C++ 程序设计 I

徐东

xu.dong.sh@outlook.com

信息与计算科学

数学系

上海师范大学

2018 年 9 月 8 日

内容

一维数组作为函数的形式参数

- 问题

  设计一个计算标准差的函数

- 一维数组作为形式参数
  - 参数名后必须出现 [] (表示该参数代表一个一维数组)
  - “数组大小”应该作为另一个 (形式) 参数 (传入函数)
- 调用时
  - 实际数组 (实参数组) 的数据类型必须和形参 (数组) 的数据类型完全一致
  - 只需提供实际数组的数组名即可 (不需要跟 [])

计算标准差的函数

```
1  double calStd(double a[], int a_size){
2      double sum = 0.0;
3      for(int i = 0; i < a_size; ++i){
4          sum += a[i];
5      }
6      double m = sum / a_size;
7
8      sum = 0.0;
9      for(int i = 0; i < a_size; ++i){
10         sum += (a[i] - m) * (a[i] - m);
11     }
12     double variance = sum / (a_size - 1);
```

重构代码：消除冗余变量

```
1  double calStd(double a[], int a_size){
2      double sum = 0.0;
3      for(int i = 0; i < a_size; ++i){
4          sum += a[i];
5      }
6      double m = sum / a_size;
7
8      sum = 0.0;
9      for(int i = 0; i < a_size; ++i){
10         sum += (a[i] - m) * (a[i] - m);
11     }
12
```

重构代码：一个函数只实现一个功能

```
1  double calmean(double x[], int x_size);
2
3  double calStd(double a[], int a_size){
4     double m = calmean(a,a_size);
5
6     double sum = 0.0;
7     for(int i = 0; i < a_size; ++i){
8        sum += (a[i] – m) * (a[i] – m);
9     }
10    return sqrt(sum / (a_size – 1));
11 }
12 double calmean(double x[], int x_size){
13    double sum = 0.0;
14    for(int i = 0; i < x_size; ++i){ sum += x[i]; }
```

调用函数计算平均值

```
1  double calmean(double x[], int x_size);
2
3  int main(){
4    double data[10] = {1};
5    double m = calmean(data, 10);
6    cout << ”the␣mean␣of␣data␣=␣” << m << endl;
7    return 0;
8  }
9  double calmean(double x[], int x_size){
10   double sum = 0.0;
11   for(int i = 0; i < x_size; ++i){
12       sum += x[i];
```

调用函数计算平均值

```
1  int main(){
2      double data[10] = {1};
3      double m = calmean(data, 0);
4      cout << "the␣mean␣of␣data␣=␣" << m << endl;
5      return 0;
6  }
```

调用函数计算平均值

```
1  int main(){
2      double data[10] = {1};
3      double m = calmean(data, 0);
4      cout << "the mean of data = " << m << endl;
5      return 0;
6  }
```

- 如何确保传入 calmean() 的第二个参数值是合理值?

调用函数计算平均值

```
1  int main(){
2      double data[10] = {1};
3      double m = calmean(data, 0);
4      cout << "the␣mean␣of␣data␣=␣" << m << endl;
5      return 0;
6  }
```

- 如何确保传入 calmean() 的第二个参数值是合理值?
  - if 语句 (防卫语句)
  - 异常处理 (try-catch 语句块)

计算平均值的函数

```
1  double calmean(double x[], int x_size){
2
3      if(x_size == 0) return 0; //防卫语句
4
5      double sum = 0.0;
6      for(int i = 0; i < x_size; ++i){
7          sum += x[i];
8      }
9
10     return sum / x_size ;
11 }
```

计算平均值的函数

```
1  double calmean(double x[], int x_size){
2      //异常处理
3      try{
4            double sum = 0.0;
5            for(int i = 0; i < x_size; ++i){
6                sum += x[i];
7            }
8            return sum / x_size ;
9      }
10     catch(...){ //... 代表一切可能出现的错误（异常）
11           return 0;
```

