



西安交通大学

XI'AN JIAOTONG UNIVERSITY

钱学森学院 程序设计思想方法与实践

杨义军

计算机科学与技术学院



模块化程序设计

- 模块化程序设计将整体的程序进行模块化分解，将大的任务拆解成小的可以实现的功能，模块化程序以功能的任务拆解为目标，有利于程序的规模控制和可维护

标识符

- 所有的编程元素都需要通过命名区分。
- 标识符：必须是下划线、小写字母、大写字母和数字的字符序列，其首字符必须是下划线、小写字母或者大写字母。
- 合法的标识符：如 `theKernal`, `model_geometry`, `circle`, `math`, `topology`, `seago`
- 不合法的标识符：`construction.geometry`, `¥123`, `#33`, `378`, `b>c`

标识符

- 大小写敏感的语言（PASCAL语言大小写不敏感）。
 - A1和a1是两个不同的标识符
 - 两个标识符是同一标识符当且仅当其在内存中对应的ASCII序列严格一致（效率高）。
-
- 保留字：if, else, int
 - Microsoft C++ 标识符的前 2048 个字符是有意义的，为了编程方便，其长度一般不超过80个字符。

标识符

- 对象或变量名称
- 类、结构或联合名称
- 枚举类型名称
- 类、结构、联合或枚举的成员
- 函数或类成员函数
- typedef 名称
- 标签名称
- 宏名称
- 宏参数

常量

在程序运行过程中，其值不能被改变的量。

整型常量： -2, -1, 0, 1, 2, 3,

实型常量(浮点型)

- 十进制小数形式：如0.34 -56.79 0.0

- 指数形式：如12.34e3 (代表 12.34×10^3)

字符常量：如'?' , 'a' , '1' , 以及不能显示的符号

转义字符：如' \n'

常量

字符串常量：如" boy"

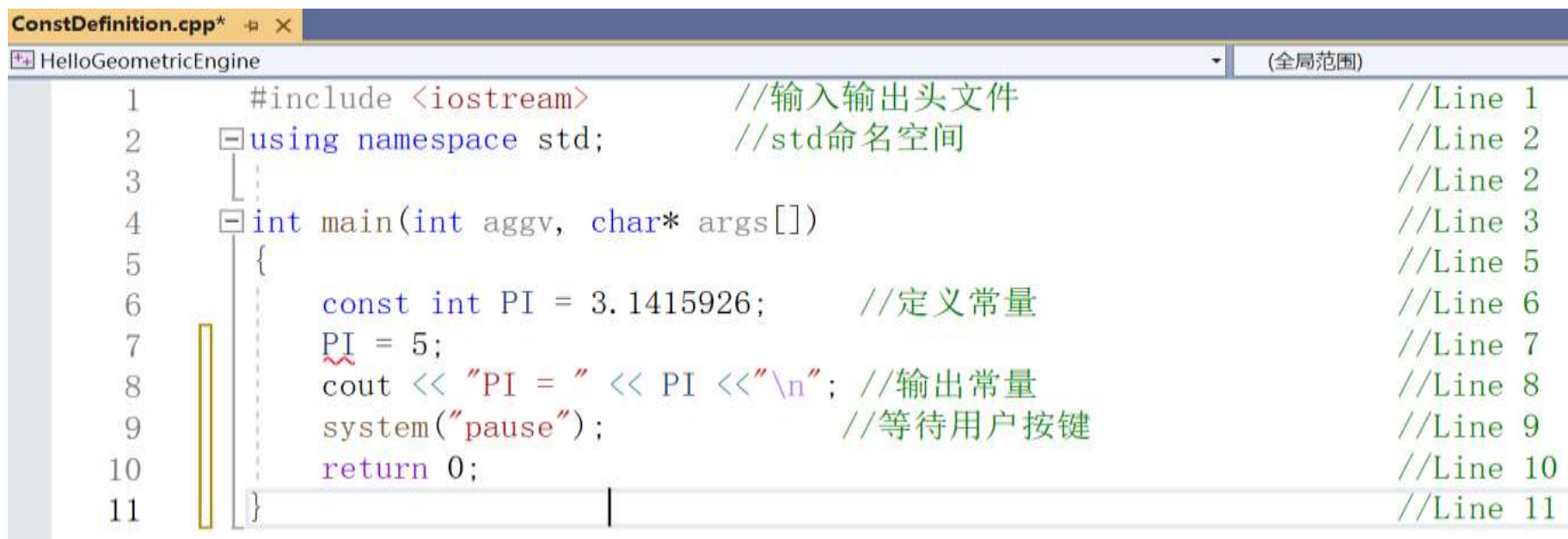
符号常量：#define PI 3.1416

```
ConstDefinition.cpp  + x
HelloGeometricEngine  (全局范围)
1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间           //Line 2
3                               //Line 2
4  int main(int argv, char* args[]) //Line 3
5  {                             //Line 5
6      const int PI = 3.1415926; //定义常量           //Line 6
7      cout << "PI = " << PI << "\n"; //输出常量           //Line 7
8      system("pause");          //等待用户按键           //Line 8
9      return 0;                 //Line 9
10 }
```

