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 6 日

内容

问题

- 计算三角形三条边的长度
  - 三个顶点坐标 (1.5, -3.4), (4.6, 5), (9.5 -3.4)

问题

- 计算三角形三条边的长度
  - 三个顶点坐标 (1.5, -3.4), (4.6, 5), (9.5 -3.4)
- 过多重复代码带来的问题

问题

- 计算三角形三条边的长度
  - 三个顶点坐标 (1.5, -3.4), (4.6, 5), (9.5 -3.4)
- 过多重复代码带来的问题
  - 容易隐藏错误

问题

- 计算三角形三条边的长度
  - 三个顶点坐标 (1.5, -3.4), (4.6, 5), (9.5 -3.4)
- 过多重复代码带来的问题
  - 容易隐藏错误
  - 不容易查错、修改、维护

问题

- 计算三角形三条边的长度
  - 三个顶点坐标 (1.5, -3.4), (4.6, 5), (9.5 -3.4)
- 过多重复代码带来的问题
  - 容易隐藏错误
  - 不容易查错、修改、维护
- 消除代码冗余的方法
  - 函数

函数

- 函数的作用
  - 定义可重用的代码
  - 组织和简化编码
- 函数的组成
  - 函数名称
  - 参数（列表）
  - 返回值类型
  - 函数体

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

- 函数头部

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

- 函数头部
  - 返回值类型
  - 函数名称
  - 参数

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

- 函数头部　　规定函数使用（调用）的方式
  - 返回值类型
  - 函数名称
  - 参数

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

- 返回值类型

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, $\cdots$)
{
    处理参数的若干代码;
    return 返回值;
}

- 返回值类型
  - 合法的 $C++$ 数据类型
  - 代表函数调用后得到的结果（定性）
  - 便于编译器检查后续操作是否合法

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

- 函数名

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, $\cdots$)
{
    处理参数的若干代码;
    return 返回值;
}

- 函数名
  - 合法的 $C++$ 标识符
  - 代表整个代码段
  - 绝对不能省略函数名后的小括号 ()

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

- 形式参数

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

- 形式参数
  - 代表调用函数时需要提供的数据值（占位符）
  - 在函数体内，把“形参”作为一个普通变量去使用。
  - 形式参数必须逐个声明并以逗号（,）隔开

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

- 形式参数可省略
  - 调用时无需提供任何具体值（小括号 () 不能省略）
- 实际参数
  - 函数调用时，(传递) 给 (形式) 参数的实际值。

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, ···)
{
    处理参数的若干代码;
    return 返回值;
}

- 函数体的边界 {}
  - 隔离外部代码
  - 不能省略

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, · · ·)
{
    处理参数的若干代码;
    return 返回值;
}

- 函数体
  - 实现函数功能的语句集合 (做什么，怎么做)

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, $\cdots$)
{
    处理参数的若干代码;
    return 返回值;
}

- *return* 语句
  - 为带返回值的函数返回一个结果
  - 函数终止

函数定义

返回值类型 函数名(数据类型 参数_1, 数据类型 参数_2, $\cdots$)
{
    处理参数的若干代码;
    return 返回值;
}

- $C++$ 函数不能嵌套定义
- $C++$ 函数之间是平行关系 (但有先后)

