函数调用的执行过程

- 计算实参的值；
- 把实参传递给被调用函数的形参；
- 执行函数体；
- 函数体中执行return语句返回函数调用点，调用点获得返回值（如果有返回值）并执行调用之后的操作。

函数调用需要一些开销！

- 保存“现场”和切换断点
- 开辟局部内存变量，值传递

- 求两(三)个值x、y的最小或最大值
- 对一个很常用的简单表达式求值，例如：

  sqrt(x)+x*y^2+x+y;
- …

一些常用的简单功能

除了定义为小函数

是否有其他方案？

4.5 宏与内联函数和函数重载

郭 延 文

2019级计算机科学与技术系

解决小函数的低效问题

- 函数调用需要一定的开销，特别是对一些小函数的频繁调用将可能极大地降低程序的执行效率
  - 比如求两个值x,y的最小或最大值

- C++提供了两种办法解决上述问题：
  - 宏定义
  - 内联函数

宏定义

- 在C++中，利用一种编译预处理命令：宏定义，用它可以实现类似函数的功能：
  - #define <宏名>(<参数表>) <文字串>

  例如：
  - #define max(a,b) (((a)>(b))?(a):(b))

- 在编译之前，将对宏的使用进行文字替换！

  例如：编译前将把
  - cout << max(x,y);

  替换成：
  - cout << (((x)>(y))?(x):(y));

一些需要特别注意的“点”

- **需要加上一定的括号**，例如：
  - #define max(a,b) a>b?a:b
  - 10+max(x,y)+z 将被替换成：
    - 10+x>y?x:y+z
- 有时会出现重复计算，例如：
  - #define max(a,b) (((a)>(b))?(a):(b))
  - max(x+1,y*2)将被替换成：
    - (((x+1)>(y*2))?(x+1):(y*2))
- **不进行参数类型检查和转换**
- 不利于一些工具对程序的处理

内联函数

- 内联函数是指在定义函数定义时，在函数返回类型之前加上一个关键词inline，例如：

```
inline int max(int a, int b)
{
    return a>b?a:b;
}
```

- 内联函数的作用是建议编译程序把该函数的函数体展开到调用点，以提高函数调用的效率。
- 内联函数形式上属于函数，它遵循函数的一些规定，如：参数类型检查与转换。
- 使用内联函数时应注意以下几点：
  - 编译程序对内联函数的限制
  - 内联函数名具有文件作用域

能否把所有函数定义为内联函数？

- 内联函数的高效是以代码膨胀（复制）为代价，仅仅省去了函数调用的开销：
  - 对于复杂功能，定义为内联函数效率的收获会很少
  - 每一处内联函数的调用都要复制代码，将使程序的总代码量增大，消耗更多的内存空间。
- 有两种情况，甚用：
  - 如果函数体内的代码比较长，使用内联将导致内存消耗代价较高。
  - 如果函数体内出现循环...

带缺省值的形式参数

在C++中允许在声明函数时，为函数的某些参数指定默认值。如果调用这些函数时没有提供相应的实参，则相应的形参采用指定的默认值。

例如，对于下面的函数声明：

```
void print(int value, int base=10);
```

下面的调用：

```
print(28); //28传给value；10传给base
print(32, 2); //32传给value；2传给base
```

指定默认参数值的注意事项

- 在指定函数参数的默认值时，应注意下面几点：
  - 有默认值的形参应处于形参表的右部。例如：
    - void f(int a, int b=1, int c=0); //OK
    - void f(int a, int b=1, int c); //Error
  - 对参数默认值的指定只在**函数声明**（包括定义性声明）处有意义
  - 在同一个源文件中，对同一个函数的声明只能对它的每一个参数指定一次默认值
  - 在不同的源文件中，对同一个函数的声明可以对它的同一个参数指定不同的默认值

经常可能用到函数功能完全类似，但处理的形参类型不一致的情况，如：

```
void print_int(int i) { ..... }
void print_double(double d) { ..... }
void print_char(char c) { ..... }
void print_A(A a) { ..... } //A为自定义类型

int max(int a, int b);
float max(float a, float b);
double max(double a, double b);
```

函数名重载

- 对于一些功能相同、参数类型或个数不同的函数，有时给它们取相同的名字会带来使用上的方便。例如，把下面的函数：

```
void print_int(int i) { ..... }
void print_double(double d) { ..... }
void print_char(char c) { ..... }
void print_A(A a) { ..... } //A为自定义类型
```