常量

常量必须在定义的时候就给它赋初值，否则编译会报编号C2734的错误“如果不是外部的，则必须初始化常量对象”。

如果在常量定义后再修改常量的值（见图4.2 第7行），编译时会报编号为C3892的错误“不能给常量赋值”。



```
ConstDefinition.cpp*  中  X
HelloGeometricEngine  (全局范围)
1      #include <iostream>           //输入输出头文件           //Line 1
2      using namespace std;         //std命名空间           //Line 2
3                                     //Line 2
4      int main(int argv, char* args[]) //Line 3
5      {                             //Line 5
6          const int PI = 3.1415926; //定义常量           //Line 6
7          PI = 5;                   //Line 7
8          cout << "PI = " << PI << "\n"; //输出常量           //Line 8
9          system("pause");          //等待用户按键         //Line 9
10         return 0;                 //Line 10
11     }
```


#define 定义常量

#define 宏定义常量，通过宏定义实现，在编译的预处理阶段会将对应的常量名进行字符串替换。

```
ConstDefinition_define.cpp  ✕
+ 杂项文件  (全局范围)
1  #include <stdio.h>           //输入输出头文件           //Line 1
2  #include <iostream>          //Line 2
3  using namespace std;        //Line 3
4  #define PI 3.1415926        //Line 4
5  int main(int argv, char* args[]) //Line 5
6  {                             //Line 6
7      //const double PI = 3.1415926; //Line 7
8      cout << "PI = " << PI;    //Line 8
9      system("pause");         //等待用户按键           //Line 9
10     return 0;                //Line 10
11 }
```

常量与宏定义#define的区别

编译器处理阶段

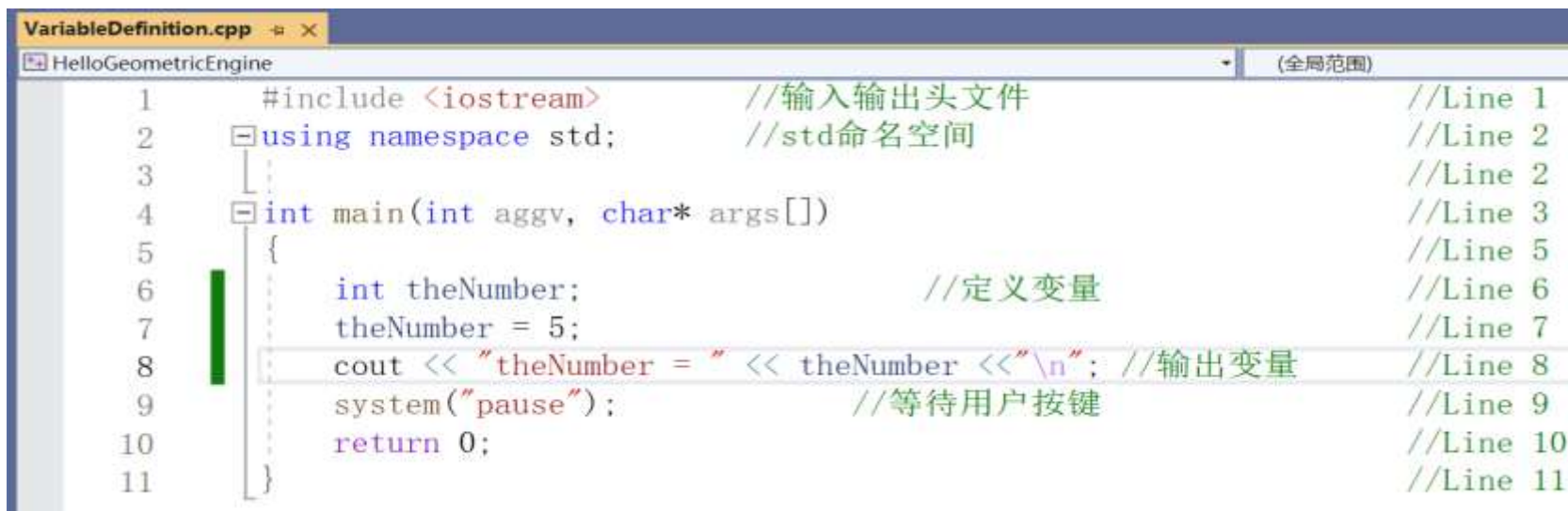
- #define是编译预处理器预处理阶段完成，const是编译器开始编译程序阶段完成，因此#define先于const处理。

语法语义方面

- #define是宏定义，特点是简单字符串替换，其定义的常量值没有类型限定，也不做类型检查，在出现宏名称的地方直接展开。const是保留字，本质上会定义一个只读变量，不可以更改。

变量

- 在程序运行过程中，其值可以改变的量为变量。
- 变量必须先定义，然后再被使用。
- 变量名相当于给内存的某一段空间（其大小由变量类型确定）起了一个名字，通过变量名可以对变量进行引用和赋值。
- 语法定义：类型说明符 变量1标识符，变量2标识符，…；



```
VariableDefinition.cpp x
HelloGeometricEngine (全局范围)
1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间               //Line 2
3  ...                           //Line 2
4  int main(int argv, char* args[]) //Line 3
5  {                             //Line 5
6      int theNumber;           //定义变量                 //Line 6
7      theNumber = 5;           //Line 7
8      cout << "theNumber = " << theNumber << "\n"; //输出变量 //Line 8
9      system("pause");         //等待用户按键             //Line 9
10     return 0;                //Line 10
11 }
```

查看和修改变量的值

VariableDefinition.cpp (全局范围)

```
1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;           //std命名空间             //Line 2
3                                     //Line 2
4  int main(int argv, char* args[]) //Line 3
5  {                               //Line 5
6      int theNumber;              //定义变量               //Line 6
7      theNumber = 5;              //Line 7
8      cout << "theNumber = " << theNumber << "\n"; //输出变量 //Line 8
9      system("pause");            //等待用户按键           //Line 9
10     return 0;                  //Line 10
11 }
```