定义计算两点间距离的函数

- 函数名
- 输入数据
- 计算结果

定义计算两点间距离的函数

- 函数名　　distance
- 输入数据　　参数列表
- 计算结果　　返回值类型

函数调用

- 调用方式

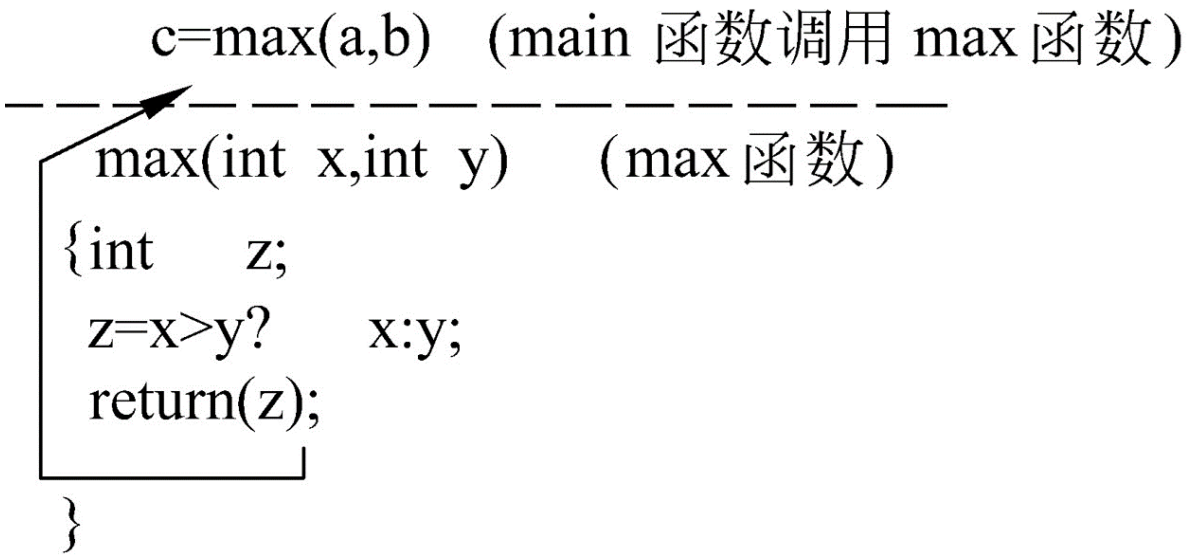
函数名(实际参数)

