内容回顾

程序的流程控制

- 程序的基本控制流程
  - 顺序流程、分支流程、循环流程
    - if else, if/else if … /else, switch,
    - for, while, do while …
  - 无条件转向控制语句
    - break, continue, goto, return
  - 综合应用

常用调试操作 重点

内容回顾

- 操作一：在菜单中选择“调试”->“继续”(F5)，程序会运行到下一个断点处暂停或到程序结束。

- 操作二：如要仔细观察代码执行情况，可采用F10或者F11单步调试找到精确错误处。其中F10(step over)是跳过函数调用，F11(step over)是进入函数体调用。一般先用F10，确定函数结果是否正确，如不正确则使用F11进入函数体调试。还可以用Shift+F11跳出函数体。

常用调试操作

内容回顾

- 操作三：运行到当前光标下，在某一行代码，右击鼠标，选择“运行到光标”，程序就会执行到鼠标的地方，快捷键(Ctrl + F10)
- 操作四：locals窗口，可以查看运行到断点处局部作用域的所有变量的值，若变量的值有变化会用红色标出
- 操作五：watch窗口，在watch栏中输入变量的名，可以查看运行到断点处内存中变量的值，通常在local窗口中为列出自己感兴趣的变量时使用。

常用调试操作

内容回顾

- 操作六：Autos窗口，列出在当前程序执行的局部所相关的变量，在当前局部作用域中变量较多时可以使用该窗口。
- 操作六：在Debug状态下，若未出现想要的调试窗口时，可以选择菜单“调试”->“窗口”，可以选出watch, locals等窗口。
- 操作七：可以在程序的关键位置打印出关键的变量，这样有助于判断问题是出现在之前还是之后

常用的调试快捷键总结

内容回顾

熟练地使用常用的快捷键比每次用鼠标点按钮要方便准确得多，大家要牢记下面的快捷键。所有的快捷键在相应的菜单下都有写出，随时可查看。

未进入调试时：

- Ctrl+Shift+B：编译
- F5：开始调试，若遇到断点会停止。
- Ctrl+F5：开始运行，若遇到断点不会停止。
- F9：在光标所在行设置或取消一个简单断点。

内容回顾

常用的调试快捷键总结（续）

进入调试后：

- F5：继续执行程序直到遇到下一个断点。
- Shift+F5：停止调试。
- Ctrl+Shift+F5：重新开始调试。
- F10：Step over，执行下一行语句，若有函数调用不会进入函数体。
- F11：Step into，执行下一行语句，若有函数调用会进入函数体。
- Shfit+F11：Step out，跳出当前函数。