VariableDefinition.cpp 局部变量

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
argv	1	int
args	0x0000020d604ac830 0x0000020d604ac840 "D:\workingtu\图书\程序设计思想方法与实践\教材\图书初稿...	char **
theNumber	525	int

自动窗口 局部变量 监视 1

```
1  #inc
2  usin
3
4  int main(int argv, char* args[]) //Line 3
5  {                               //Line 5
6      int theNumber;              //定义变量               //Line 6
7      theNumber = 5;              //Line 7
8      cout << theNumber << "\n"; //输出变量 //Line 8
9      system("pause");            //等待用户按键           //Line 9
10     return 0;                  //Line 10
11 }
```

局部变量

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
argv	1	int
args	0x0000020d604ac830 0x0000020d604ac840 "D:\workingtu\图书\程序设计思想方法与实践\教材\图书初稿...	char **
theNumber	525	int

自动窗口 局部变量 监视 1

查看和修改变量的值

```
VariableDefinition.cpp [X]
HelloGeometricEngine (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间           //Line 2
3  ...                           //Line 2
4  int main(int argv, char* args[]) //Line 3
5  {                             //Line 5
6      int theNumber;           //定义变量           //Line 6
7      theNumber = 5;           //Line 7
8      cout << theNumber << "\n"; //输出变量           //Line 8
9      system("pause");         //等待用户按键           //Line 9
10     return 0;                //Line 10
11 }
```

局部变量

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
argv	1	int
args	0x0000020d604ac830 {0x0000020d604ac840 "D:\\work\\xjtu\\图书\\程序设计思想方法与实践\\教材\\图书初稿...	char **
theNumber	5	int

自动窗口 局部变量 监视 1

数据类型

- 数据类型就是对数据分配存储单元的安排，包括存储单元的长度(占多少字节)以及数据的存储形式，不同的类型分配不同的长度和存储形式。
 - 整数类型、浮点类型、枚举类型、空类型、派生类型。
 - 整数类型：基本整型、短整型、长整型、双长整型、字符型、布尔型
 - 浮点类型：单精度浮点型、双精度浮点型和复数浮点型
 - 派生类型包括指针类型、数组类型、结构体类型、共用体类型和函数类型
-
- void类型：void 类型描述值的空集。 无法指定类型为 void 的变量。
void 类型主要用于声明不返回值的函数，或用于声明指向非类型化或任意类型化数据的一般指针。
-
- 布尔类型：bool 类型值可以是 true 和 false。 bool 类型占用一个字节

整型

字符类型：char 类型是一种字符表示类型，可有效地对基本执行字符集的成员进行编码。C++ 编译器将 char, signed char 和 unsigned char 类型的变量视为不同类型。整形类型前面可以添加无符号保留字 unsigned 将其定为无符号类型, signed 为有符号数。

整数类型：int 类型是默认的基本整数类型。带符号整数表示形式可以同时保存正值和负值。它默认使用，或者在存在 signed 修饰符关键字时使用。unsigned 修饰符关键字指定一个只能保存非负值的无符号表示形式

`1 == sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)`

整型

变量类型↵	占用字节数↵
bool, char, char8_t, unsigned char, signed char, __int8↵	1↵
char16_t, __int16, short, unsigned short, <u>wchar_t</u> , <u>__wchar_t</u> ↵	2↵
char32_t, float, __int32, int, unsigned int, long, unsigned long↵	4↵
double, __int64, long double, long <u>long</u> , unsigned long <u>long</u> ↵	8↵

整型同一类型

- short, short int, signed short, signed short int
- unsigned short, unsigned short int
- int, signed, signed int
- unsigned, unsigned int
- long, long int, signed long, signed long int
- unsigned long, unsigned long int
- long long, long long int, signed long long, signed long long int
- unsigned long long, unsigned long long int

整型

VariableDefinition2.cpp

HelloGeometricEngine

(全局范围)

```
1    #include <iostream>           //输入输出头文件           //Line 1
2    using namespace std;         //std命名空间           //Line 2
3                                   //Line 3
4    int main(int argv, char* args[]) //Line 4
5    {                             //Line 5
6        cout << "int " << sizeof(int) << "\n";           //Line 6
7        cout << "short int " << sizeof(short int) << "\n"; //Line 7
8        cout << "long int " << sizeof(long int) << "\n";  //Line 8
9        cout << "long long int " << sizeof(long long int) << "\n"; //Line 9
10       cout << "char 大小 " << sizeof(char) << "\n";      //Line 10
11       cout << "bool 大小 " << sizeof(bool) << "\n";      //Line 11
12       system("pause");          //等待用户按键           //Line 12
13       return 0;                 //Line 13
14    }
```

整型表示范围

表4.2 部分整型数范围

类型	类型说明符	内存所占字节数	数的范围
基本整型	int	4	$-2^{31} \sim 2^{31}-1$
短整型	short	2	$-2^{15} \sim 2^{15}-1$
无符号整型	unsigned int	4	$0 \sim 2^{32}-1$
无符号短整型	unsigned short	2	$0 \sim 2^{16}-1$
字符型	char	1	$-128 \sim 127$
无符号字符型	unsigned char	1	$0 \sim 255$

整型

- 符号数的表达范围要比对应的有符号数的表达范围扩大了一倍，其原因在于将首位符号标志（1为负，0为正）用于表示数字。字符型变量在c语言中当作一个字节的整型来进行处理。

