=p;

p=q;

}

}

Merge(LinkList ha, LinkList hb) //合并后由 ha 表示

{

PNode pa,qa, pb,qb;

Inverse\_sort(ha);

Inverse\_sort(hb); pa=ha->link;

qa=ha;

pb=hb->link;

qb=hb;

while(!pa=NULL && !pb=NULL)

{

if(pa->info< pb->info)

{

qb->link=qa->link;

pb->link=pa;

qa->link=pb;

qa=pb;

pb= qb->link

}

} // while

if(pa==NULL)

qa->link=pb;

} // Merge

3. [解析] 先构造一个只有\*head一个结点的单循环链表，将其看成是带头结点的循环单链表，将剩余的结点采用头插入法插入到\*head结点之后，算法如下：

[参考程序]

void invert(NODE \*head)

{

NODE \*p,\*q,\*r;

p=head->next;

head->next = head; //构造只有\*head一个结点的单循环链表

while(p!=head) //p不指向head时循环

{

q=p->next;

p->next=head->next; //将\*p结点插入到\*head结点之后

head->next = p;

p=q; //p后移一个结点

}

}

4. [参考程序]

(1)

void siftToRight ( ListNode \*& p, ListNode \*& pr, int k ) {

if ( p == NULL && pr != first ) { //已经在链的最右端

cout << "已经在链的最右端，不能再右移。" << endl;

return;

}

int i; ListNode \*q;

if ( p == NULL ) //从链头开始

{ i = 1; pr = NULL; p = first; } //重置p到链头也算一次右移

else i = 0;

while ( p != NULL && i < k ) { //右移k个结点

q = p→link; p→link = pr; //链指针p→link逆转指向pr

pr = p; p = q; i++; //指针pr, p右移

}

cout << "右移了" << i << "个结点。" << endl;

}

(2)

Void siftToLeft ( ListNode \*& p, ListNode \*& pr, int k ) {

if ( p == NULL && pr == first ) { //已经在链的最左端

cout << "已经在链的最左端，不能再左移。" << endl;

return;

}

int i = 0; ListNode \*q;

while ( pr != NULL && i < k ) { //左移k个结点

q = pr→link; pr→link = p; //链指针pr→link逆转指向p

p = pr; pr = q; i++; //指针pr, p左移

}

cout << "左移了" << i << "个结点。" << endl;

if ( i < k ) { pr = p; p = NULL; } //指针p移出表外，重置p, pr

}

5. [解析] 实现以单链表为存储结构的字符串的基本操作。

[参考程序]

#include <iostream.h> //预编译命令

const int maxLen = 300; //字符串最大长度为300（理论上可以无限长）

class string1 {

public:

string1 ( ); //构造空字符串

string1 ( char \* obstr ); //从字符数组建立字符串

~string1 ( ); //析构函数

int Length ( ) const { return curLen; } //求字符串长度

string1& operator = ( string1& ob ); //串赋值

int operator == ( string1& ob ); //判两串相等

char& operator [ ] ( int i ); //取串中字符

string1& operator ( ) ( int pos, int len ); //取子串

string1& operator += ( string1& ob ); //串连接

int Find ( string1& ob ); //求子串在串中位置(模式匹配)

friend ostream& operator << ( ostream& os, string1& ob );

friend istream& operator >> ( istream& is, string1& ob );

private:

ListNode<char>\*chList; //用单链表存储的字符串

int curLen; //当前字符串长度

}

//单链表表示的字符串类string1成员函数的实现，在文件string1.cpp中

#include <iostream.h>

#include "string1.h"

string1 :: string1( ) { //构造函数

chList = new ListNode<char> ( '\0' );

curLen = 0;

}

string1 :: string1( char \*obstr ) { //复制构造函数

curLen = 0;

ListNode<char> \*p = chList = new ListNode<char> ( \*obstr );

while ( \*obstr != '\0' ) {

obstr++;

p = p→link = new ListNode<char> ( \*obstr );

curLen++;

}

}

string1& string1 :: operator = ( string1& ob ) { //串赋值

ListNode<char> \*p = ob.chList;

ListNode<char> \*q = chList = new ListNode<char> ( p→data );

curLen = ob.curLen;

while ( p→data != '\0' ) {

p = p→link;

q = q→link = new ListNode<char> ( p→data );

}

return \*this;

}

int string1 :: operator == ( string1& ob ) { //判两串相等

if ( curLen != ob.curLen ) return 0;