实用编程技巧1

- 写完一个部分（如，一个函数，一个循环等）就进行编译，而不是写完整个程序才进行编译

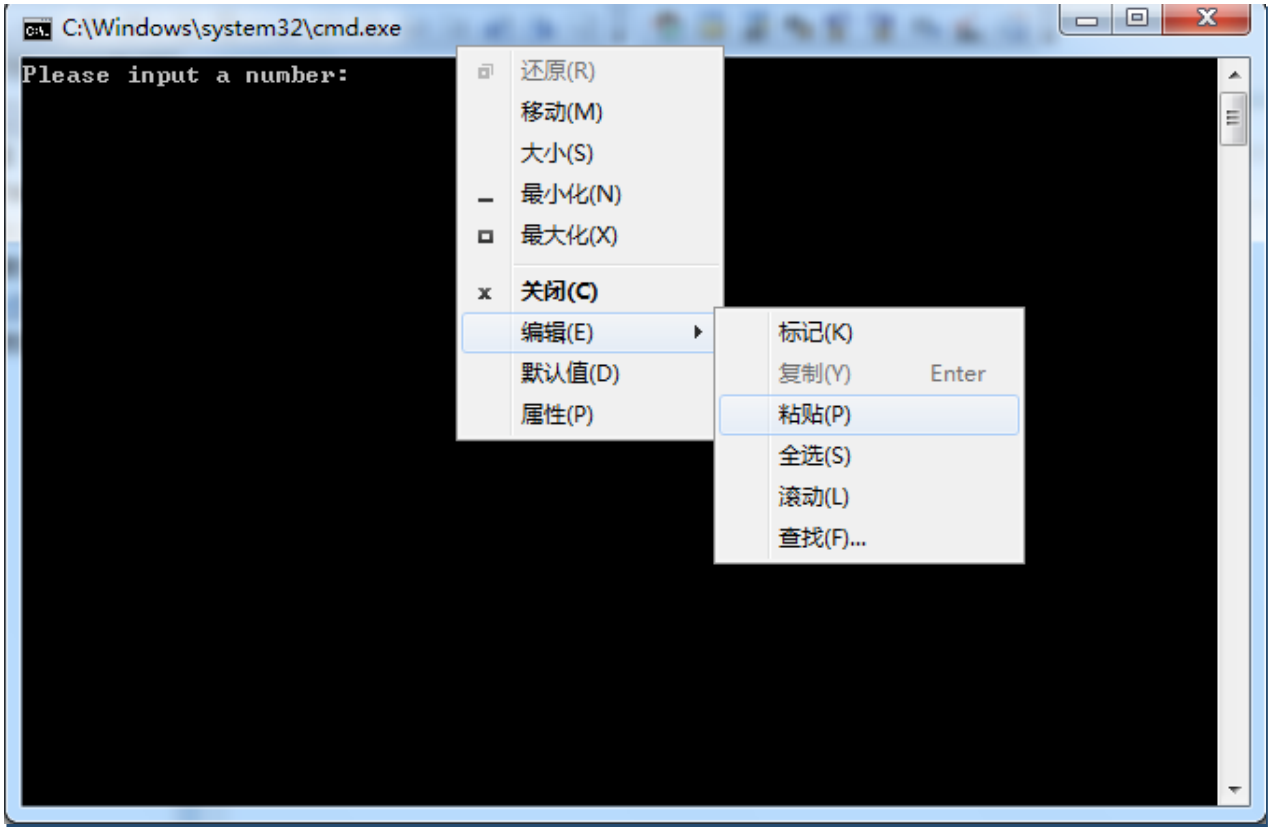
```
#include <iostream>

using namespace std;

void Func();
{
    //...
}

int main()
{
    return 0;
}
```

常见问题

在编写Func函数后就可以找出它，而不必等到main函数写完

实用编程技巧2

- 控制台中也可以粘贴/框选/复制



实用编程技巧3

- 在比较语句中，将常量置于左侧，以避免错误

```
if( a == 1 )
{
    //...
}
```

常用比较语句

```
if( 1 == a )
{
    //...
}
```

更好的写法

```
if( a = 1 )
{
    //...
}
```

常见错误，不易发现

```
if( 1 = a )
{
    //...
}
```

编译不通过，易于发现

实用编程技巧4

- 尽量推迟变量的定义

```
int iSum = 0;
for( int i = 0 ; i < iStuNum ; i++ )
{
    for( int j = 0 ; j < iScoreNum ; j++ )
    {
        iSum += aaiScores[i][j];
    }
    cout<<"Average = "<<iSum/(float)iScoreNum<<endl;
}
```

常见问题：此处忘记清零iSum

实用编程技巧4

- 尽量推迟变量的定义

```
for( int i = 0 ; i < iStuNum ; i++ )
{
    int iSum = 0;
    for( int j = 0 ; j < iScoreNum ; j++ )
    {
        iSum += aaiScores[i][j];
    }
    cout<<"Average = "<<iSum/(float)iScoreNum<<endl;
}
```

推迟iSum的定义

例：编程求出小于n的所有素数（质数）

```
#include <iostream>
using namespace std;
int main()
{    int n;
     cout <<"请输入一个正整数："
     cin >> n; //从键盘输入一个正整数
     for (int i=2; i<n; i++) //循环：分别判断2、3、…、n-1是否为素数
     {    int j=2;
          while (j < i && i%j != 0) //循环：分别判断i是否能被2～i-1整除
               j++;
          if (j == i) //i是素数
               cout << i << " ";
     }
     cout << endl;
     return 0;
}
```