一维数组作为函数的形式参数

- 问题

  设计一个计算二维数组均值的函数

- 二维数组作为形式参数
  - 参数名后必须出现 [][n] (表示该参数代表一个二维数组)
  - n 代表二维数组的“列的大小”(不能省略)
  - 数组“行的大小 (m)”应该作为另一个形式参数
- 调用时
  - 实际数组的数据类型必须和形参的数据类型完全相同
  - 实际数组“列的大小”必须和形参“列的大小”完全相同
  - 只需提供实际数组的数组名

计算二维数组平均值的函数

```
1  double calmean(double x[][30], int x_row_size){
2      double sum = 0.0;
3      for(int i = 0; i < x_row_size; ++i)
4      {
5          for(int j = 0; j < 30; ++j){
6              sum += x[i][j];
7          }
8      }
9      return sum/(x_row_size * 30);
10 }
```

调用函数计算 3 个班一门课程的平均成绩

```
1  double calmean(double x[][30], int x_row_size);
2
3  int main(){
4     double score[3][30] = {89};
5     double m = calmean(score, 3);
6     cout << ”the␣mean␣of␣scores␣=␣” << m << endl;
7     return 0;
8  }
9  double calmean(double x[][30], int x_row_size){
10    double sum = 0.0;
11    for(int i = 0; i < x_row_size; ++i){
12       for(int j = 0; j < 30; ++j){
13           sum += x[i][j];
14       }
```

计算一维数组平均值的函数

```
1    double calmean(double x[], int x_size){
2        double sum = 0.0;
3        for(int i = 0; i < x_size; ++i){
4            sum += x[i];
5        }
6        return sum / x_size ;
7    }
```

计算一维数组平均值的函数

```
1  double calmean(double x[], int x_size){
2     double sum = 0.0;
3     for(int i = 0; i < x_size; ++i){
4        sum += x[i];
5     }
6     return sum / x_size ;
7  }
```

- 只能计算 double[] 的平均值

计算一维数组平均值的函数

```
1   double calmean(double x[], int x_size){
2       double sum = 0.0;
3       for(int i = 0; i < x_size; ++i){
4           sum += x[i];
5       }
6       return sum / x_size ;
7   }
```

- 只能计算 double[] 的平均值
- 如何计算 int[] 的平均值?

计算一维整型数组平均值的函数

```
1    double calmeanInt(int x[], int x_size){
2        double sum = 0.0;
3        for(int i = 0; i < x_size; ++i){
4            sum += x[i];
5        }
6        return sum / x_size ;
7    }
```

- 数组作为参数　实参类型与形参类型必须完全相同

计算一维数组平均值的函数

```
1  double calmean(double x[], int x_size){
2     double sum = 0.0;
3     for(int i = 0; i < x_size; ++i){
4        sum += x[i];
5     }
6     return sum / x_size ;
7  }
```

```
1  double calmeanInt(int x[], int x_size){
2     double sum = 0.0;
3     for(int i = 0; i < x_size; ++i){
4        sum += x[i];
5     }
```

- 函数名不同

计算一维数组平均值的函数

```
1  double calmean(double x[], int x_size){
2     double sum = 0.0;
3     for(int i = 0; i < x_size; ++i){
4        sum += x[i];
5     }
6     return sum / x_size ;
7  }
```

```
1  double calmeanInt(int x[], int x_size){
2     double sum = 0.0;
3     for(int i = 0; i < x_size; ++i){
4        sum += x[i];
5     }
```

- 函数名不同
- 函数名可以相同
  - 函数重载

函数重载

- 函数名相同
- 参数列表不同（形参个数、形参类型、形参顺序）
- 函数返回值类型不是区别函数重载的标志
- 编译器根据最匹配的参数列表进行调用。但，不能产生歧义（编译错误）。

函数重载

```
1  void f(int x, int y){
2     cout << ”a” ;
3  }
4  void f(int x, double y){
5     cout << ”b” ;
6  }
7  void f(double x, int y){
8     cout << ”c” ;
9  }
10 int main(){
11    f(1, 2);
12    return 0;
13 }
```

```
1
2
3
4  void f(int x, double y){
5     cout << ”b” ;
6  }
7  void f(double x, int y){
8     cout << ”c” ;
9  }
10 int main(){
11    f(1, 2);
12    return 0;
13 }
```