其中' 1' 和1在c语言中表达不同的概念，' 1' 表达字符1对应的ASCII码（49），1表达整数1。在C语言中可以将字符直接赋值给整型变量，其效果是将字符对应的ASCII码值赋值给整型变量

浮点

浮点型数据是用来表示具有小数点的实数，c语言中分为单精度、双精度和长双精度浮点数。浮点类型使用 IEEE-754 表示形式在各种数量级上提供小数值的近似值。



类 型	符号符	阶码	尾数码	总位数	偏置值	
					十六进制	十进制
短浮点数	1	8	23	32	7FH	127
长浮点数	1	11	52	64	3FFH	1023
临时短浮点数	1	15	64	80	3FFFH	16383

浮点

单精度和双精度格式中，假定在小数部分中有前导 1。小数部分称为“有效数字”（有时亦称为“尾数”）。这个前导 1 没有存储在内存中，所以有效数字实际上是 24 或 53 位，尽管少存储了 1 位。

表4.4 浮点数存储格式

值	存储格式
单精度 (single-precision)	符号位, 8 位指数, 23 位有效数
双精度	符号位, 11 位指数, 52 位有效数

运算符和表达式

表达式是用于实现以下一个或多个目的而使用的运算符和操作数的序列：

- 计算来自操作数的值。
- 指定对象或函数。
- 产生“副作用”。（副作用是除表达式计算之外的任何操作 - 例如，修改对象的值。

基本算数运算符

$+$: 正号运算符(单目运算符)

$-$: 负号运算符(单目运算符)

$*$: 乘法运算符

$/$: 除法运算符

$\%$: 求余运算符

$+$: 加法运算符

$-$: 减法运算符

两个整数相除的结果为整数, 比如 $5/4 = 1$, 小数部分被舍弃。如果除数或被除数中有一个为负值, VC++采取“向零取整”的方法。例如, $-5/4 = -1$; $-4/5 = 0$.

基本算数运算符

% 运算符要求参加运算的运算对象(即操作数)为整数，结果也是整数。如 $7\%3$ ，结果为1。

```
IntegerDivide.cpp
HelloGeometricEngine (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间               //Line 2
3                               //Line 3
4  int main(int argv, char* args[]) //Line 4
5  {                             //Line 5
6      int theNumber;           //定义变量                 //Line 6
7      theNumber = 5 / 4;        //Line 7
8      cout << "5 / 4 = " << theNumber << "\n"; //整数除法       //Line 8
9      theNumber = 4 / 5;        //Line 9
10     cout << "4 / 5 = " << theNumber << "\n"; //整数除法       //Line 10
11     theNumber = -4 / 5;        //Line 11
12     cout << "-4 / 5 = " << theNumber << "\n"; //整数除法       //Line 12
13     theNumber = -5 / 4;        //Line 13
14     cout << "-5 / 4 = " << theNumber << "\n"; //整数除法       //Line 14
15     theNumber = 7 % 2;        //Line 15
16     cout << "7 % 2 = " << theNumber << "\n"; //整数取余       //Line 16
17     theNumber = -5 % 4;        //Line 17
18     cout << "-5 % 4 = " << theNumber << "\n"; //整数取余       //Line 18
19     system("pause");          //等待用户按键             //Line 19
20     return 0;                 //Line 20
21 }
```

自增、自减运算符

- 作用是使变量的值加 1 或减 1，前缀递增运算符（++）向其操作数增加 1；此递增值是表达式的结果。操作数必须是类型不为 `const` 的 l-value。结果是与操作数相同类型的左值。前缀递减运算符（--）与前缀递增运算符类似，只不过操作数将减少 1，并且结果是递减值。
- 前缀和后缀递增和递减运算符均会影响其操作数。它们之间的主要差异是递增或递减在表达式的计算中出现的顺序。
- 在前缀形式中，将在表达式计算中使用值之前进行递增或递减，因此表达式的值与操作数的值不同。
- 在后缀形式中，将在表达式计算中使用值之后进行递增或递减，因此表达式的值与操作数的值相同。

自增、自减运算符

- ++i, --i: 在使用i之前, 先使i的值加(减) 1
- i++, i--: 在使用i之后, 使i的值加(减) 1

```
Incrementan...perator.cpp  X
HelloGeometricEngine (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间           //Line 2
3  [                             //Line 3
4  int main(int argv, char* args[]) //Line 4
5  {                             //Line 5
6      int theNumber = 4;         //定义变量           //Line 6
7      cout << "++theNumber = " << ++theNumber << endl; //递增           //Line 7
8      system("pause");           //等待用户按键       //Line 8
9      return 0;                 //Line 9
10 }
```

算数表达式

- 用算术运算符和括号将运算对象（也称操作数）连接起来的、符合C++语法规则的式子，称为算术表达式。
- 运算对象包括常量、变量、函数等。运算符优先级指定了包含多个运算符的表达式中的运算顺序。
- 运算符关联性指定了在包含多个具有相同优先级的运算符的表达式中，操作数是与其左侧还是右侧的操作数组合。
- 表4.5显示 C++ 运算符的优先级和关联性（从最高优先级到最低优先级）。优先级级别编号相同的运算符具有等同的优先级，除非由括号显式施加另一种关系

混合运算

- `+`、`-`、`*`、`/` 运算的两个数中有一个数为float或double型，结果是double型。系统将float型数据都先转换为double型，然后进行运算；
- 如果int型与float或double型数据进行运算，先把int型和float型数据转换为double型，然后进行运算，结果是double型
- 字符型数据与整型数据进行运算，就是把字符的ASCII代码与整型数据进行运算。