- 函数名后的小括号不能省略
- 必须提供完全正确的输入数据（实际参数）
  - 数理相同
  - 类型匹配（或赋值兼容）

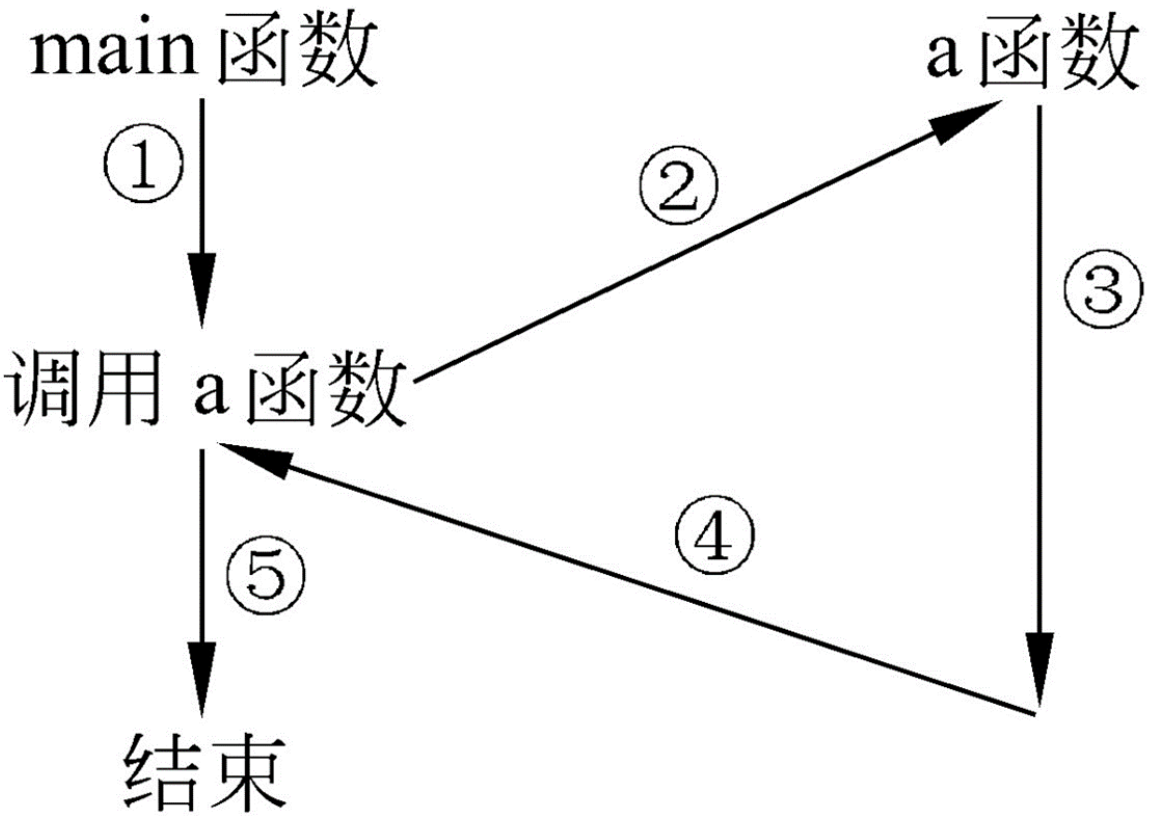
函数调用

- 调用函数 $distance()$ 计算两点间的距离

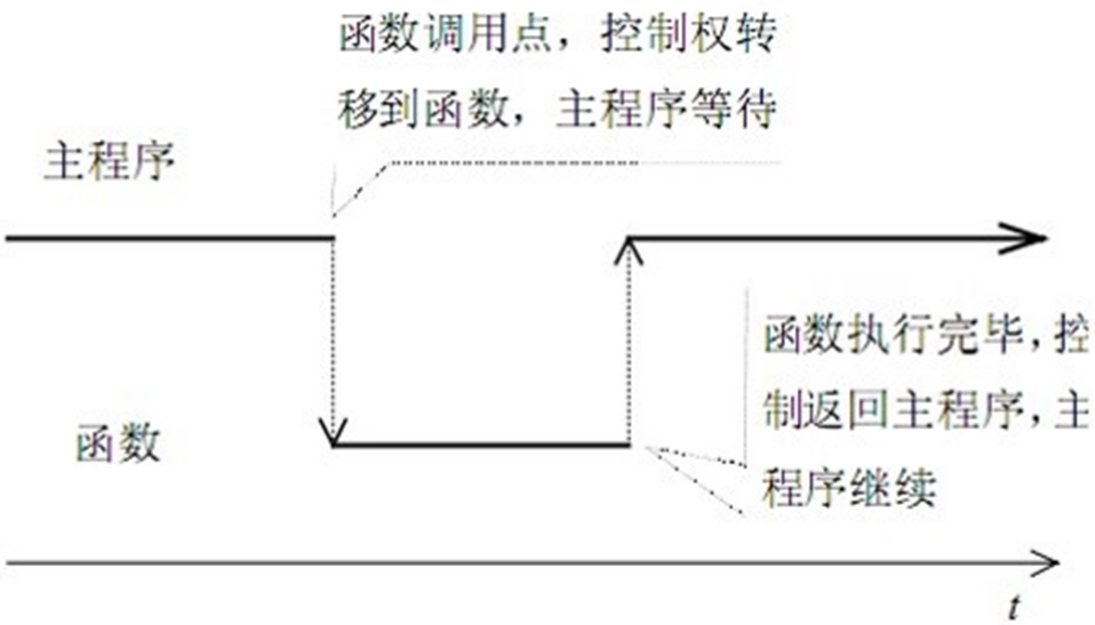
函数调用时的程序流程


c=max(a,b) (main 函数调用 max 函数)
max(int x,int y) (max 函数)
{int z;
z=x>y? x:y;
return(z);
}