计算一维数组平均值的函数：重载函数

```
1  double calmean(double x[], int x_size){
2     double sum = 0.0;
3     for(int i = 0; i < x_size; ++i){
4        sum += x[i];
5     }
6     return sum / x_size ;
7  }
```

```
1  double calmean(int x[], int x_size){
2     double sum = 0.0;
3     for(int i = 0; i < x_size; ++i){
4        sum += x[i];
5     }
```

- 函数名相同
- 形参列表不同

重构

```
1  double calmean(double x[], int x_size){
2  double sum = 0.0;
3  for(int i = 0; i < x_size; ++i){
4  sum += x[i];
5  }
6  return sum / x_size ;
7  }
```

```
1  double calmean(int x[], int x_size){
2  double sum = 0.0;
3  for(int i = 0; i < x_size; ++i){
4  sum += x[i];
5  }
```

- 第一个形参的数据类型不同
- 代码完全一样

重构

```
1  double calmean(double x[], int x_size){
2  double sum = 0.0;
3  for(int i = 0; i < x_size; ++i){
4  sum += x[i];
5  }
6  return sum / x_size ;
7  }
```

```
1  double calmean(int x[], int x_size){
2  double sum = 0.0;
3  for(int i = 0; i < x_size; ++i){
4  sum += x[i];
5  }
```

- 第一个形参的数据类型不同
- 代码完全一样

- 函数模板

函数模板

- 通用函数
- 代码的区别仅限于数据类型
- 在 STL(Standard Template Library，标准模板库) 中广泛使用
- 在调用函数时，系统会根据实参的类型来取代模板中的虚拟类型，从而实现了不同类型的参数传递。
- 虚拟类型 (类型的占位符)

```
template<typename T>
template<typename T1, typename T2>
```

函数模板

```
1  template <typename T>
2  double calmean(T x[], int x_size){
3     double sum = 0.0;
4     for(int i = 0; i < x_size; ++i){
5        sum += x[i];
6     }
7     return sum / x_size ;
8  }
```

- T 代表一个（待定的）数据类型

函数模板的使用 I

```
1  #include<iostream>
2  using namespace std;
3
4  template <typename T>
5  double calmean(T x[], int x_size);
6
7  int main(){
8     double a[10] = {1.1, 2.1, 3.1};
9     cout << calmean(a, 10) << endl;
10
11    int b[3] = {1, 2, 3};
12    cout << calmean(b,3) << endl;
```

函数模板的使用 II

```
13
14      return 0;
15  }
16  template <typename T>
17  double calmean(T x[], int x_size){
18      double sum = 0.0;
19      for(int i = 0; i < x_size; ++i){
20          sum += x[i];
21      }
22      return sum / x_size ;
23  }
```

问题

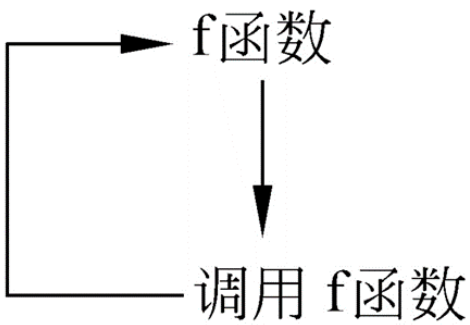
- 设计一个弧度转换函数 toRad()
  - 若未明确指定，则 π 取 3.14。
- π 带默认值的参数
- 实现方式
  - 在函数声明语句中给出参数的默认值
  - 带默认值的参数只能连续地出现在形参列表的尾部
- 重载函数不能具有默认值参数（导致歧义）

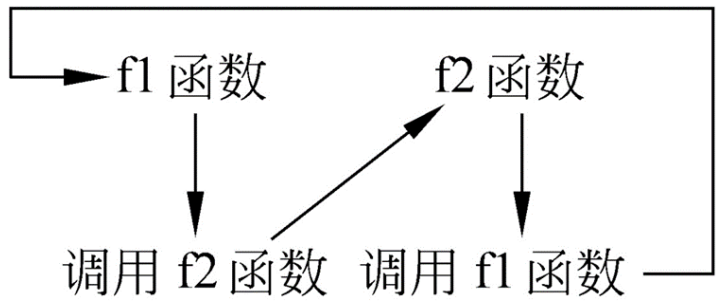
带默认参数的弧度转换函数

```
1  #include<iostream>
2  using namespace std;
3
4  double toRad(double x, double pi = 3.14);
5
6  int main(){
7     cout << toRad(30) << endl; //采用默认值
8     cout << toRad(30, 3.14159265) << endl; //采用实际值
9
10    return 0;
11 }
12
13 double toRad(double x, double pi){
14    return pi * x / 180;
```