强制类型转换

- 强制类型转换运算符的一般形式为
- (类型名) (表达式)

```
expre_CastOperator.cpp + x
HelloGeometricEngine (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间             //Line 2
3
4  int main(int argv, char* args[]) //Line 4
5  {                             //Line 5
6      double theDoubleNumber = 3.1; //定义浮点变量         //Line 6
7      int theIntNumber;           //定义整型变量           //Line 7
8      cout << "theDoubleNumber = " << theDoubleNumber << endl; //Line 8
9      theIntNumber = (int)theDoubleNumber; //将theDoubleNumber整数部分赋值给theIntNumber //Line 9
10     cout << "theIntNumber = " << theIntNumber << endl; //Line 10
11     system("pause");           //等待用户按键           //Line 11
12     return 0;                  //Line 12
13 }
```

C++ 语句

C++ 语句是控制操作对象的方式和顺序的程序元素。C++ 语句将按顺序执行，除非表达式语句、选择语句、迭代语句或跳转语句特意修改了顺序。

- 1. 表达式语句。
- 2. Null 语句。
- 3. 复合语句。
- 4. 选择语句。
- 5. 循环语句。
- 6. 跳转语句。
- 7. 声明语句。

C++语句

变量 赋值运算符 表达式;

```
CaculateCircleArea.cpp  x
HelloGeometricEngine (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间               //Line 2
3  [
4  int main(int argv, char* args[]) //Line 4
5  {                               //Line 5
6      double theRadius;         //定义圆的半径             //Line 6
7      double theArea;           //圆的面积               //Line 7
8      const double PI = 3.1415926; //PI                   //Line 8
9      cout << "请输入圆的半径: "; //输出提示             //Line 9
10     cin >> theRadius;           //用户输入半径         //Line 10
11     theArea = PI * theRadius * theRadius; //赋值语句             //Line 11
12     cout << "圆的面积为 = " << theArea << endl; //输出结果             //Line 12
13     system("pause");           //等待用户按键         //Line 13
14     return 0;                 //Line 14
15 }
```


三种基本结构

C++中有三种基本结构，顺序结构、选择结构和循环结构。

关系运算符

用来对两个数值进行比较的比较运算符。C++里面有六个关系运算符：

- 小于号 (<)
- 大于号 (>)
- 小于或等于 (<=)
- 大于或等于 (>=)
- 等于 (==)
- 不等于 (!=)

其中前四个操作符的优先级相等，大于后面两个操作符（后面两个操作符的优先级也相等）的优先级。

关系运算符具有从左到右的关联性。

关系运算符的两个操作数必须是算术或指针类型。 它们将生成 `bool` 类型的值。 如果表达式中的关系为 `false`，则返回的值为 `false` (0)；否则返回的值为 `true` (1)

逻辑运算符

逻辑与&&，逻辑或||和逻辑非!

逻辑“与”具有从左到右的关联性。

逻辑“与”运算符的操作数不需要具有相同的类型，但它们必须是布尔值、整数或指针类型。操作数通常为关系或相等表达式。

逻辑与短路计算：第一个操作数将完全计算，仅当第一个操作数的计算结果为 true（非零）时，才会计算第二个操作数。当逻辑“与”表达式为 false 时，这种计算方式可消除不必要的对第二个操作数的计算。

逻辑或

逻辑“或”具有从左向右的关联性。

逻辑“或”运算符的操作数不需要具有相同的类型，但它们必须是布尔值、整数或指针类型。操作数通常为关系或相等表达式。第一个操作数将完全计算，仅当第一个操作数的计算结果为 `false` 时计算第二个操作数，因为当逻辑“或”表达式为 `true` 时不需要计算。这称作“短路”计算。

$$x == w \ || \ x == y \ || \ x == z$$

逻辑非运算符 (`!`) 反转其操作数的含义。操作数必须是算法或指针类型（或计算结果为算法或指针类型的表达式）。操作数将隐式转换为类型 `bool`。如果已转换的操作数是 `false`，则结果是 `true`；如果已转换的操作数是 `true`，则结果是 `false`。结果的类型为 `bool`。

运算符优先级

赋值运算符 < 逻辑运算符&&和|| < 关系运算符 < 算数运算符 < !

```
AndOperator.cpp x
HelloGeometricEngine (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间               //Line 2
3                                  //Line 3
4  //主函数                       //Line 4
5  int main(int argv, char* args[]) //Line 5
6  {                             //Line 6
7      //定义变量                 //Line 7
8      int theIntValueA = 5;      //Line 8
9      int theIntValueB = 10;     //Line 9
10     int theIntValueC = 15;     //Line 10
11                                  //Line 11
12     //计算输出逻辑表达式       //Line 12
13     cout << "真表达式: theIntValueA < theIntValueB && theIntValueB < theIntValueC 等于"; //Line 13
14     cout << (theIntValueA < theIntValueB && theIntValueB < theIntValueC) << endl; //Line 14
15     cout << "假表达式: theIntValueA > theIntValueB && theIntValueB < theIntValueC 等于"; //Line 15
16     cout << (theIntValueA > theIntValueB && theIntValueB < theIntValueC) << endl; //Line 16
17                                  //Line 17
18     //                          //Line 18
19     system("pause");           //等待用户按键             //Line 19
20     return 0;                  //Line 20
21 }
```