函数调用时的程序流程（续）



函数调用时的程序流程（续）

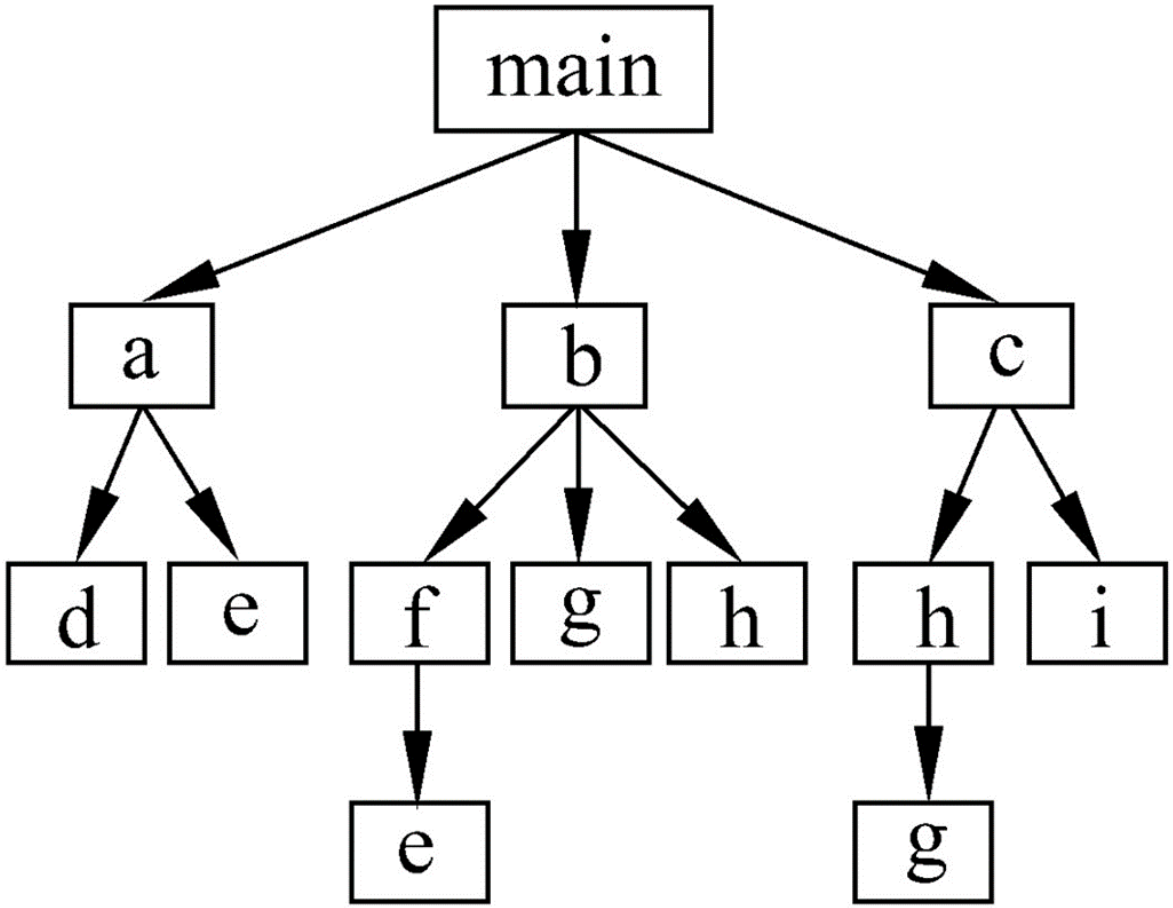


- t　时间

函数调用

- 主函数能调用（除自身外的）其他任何函数，但不能被其他函数调用。
- 主函数由系统（自动）调用
- 一个应用程序只能有一个主函数
- 其他函数之间可以相互调用

应用程序的组成



- 函数不能嵌套定义

主函数的特点

- 任何一个应用程序只能有一个主函数
- main() 函数由系统调用
- 主函数是应用程序执行的唯一入口点
  - 程序从 main() 函数中开始 (主函数的第一条语句)
  - 顺序依次执行 main() 函数中的语句 (顺序结构)
  - 程序在 main() 函数中结束 (主函数中的 return 语句)

void

- 设计一个在屏幕上绘制 9×9 棋盘的函数

void

- 设计一个在屏幕上绘制 9×9 棋盘的函数
- 是否需要输入数据?

void

- 设计一个在屏幕上绘制 9×9 棋盘的函数
- 是否需要输入数据?
  - 否　　参数列表为空 (函数名后的小括号不能省略)

void

- 设计一个在屏幕上绘制 9×9 棋盘的函数
- 是否需要输入数据?
  - 否　　参数列表为空 (函数名后的小括号不能省略)
- 是否需要返回值?

void

- 设计一个在屏幕上绘制 9×9 棋盘的函数
- 是否需要输入数据?
  - 否　　参数列表为空 (函数名后的小括号不能省略)
- 是否需要返回值?
  - 否

void

- 设计一个在屏幕上绘制 9×9 棋盘的函数
- 是否需要输入数据?
  - 否　　参数列表为空 (函数名后的小括号不能省略)
- 是否需要返回值?
  - 否　　函数返回类型为 *void* (函数不会产生一个计算结果)

void 函数的两种形式

```
void 函数名(形式参数列表){
    若干语句;
    return ;
}
void 函数名(形式参数列表){
    若干语句;
}
```