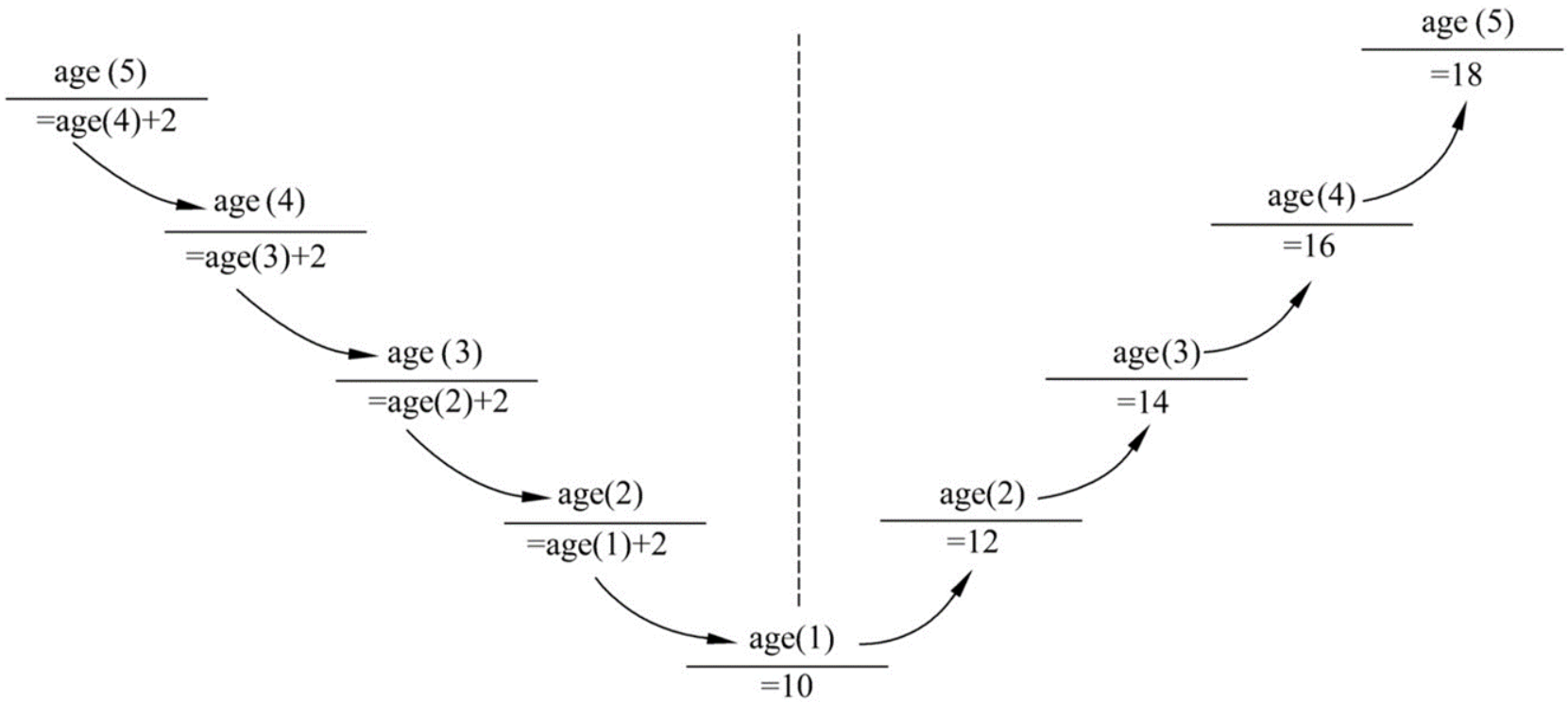
函数递归

- 递归函数
  - 在函数体内部直接或间接地自我调用（函数的嵌套调用是函数本身）





递归策略

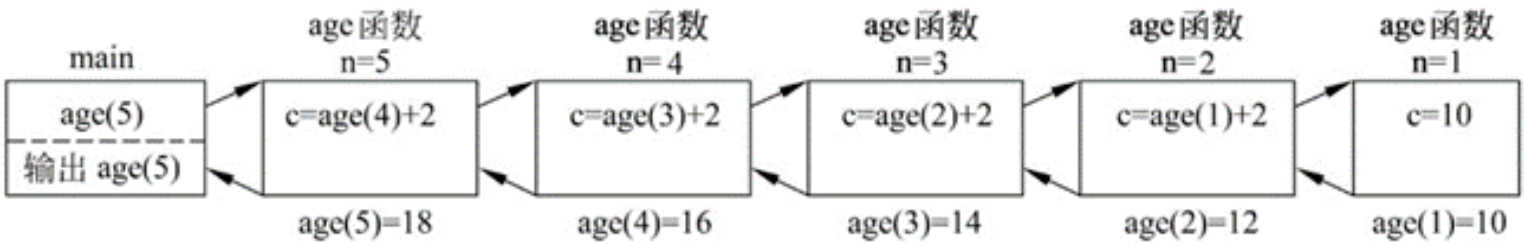

age (5)
=age(4)+2
age (4)
=age(3)+2
age (3)
=age(2)+2
age(2)
=age(1)+2
age(1)
=10
age(2)
=12
age(3)
=14
age(4)
=16
age (5)
=18

递归策略的两个阶段

- 递推（第一阶段）
  - 在该阶段中，将原问题不断化为新的、类似的、较小的问题，逐步从“未知的”向“已知的”方向推进。最终达到已知条件，即递归结束条件，该阶段工作结束。
- 回溯（第二阶段）
  - 在该阶段中，从已知条件（递归结束条件）出发，按“递推”的逆过程，逐步求值回归。最后，达到递推的开始之处，完成递归调用。

计算年龄的递归函数

```
1  int age(int n){
2     int result = 0;
3     if(n == 1) result = 10;
4     else result = age(n-1) + 2;
5
6     return result;
7  }
```



递归结束条件

- 递归类似于循环结构
- 递归结束条件
  - 只有在某一条件成立时，才继续执行递归调用。
  - 否则，就不再继续递归调用，从而避免“无穷递归”（类似无限循环）的发生。
- 使用 `if` 语句实现“递归结束条件”