ListNode <char> \*p = chList, \*q = ob.chList;

for ( int i = 0; i < curLen; i++ )

if ( p→data != q→data ) return 0;

else { p = p→link; q = q→link; }

return 1;

}

char& string1 :: operator [ ] ( int i ) { //取串中字符

if ( i >= 0 && i < curLen ) {

ListNode <char> \*p = chList; int k = 0;

while ( p != NULL && k < i ) { p = p→link; k++; }

if ( p != NULL ) return p→data;

}

return '\0';

}

string1& string1 :: operator ( ) ( int pos, int len ) { //取子串

string1 temp;

if ( pos >= 0 && len >= 0 && pos < curLen && pos + len - 1 < curLen ) {

ListNode<char> \*q, \*p = chList;

for ( int k = 0; k < pos; k++; ) p = p→link; //定位于第pos结点

q = temp.chList = new ListNode<char> ( p→data );

for ( int i = 1; i < len; i++ ) { //取长度为len的子串

p = p→link;

q = q→link = new ListNode<char> ( p→data );

}

q→link = new ListNode<char> ( '\0' ); //建立串结束符

temp.curLen = len;

}

else { temp.curLen = 0; temp.chList = new ListNode<char> ( '\0' ); }

return \*temp;

}

string1& string1 :: operator += ( string1& ob ) { //串连接

if ( curLen + ob.curLen > maxLen ) len = maxLen - curLen;

else len = ob.curLen; //传送字符数

ListNode<char> \*q = ob.chList, \*p = chList;

for ( int k = 0; k < curLen - 1; k++; ) p = p→link; //this串的串尾

k = 0;

for ( k = 0; k < len; k++ ) { //连接

p = p→link = new ListNode<char> ( q→data );

q = q→link;

}

p→link = new ListNode<char> ( '\0' );

}

int string1 :: Find ( string1& ob ) { //求子串在串中位置(模式匹配)

int slen = curLen, oblen = ob.curLen, i = slen - oblen;

string1 temp = this;

while ( i > -1 )

if ( temp( i, oblen ) == ob ) break;

else i-- ;

return i;

}

6. [解析] 该题分为两种情况：带头结点的非循环双链表和带头结点的循环双链表。

对于带头结点的非循环双链表，在每次查找到对应的结点\*p时，将其freq域增1，再在\*p结点的前面找到一个结点\*q，它或者是头结点或者满足q->freq>=p->freq,然后删除\*p结点，将其插入到\*q结点之后。

[参考程序]

typedef struct node

{

ElemType data; //数据域

struct node \*prior; //指向直接前驱结点的指针

struct node \*next; //指向直接后继结点的指针

int freq; //该结点的频度，初始化为0

}DNode;

int LocateNode(DNode \*h,ElemType x)

{

DNode \*p = h->next,\*q;

while(p!=NULL && p->data !=x)

p = p ->next; //找data域值为x的结点\*p

if(p == NULL) //未找到这样的结点

return 0;

else //找到这样的结点

{

p->freq++; //频度增1

q = p->prior; //q为p的前驱

if(q!=h) //若p为第一个数据结点,则不移动

{

while(q!=h && q->freq < p->freq) //找到q结点，q->freq >= p->freq

q = q->prior;

p->prior->next = p->next; //先删除p结点

if(p->next !=NULL)

p->next->prior = p->prior;

p->next = q->next; //将p结点插入到q结点之后

if(q->next !=NULL)

q->next->prior=p;

q->next = p;

p->prior = q;

}

return 1;

}

}

对于带头结点的循环双链表，在每次查找到对应的结点\*p时，将其freq域增1，再在\*p结点的前面找到一个结点\*q，满足q->freq>p->freq,然后删除\*p结点，将其插入到\*q结点之后。

int LocateNode(DNode \*h,ElemType x)

{

DNode \*p = h->next,\*q;

while(p!=h&& p->data !=x)

p = p ->next; //找data域值为x的结点\*p

if(p == h) //未找到这样的结点

return 0;

p->freq++; //频度增1

q = p->prior; //q为p的前驱

while(q->freq <= p->freq) //找到q结点，q->freq >= p->freq

q = q->prior;

if(q!=p->proir)

{

p->prior->next = p->next; //将p移动到q之后

p->next->prior = p->prior; //先删除p结点, 将p结点插入到q结点之后

q->next->prior=p;

p->next = q->next;

q->next = p;

p->prior = q;

}

return 1;

}