- return 语句
  - 不能返回值
  - 可省略

void 函数的两种形式

```
void 函数名(形式参数列表){
    若干语句;
    return ;
}
void 函数名(形式参数列表){
    若干语句;
}
```

- return 语句
  - 不能返回值
  - 可省略

- void 函数不能参与表达式的组成（只能作为单语句出现）

函数声明

- 函数的使用必须”先定义（或声明）再调用”
- 函数声明语句（函数原型）
  - 不用事先考虑函数之间的相互位置
- 函数声明语句

```
函数头部;
```

函数声明的作用

- 在编译阶段，根据函数原型，对调用函数的合法性进行全面地检查。
- 在函数尚未定义的情况下，事先将该函数的有关信息通知编译系统，从而使得编译能正常进行。
  - 返回值类型
  - 函数名
  - 形式参数（列表）
- 通常，把函数声明放在函数体外，并置于源文件中的所有函数之前。

类型转换的另两个特例

- 参数的类型转换
  - 以形参类型为准
- 返回值的类型转换
  - 以函数返回值的类型为准

return 语句

- return 语句决定函数的返回值
- 函数中可以包含多条 return 语句
  - 只有一条会被执行并由它决定函数的当前返回值
- return 语句最多只能返回一个输出结果
  - 返回类型为 void 的函数可以省略 return 语句中的表达式或者完全省略 return 语句 (当系统执行完该函数的最后一条语句后默认函数调用结束)

函数设计原则

- 一个函数只实现一项功能
- 一个函数的输出可以作为另一个函数的输入

统计单词数量 I

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5     string text="";
6     string line;
7     while(true){
8        getline(cin,line);
9        if(!cin) break;
10       text += line + "\n";
11    }
```

统计单词数量 II

```
12
13      int word_counter = 0;
14      int space_counter=0;
15      bool is_space=false;
16      for(int i=0;i<text.size();++i){
17          if(text[i]==’\n’){
18              word_counter += space_counter+1;
19              space_counter=0;
20          }
21          if(text[i]==’␣’){
22              is_space=true;
```

统计单词数量 III

```
23              while(is_space==true){
24                  if(!(text[i+1]==’␣’)){
25                      is_space=false;
26                  }
27                  else{
28                     i=i+1;
29                  }
30              }
31              space_counter++;
32          }
33      }
```

统计单词数量 IV

```
34
35      cout<<"~~~~~~~~~~~~~~~~~~~~~~~~"<<endl;
36      cout<<text<<endl;
37      cout<<"number␣of␣words␣=␣"<<word_counter<<endl;
38
39      return 0;
40  }
```

统计单词数量（函数版）I

```
1  #include<iostream>
2  using namespace std;
3
4  string readTextFromKeyBoard();
5  int numberOfWords(string p);
6
7  int main()
8  {
9     string text = readTextFromKeyBoard();
10    int words_count = numberOfWords(text);
11
```

统计单词数量（函数版）II

```
12      cout<<"~~~~~~~~~~~~~~~~~~~~~~~~"<<endl;
13      cout<<text<<endl;
14      cout<<"number␣of␣words␣=␣"<<word_counter<<endl;
15      return 0;
16  }
17
18  string readTextFromKeyBoard(){
19      string text="";
20      string line;
21      while(true){
22          getline(cin,line);
```

统计单词数量（函数版）III

```
23        if(!cin) break;
24        text += line + "\n";
25     }
26     return text;
27  }
28
29  int numberOfWords(string p){
30     int word_counter = 0;
31     int space_counter=0;
32     bool is_space=false;
33     for(int i=0;i<p.size();++i){
```

统计单词数量（函数版）IV

```
34          if(p[i]==’\n’){
35              word_counter += space_counter+1;
36              space_counter=0;
37          }
38          if(p[i]==’␣’){
39              is_space=true;
40              while(is_space==true){
41                  if(!(p[i+1]==’␣’)){
42                      is_space=false;
43                  }
44                  else{
```

统计单词数量（函数版）V

```
45                  i=i+1;
46                }
47              }
48              space_counter++;
49          }
50      }
51      return word_counter;
52  }
```

- 代码组织清晰、便于阅读、便于维护。

Q & A