所有代码写在main函数里
仅仅能写
几十、百行量级的代码！

注意：1、上面的for循环中，偶数没有必要再判断它们是否为素数；
2、上面的while循环没有必要到i-1，只需要到：sqrt(double(i))+1

克莉丝汀、元祖
85度C
宜芝多
PARIS BAGUETTE
红跑车
爸爸糖（吐司）

函数 做甜点

和面

输入：面粉、水、鸡蛋

输出：面团

配料

输入：面团、配料

输出：面包胚子

造型

输入：面包胚子

输出：造型的生面包

烘焙

输入：造型的生面包

输出：香喷喷的面包

做甜点

```
int main()
{
    cin>>面粉>>水>>鸡蛋>>牛奶...;

    面团=Func_和面（面粉、水、鸡蛋）；
    面包胚子=Func_配料（面团、配料）；
    造型的生面包= Func_造型（面包胚子）；
    香喷喷的面包= Func_烘焙（造型的生面包）；

    return 香喷喷的面包;
}
```

第 四 章
过程抽象与封装-函数

郭 延 文

2019级计算机科学与技术系

过程抽象

- 过程抽象：一个功能的使用者只需要知道相应功能是什么（what to do），而不必一定知道它是如何做（how to do）的（比如求sin(x)）。
  - 为解决大型、复杂问题提供了一种重要手段，使程序设计者能驾驭问题的复杂度
- 子程序是实现过程抽象的一种方法
  - C++函数

本章内容

- 基于过程抽象的程序设计
- 子程序的概念
- C/C++的函数
- 变量的局部性和变量的生存期
- 标识符的作用域
- 递归函数
- 内联函数
- 函数名重载
- 条件编译——程序调试与多环境程序编制
- 标准库函数

4.1 基于过程抽象的程序设计

郭 延 文

2019级计算机科学与技术系

函数 做甜点

和面

输入：面粉，水

输出：面团

造型

输入：面包胚子

输出：成品面包

配料

输入：面团

输出：面包胚子

烘焙

输入：造型的生面包

输出：香喷喷的面包

做甜点分解为：和面、配料、造型和烘焙等功能过程

基于过程抽象的程序设计

- 人们在设计一个复杂的程序时，经常会用到功能分解和复合两种手段：
  - 功能分解：在进行程序设计时，首先把程序的功能分解成若干子功能，每个子功能又可以分解成若干子功能，等等，从而形成了一种自顶向下（top-down）、逐步精化（step-wise）的设计过程。

辽 宁 舰

基于过程抽象的程序设计

- 人们在设计一个复杂的程序时，经常会用到功能分解和复合两种手段：
  - 功能分解：在进行程序设计时，首先把程序的功能分解成若干子功能，每个子功能又可以分解成若干子功能，等等，从而形成了一种自顶向下（top-down）、逐步精化（step-wise）的设计过程。
  - 功能复合：把已有的（子）功能逐步组合成更大的（子）功能，从而形成一种自底向上（bottom-up）的设计过程。

子程序

- 子程序是取了名的一段程序代码，在程序中通过名字来使用（调用）它们。
- 子程序的作用：
  - 实现过程抽象（功能抽象）
  - 封装和信息隐藏的作用
  - 减少重复代码，节省劳动力

造型

输入：面包胚子

输出：成品面包

烘焙

输入：造型的生面包

输出：香喷喷的面包

配料

输入：面团

输出：面包胚子

函数

做甜点

和面

输入：面粉，水

输出：面团

配料

输入：面团

输出：面包胚子

造型

输入：面包胚子

输出：成品面包

烘焙

输入：造型的生面包

输出：香喷喷的面包

函数 做甜点

和面

输入：面粉，水

输出：面团

配料

输入：面团

输出：面包胚子

造型

输入：面包胚子

输出：成品面包

烘焙

输入：造型的生面包

输出：香喷喷的面包

下一步骤以上一步骤的输出为输入！

子程序之间的数据传递

- 一个子程序所需要的数据往往要从调用者（也是一个子程序）那里获得，计算结果也需要返回给调用者。
- 子程序之间的数据传递方式可以通过：
  - 全局变量：所有子程序都能访问到的变量。
  - 参数：形式参数（形参）和实在参数（实参）。
    - 值传递：把实参的值复制一份给形参。
    - 地址或引用传递：把实参的地址传给形参。
  - 返回值机制：返回计算结果。

Q & A