递归函数

- 递归的优点
  - 简化程序设计（提高代码的可读性）
- 递归与循环等价
  - 递归　　可读性好、速度慢
  - 循环　　可读性差、速度快
- 计算阶乘
- 计算斐波那契数列的第 n 项
- 汉诺塔

补遗

- return 语句最多只能返回一个值
- 函数返回类型 void
  - 不需要得到一个计算结果
  - return 语句不返回值，或直接省略 return 语句。
  - 函数调用只能作为单语句出现，不能出现在表达式中。
- 函数必须先定义（或声明）再使用
- 使用函数声明语句（函数原型）消除对函数定义前后顺序的考虑

变量的作用域和生命周期

- 变量的作用域（有效范围）
  - 全局变量
  - 局部变量
  - 静态局部变量　　关键字 *static*
- 变量的生命周期
  - 变量在内存中的存在时间
  - 与变量的作用域相关
- 在局部变量的作用域中，同名的全局变量会被屏蔽。
  - 就近原则

多文件系统

- 头文件（*.h）
  - 函数声明语句
  - 常量声明
  - 预处理指令（如，#*include* 等）
- 源文件（*.cpp）
  - 函数的具体实现代码（和对应的 #*include* 语句）
  - 源文件名与（对应的）头文件名相同
- 实现代码的分离和组织

多文件系统

- #*include* 语句

$$\#include \ < \text{头文件名} >$$

$$\#include \ \text{"}path\backslash \text{头文件名"}$$

- 双引号内可以放置路径 (针对自定义头文件)

Q & A