定义为：

```
void print(int i) { ..... }
void print(double d) { ..... }
void print(char c) { ..... }
void print(A a) { ..... }
```

- 上述的函数定义形式称为**函数名重载**。

对重载函数调用的绑定

- 确定一个对重载函数的调用对应着哪一个重载函数的过程称为**绑定**（binding，又称定联、联编、捆绑）。例如：
  - print(1.0)将调用void print(double d) { ..... }
- 对重载函数调用的绑定，**在编译时刻由编译程序根据实参与形参的匹配情况来决定**。从形参个数与实参个数相同的重载函数中按下面的规则选择一个：
  1. 精确匹配
  2. 提升匹配
  3. 标准转换匹配
  4. 自定义转换匹配

精确匹配

- 类型相同
- 对实参进行“微小”的类型转换：
  - 数组变量名->数组首地址
  - 函数名->函数首地址
  - ……

例如，对于下面的重载函数定义：

```
void print(int);
void print(double);
void print(char);
```

下面的函数调用：

```
print(1);    绑定到函数：  void print(int);
print(1.0);  绑定到函数：  void print(double);
print('a');  绑定到函数：  void print(char);
```

提升匹配

- 先对实参进行下面的类型提升，然后进行精确匹配：
  - 按整型提升规则提升实参类型
  - 把float类型实参提升到double
  - 把double类型实参提升到long double

- 例如，对于下述的重载函数：

```
void print(int);
void print(double);
```

根据提升匹配，下面的函数调用：

print('a'); 绑定到函数：void print(int);

print(1.0f); 绑定到函数：void print(double);

标准转换匹配

- 任何算术类型可以互相转换
- 枚举类型可以转换成任何算术类型
- 零可以转换成任何算术类型或指针类型
- 任何类型的指针可以转换成void*
- 派生类指针可以转换成基类指针

- 例如，对于下述的重载函数：

  void print(char);

  void print(char *);

  - 根据标准转换匹配，下面的函数调用：

  print(1); 绑定到函数：void print(char);

绑定失败

- 如果不存在匹配或存在多个匹配，则绑定失败

对于下述的重载函数：

```
void print(char);
void print(double);
```

- 根据标准转换匹配，下面的函数调用将会绑定失败：

```
print(1);
```

- 因为根据标准转换，1（属于int型）既可以转成char，又可以转成double

解决办法是：

- 对实参进行显式类型转换，如，
  - print ((char)1) 或 print ((double)1)
- 或增加额外的重载，如，
  - 增加一个重载函数定义： void print(int);

再论实验上机同学们遇到的一个问题

```
#include"stdio.h"
#include"math.h"
void main()
{
      int i;
      … …
     sqrt(i);
      ….
}
```

vs2008运行下提示错误： error C2668: “sqrt”：对重载函数的调用不明确 试图匹配参数列表“(int)”

错误分析：是因为sqrt函数返回的是浮点型数据，三个重载的函数，一个是double，一个是long double ,一个是float；对重载函数的调用不明确，绑定失败，因而报错。

Q & A

内容回顾

- **递归函数：**

  函数在其函数体中直接或间接地调用了自己。

- **直接递归**

```
void f()
{   .......
    ... f() ...
    .......
}
```

- **间接递归**

```
extern void g();
void f()
{   .......
    ... g() ...
    .......
}
void g()
{   ......
    ... f() ...
    ......
}
```

递归条件和结束条件

- 在定义递归函数时，一定要对两种情况给出描述：
  - 递归条件：指出何时进行递归调用，它描述了问题求解的一般情况，包括：分解和综合过程。
  - 结束条件：指出何时不需递归调用，它描述了问题求解的特殊情况或基本情况。

```
int f(int n)
{
    if (n == 0)
        return 1;
    else
        return n*f(n-1);
}
```