条件运算符

表达式1 ? 表达式2 : 表达式3

条件运算符（`:`）是一个三元运算符（采用三个操作数）。条件运算符按以下方式运行：

- 第一个操作数隐式转换为 `bool`。计算该操作数。
- 如果第一个操作数的计算结果为 `true` (1)，则计算第二个操作数。
- 如果第一个操作数的计算结果为 `false` (0)，则计算第三个操作数。

条件表达式具有从右到左的关联性。

条件运算符

表达式1 ? 表达式2 : 表达式3

```
ConditionalOperator.cpp  x
HelloGeometricEngine  (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间             //Line 2
3                               //Line 3
4  //主函数                     //Line 4
5  int main(int argv, char* args[]) //Line 5
6  {                             //Line 6
7      //定义变量               //Line 7
8      int theIntValueA = 5;     //Line 8
9      int theIntValueB = 10;    //Line 9
10     int theIntValueC = 15;    //Line 10
11                               //Line 11
12     //计算输出逻辑表达式     //Line 12
13     cout << "theIntValueA < theIntValueB ? theIntValueA : theIntValueB 等于"; //Line 13
14     cout << (theIntValueA < theIntValueB ? theIntValueA : theIntValueB) << endl; //Line 14
15                               //Line 15
16     //                         //Line 16
17     system("pause");          //等待用户按键             //Line 17
18     return 0;                 //Line 18
19 }
```

if语句

if语句，实现两个分支的选择结构。if-else 语句控制条件分支。 仅当 条件计算结果为非零值（或 真）时，才会执行 if分支 中的语句。 如果条件的值为零（或 假），则执行else 语句后面的语句（如果有）。

```
ifStatement.cpp
HelloGeometricEngine (全局范围)

1  #include <iostream> //输入输出头文件 //Line 1
2  using namespace std; //std命名空间 //Line 2
3  //Line 3
4  //主函数 //Line 4
5  int main(int argc, char* args[]) //Line 5
6  { //Line 6
7      //二次方程的系数 //Line 7
8      double theDoubleA, theDoubleB, theDoubleC; //Line 8
9      cout << "请输入ax^2 + bx + c = 0的系数: " << endl; //Line 9
10     cin >> theDoubleA >> theDoubleB >> theDoubleC; //Line 10
11     //判别式的值 //Line 11
12     double theDoubleDelta = theDoubleB * theDoubleB - 4 * theDoubleA * theDoubleC; //Line 12
13     double theDoubleFirstRoot, theDoubleSecondRoot; //Line 13
14     if (theDoubleDelta < 0) //Line 14
15     { //Line 15
16         cout << "判别式小于零，没有实根。"; //Line 16
17     } //Line 17
18     else //Line 18
19     { //Line 19
20         theDoubleFirstRoot = (-theDoubleB + sqrt(theDoubleDelta)) / (2 * theDoubleA); //Line 20
21         theDoubleSecondRoot = (-theDoubleB - sqrt(theDoubleDelta)) / (2 * theDoubleA); //Line 21
22         cout << "第一个根: " << theDoubleFirstRoot << ", 第二个根: " << theDoubleSecondRoot; //Line 22
23     } //Line 23
24     system("pause"); //等待用户按键 //Line 24
25     return 0; //Line 25
26 }
```


switch语句

switch语言允许根据整型表达式的值在多个代码段中进行选择。switch 语句使控件根据条件的值转移到其语句正文中的一个标记语句。条件必须是整数型或明确转换为整数型的类类型。

switch 语句体由一系列 case 标签和一个 optional default（可选）标签组

一个标签语句是其中一个标签和后面的语句。标记语句不是语法需求，但如果它们不存在，switch 语句是无意义的。case 语句中没有两个常量表达式的值可能计算为相同的值。

default 标签只能出现一次。default 语句通常放在末尾，但它可以出现在 switch 语句正文中的任何位置。case 或 default 标签只能显示在 switch 语句内部。每个 case 标签的 constant-expression 都转换为与条件相同的常量值。然后，与条件比较相等情况。控件将传递到 case constant-expression 值后的第一个语句，该语句与条件的值匹配。

switch语句

条件↵	操作↵	↵
转换后的值与提升的控制表达式的 <u>值匹</u> ↵	控制将转移到跟在该标签后面的语句↵	↵
没有常量与 <code>case</code> 标签中的常量匹配；存在 <code>default</code> 标签↵	控件将转移到 <code>default</code> 标签↵	↵
没有常量与 <code>case</code> 标签中的常量匹配；不存在 <code>default</code> 标签↵	控件将转移到 <code>switch</code> 语句之后的语句↵	↵

switch语句

```
SwitchStatement.cpp
HelloGeometricEngine
1 #include <iostream> //输入输出头文件 //Line 1
2 using namespace std; //std命名空间 //Line 2
3 // //Line 3
4 //主函数 //Line 4
5 int main(int argc, char* args[]) //Line 5
6 { //Line 6
7     //定义字符串 //Line 7
8     const char* theString = "I am a Geometric Engine!"; //Line 8
9     const char* headPointer = theString; //Line 9
10    //定义统计变量 //Line 10
11    int theUpperCase_A, theLowerCase_A, theOtherCase; //Line 11
12    //定义字符变量 //Line 12
13    char theCurrentCharacter; //Line 13
14    // //Line 14
15    //赋初值 //Line 15
16    theUpperCase_A = theLowerCase_A = theOtherCase = 0; //Line 16
17    // //Line 17
18    //遍历所有的字符串字符 //Line 18
19    while (theCurrentCharacter = *theString++) //Line 19
20    { //Line 20
21        switch (theCurrentCharacter) //Line 21
22        { //Line 22
23            case 'A': //Line 23
24                theUpperCase_A++; //Line 24
25                break; //Line 25
26            case 'a': //Line 26
27                theLowerCase_A++; //Line 27
28                break; //Line 28
29            default: //Line 29
30                theOtherCase++; //Line 30
31        } //Line 31
32    } //Line 32
33    //输出信息 //Line 33
34    cout << "UpperCase A: " << theUpperCase_A << endl; //Line 34
35    cout << "LowerCase A: " << theLowerCase_A << endl; //Line 35
36    cout << "OtherCase: " << theOtherCase << endl; //Line 36
37    // //Line 37
38    system("pause"); //等待用户按键 //Line 38
39    return 0; //Line 39
40 }
```

循环语句

C++ 提供四个迭代语句 - while、do、for 和 range-based for。

根据一些循环终止条件，迭代语句会导致语句（或复合语句）被执行零次或多次。 当这些语句是复合语句时，除非遇到 break 语句或 continue 语句，否则将按顺序执行它们

语句↵	计算位置↵	初始化↵	增量↵
while↵	循环的顶部↵	否↵	否↵
do↵	循环的底部↵	否↵	否↵
for↵	循环的顶部↵	是↵	是↵
基于范围的 for↵	循环的顶部↵	是↵	是↵

while语句

```
while ( expression )  
    statement
```

表达式的测试在每次执行循环之前开始，因此，while 循环执行零次或多次。表达式必须是整型类型、指针类型或可以明确转换为整型或指针类型的类型。当执行语句正文中的 break、goto 或 return 时，while 循环也可以终止。使用 continue 可在不退出 while 循环的情况下终止迭代。continue 将控制转移到 while 循环的下一次迭代。

while语句

```
whilestatement.cpp  (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间               //Line 2
3                                  //Line 3
4  //主函数                     //Line 4
5  int main(int argv, char* args[]) //Line 5
6  {                             //Line 6
7      //定义变量               //Line 7
8      int theIterNumber = 1;    //Line 1
9      int theSum = 0;           //Line 2
10                                  //Line 3
11  //循环语句                   //Line 4
12  while (theIterNumber <= 100)  //Line 5
13  {                             //Line 6
14      theSum += theIterNumber++; //Line 7
15  }                             //Line 7
16                                  //Line 7
17  //输出求和结果               //Line 12
18  cout << "从1到100求和得: " << theSum << endl;; //Line 13
19                                  //Line 15
20  //                           //Line 16
21  system("pause");              //等待用户按键           //Line 17
22  return 0;                     //Line 18
23 }
```

do语句

do

statement

while (expression) ;

终止条件的测试将在每次执行循环后进行；因此 do-while 循环将执行一次或多次，具体取决于终止表达式的值。do-while 语句还可在语句体中执行 break、goto 或 return 语句时终止。

1. 执行语句体。
2. 接着，计算 expression。如果 expression 为 false，则 do-while 语句将终止，控制权将传递到程序中的下一条语句。如果 expression 为 true（非零），则将从第 1 步开始重复此过程。

for语句

```
for ( init-expression ; cond-expression ; loop-expression )  
    statement
```

使用 for 语句可构建必须执行指定次数的循环。for 语句包括三个可选部分



```
forstatement.cpp  
HelloGeometricEngine  
1  #include <iostream>           //输入输出头文件           //Line 1  
2  using namespace std;         //std命名空间               //Line 2  
3                               //Line 3  
4  //主函数                     //Line 4  
5  int main(int argv, char* args[]) //Line 5  
6  {                             //Line 6  
7      //定义变量               //Line 7  
8      int theIterNumber = 1;    //Line 8  
9      int theSum = 0;           //Line 9  
10                               //Line 10  
11     //循环语句                //Line 11  
12     for (theIterNumber = 1; theIterNumber <= 100; theIterNumber++) //Line 12  
13     {                          //Line 13  
14         theSum += theIterNumber; //Line 14  
15     }                          //Line 15  
16                               //Line 16  
17     //输出求和结果            //Line 17  
18     cout << "从1到100求和得: " << theSum << endl;; //Line 18  
19                               //Line 19  
20     //                          //Line 20  
21     system("pause");           //等待用户按键             //Line 21  
22     return 0;                  //Line 22  
23 }
```


for语句

语法名称↵	执行时间↵	说明↵
<u>init-expression</u> ↵	在 for 语句的任何其他元素之前， <u>init-expression</u> 仅执行一次。控制权然后传递给 <u>cond-expression</u> 。↵	通常用来初始化循环索引。它可以包含表达式或声明。↵
<u>cond-expression</u> ↵	在执行 statement 的每次迭代之前，包括第一次迭代。statement 只在 <u>cond-expression</u> 的计算结果为 true（非零）时执行。↵	计算结果为整数型或明确转换为整数型的 <u>类类型</u> 的表达式。通常用于测试循环终止条件。↵
loop-expression↵	在 statement 的每次迭代结束时。执行 loop-expression 后，将计算 <u>cond-expression</u> 。↵	通常用于循环索引递增。↵

基于范围的for语句

for (for-range-declaration:expression)