例：汉诺塔问题 -

内容回顾

关键在于分析当前规模和简化后规模的关系

```
#include <iostream>
using namespace std;
void hanoi(char x,char y,char z,int n) //把n个圆盘从x表示的
                                       //柱子移至y所表示的柱子。
{  if (n == 1)
        cout << "1: " << x << "→" << y << endl; //把第1个
                                //盘子从x表示的柱子移至y所表示的柱子。
   else
   {    hanoi(x,z,y,n-1); //把n-1个圆盘从x表示的柱子移至
                          //z所表示的柱子。
        cout << n << ": " << x << "→" << y << endl;
                //把第n个圆盘从x表示的柱子移至y所表示的柱子。
        hanoi(z,y,x,n-1); //把n-1个圆盘从z表示的柱子移至
                          //y所表示的柱子。
   }
}
```

例：母牛生小牛问题 -

内容回顾

关键在于分析当前规模和简化后规模的数值关系

```
myGetCowR(int year, int m)   // 一般的情况
{
        if ( year < m )
                return 1;
        else
                return myGetCowR(year-1) + myGetCowR(year-m+1);
}
```

VS常用（键盘）快捷键

编辑快捷键

Shift + Alt+ Enter: 切换全屏编辑
Ctrl + B,T / Ctrl + K,K: 切换书签开关
Ctrl + B,N / Ctrl + K,N: 移动到下一书签
Ctrl + B,P: 移动到上一书签
Ctrl + B,C: 清除全部标签

Ctrl + I: 渐进式搜索
Ctrl + Shift +I: 反向渐进式搜索
Ctrl + F: 查找
Ctrl + Shift + F: 在文件中查找
F3: 查找下一个
Shift+F3: 查找上一个
Ctrl + H: 替换
Ctrl +Shift + H: 在文件中替换
Alt+F12: 查找符号(列出所有查找结果)
Ctrl + Shift +V: 剪贴板循环

编辑快捷键

Ctrl+左右箭头键: 一次可以移动一个单词
Ctrl+上下箭头键: 滚动代码屏幕，但不移动光标位置。
Ctrl + Shift + L: 删除当前行
Ctrl +M,M: 隐藏或展开当前嵌套的折叠状态
Ctrl + M,L: 将所有过程设置为相同的隐藏或展开状态
Ctrl + M,P: 停止大纲显示
Ctrl + E,S: 查看空白
Ctrl + E,W: 自动换行
Ctrl + G: 转到指定行
Shift + Alt+箭头键: 选择矩形文本
Alt+鼠标左按钮: 选择矩形文本
Ctrl +Shift + U: 全部变为大写
Ctrl + U: 全部变为小写

编辑快捷键

F6: 生成解决方案
Ctrl+F6: 生成当前项目
F7/Ctrl+Shift+B（2019）：编译代码
Shift+F7: 查看窗体设计器
F5: 启动调试
Ctrl+F5: 开始执行(不调试)
Shift+F5: 停止调试
Ctrl+Shift+F5: 重启调试
F9: 切换断点
Ctrl+F9: 启用/停止断点
Ctrl+Shift+F9: 删除全部断点
F10: 逐过程
Ctrl+F10: 运行到光标处
F11: 逐语句

代码快捷键

- Ctrl + J / Ctrl + K,L: 列出成员
  Ctrl + Shift+空格键 / Ctrl +K , P: 参数信息
  Ctrl + K, I: 快速信息
  Ctrl + E,C / Ctrl + K,C: 注释选定内容
  Ctrl + E,U / Ctrl + K,U: 取消选定注释内容
- Ctrl + K,M: 生成方法存根
  Ctrl + K,X: 插入代码段
  Ctrl + K,S: 插入外侧代码
- F12: 转到所调用过程或变量的声明
  Ctrl+F12: 显示代码中选定符号的定义

窗口快捷键

- Ctrl +W , W: 浏览器窗口
  Ctrl + W, S: 解决方案管理器
  Ctrl +W , C: 类视图
  Ctrl +W , E: 错误列表
  Ctrl + W, O: 输出视图
  Ctrl + W, P: 属性窗口
  Ctrl + W, T: 任务列表
  Ctrl + W, X: 工具箱
  Ctrl + W, B: 书签窗口
  Ctrl + W, U: 文档大纲
- Ctrl + D, B: 断点窗口
  Ctrl + D, I: 即时窗口
  Ctrl + Tab: 活动窗体切换
  Ctrl + Shift + N: 新建项目
  Ctrl + Shift + O: 打开项目
  Ctrl + Shift + S: 全部保存
  Shift + Alt + C: 新建类
  Ctrl + Shift + A: 新建项