语句

使用基于范围的 for 语句构造必须在“范围”中执行的循环，它定义为可循环访问的任何内容

```
regionforstatement.cpp  x
HelloGeometricEngine  (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间             //Line 2
3                                  //Line 3
4  //主函数                     //Line 4
5  int main(int argv, char* args[]) //Line 5
6  {                             //Line 6
7      //定义变量               //Line 7
8      int theIntegerArray[5] = { 1, 2, 4, 6, 7 };           //Line 8
9      int theSum = 0;         //Line 9
10                                  //Line 10
11     //循环语句               //Line 11
12     for (auto theIterNumber : theIntegerArray)           //Line 12
13     {                                                           //Line 13
14         theSum += theIterNumber;                             //Line 14
15     }                                                           //Line 15
16                                  //Line 16
17     //输出求和结果           //Line 17
18     cout << "数组求和得: " << theSum << endl;;           //Line 18
19                                  //Line 19
20     //                       //Line 20
21     system("pause");         //等待用户按键               //Line 21
22     return 0;               //Line 22
23 }
```

auto类型

编译器可以根据表达式能确定类型的变量

break语句

- break 语句与 switch 条件语句以及 do、for 和 while 循环语句配合使用。用break语句可以提前终止循环，其可以使用在循环体内部。一般情况其前面会跟一个条件判断，使得在满足某种条件的情况循环体提前结束。控制权将传递给终止语句之后的语句（如果有的话）。
- 在嵌套语句中，break 语句只终止直接包围它的 do、for、switch 或 while 语句。 你可以用 return 或 goto 语句从较深嵌套的结构转移控制权。

break语句

```
breakStatement.cpp  x
HelloGeometricEngine  (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间           //Line 2
3  //主函数           //Line 3
4  int main(int argv, char* args[]) //Line 4
5  {
6      //调用函数           //Line 5
7      for (int theIntValue = 1; theIntValue < 10; theIntValue++) //Line 6
8      {
9          if (theIntValue == 4) //Line 7
10             break;           //Line 8
11             cout << theIntValue << endl; //Line 9
12         }
13         system("pause");       //等待用户按键           //Line 10
14         return 0;             //Line 11
15     }
16
17
18
```

continue语句

将不会执行当前迭代中的所有剩余语句，执行循环的下一次迭代，如下所示

- 1. 在 do 或 while 循环中，下一个迭代首先会重新计算 do 或 while 语句的控制表达式。
- 2. 在 for 循环中（使用语法 for(<init-expr> ; <cond-expr> ; <loop-expr>)），将执行 <loop-expr> 子句。然后，重新计算 <cond-expr> 子句，并根据结果确定该循环结束还是进行另一个迭代。

continue语句

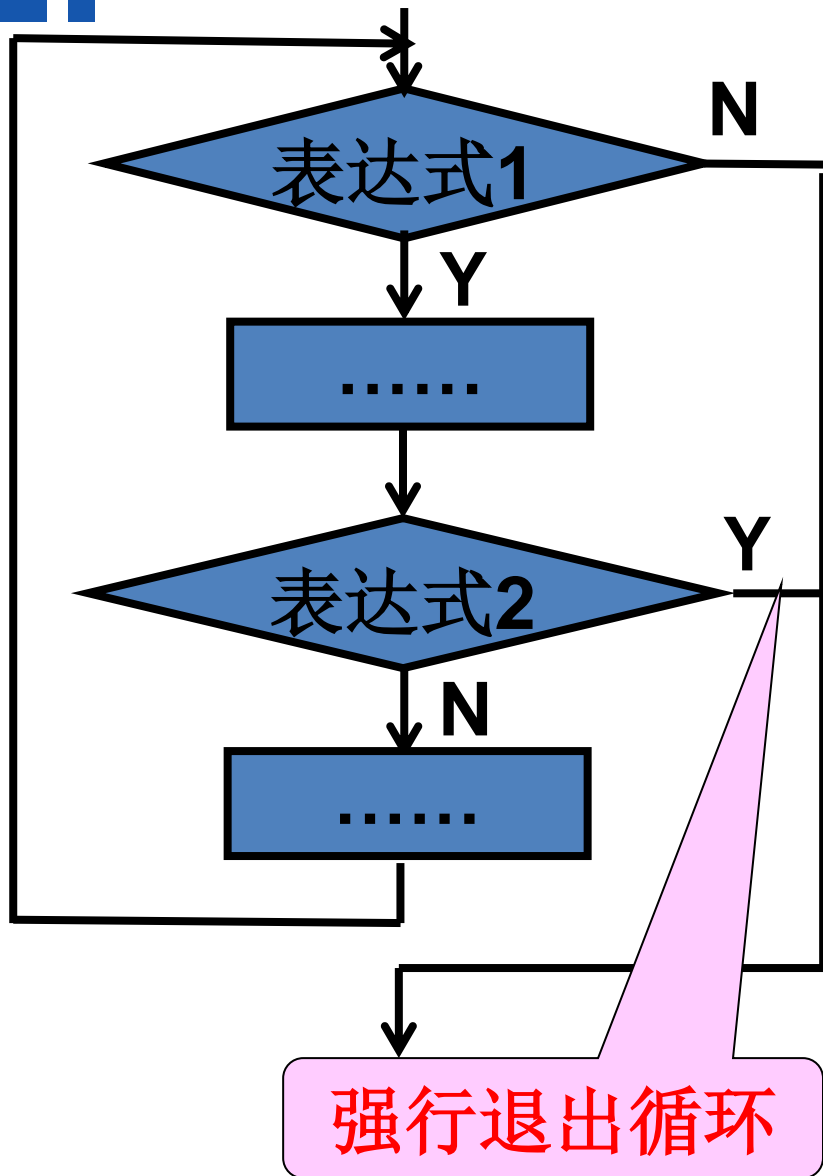
```
continueStatement.cpp  X
HelloGeometricEngine (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间           //Line 2
3                                   //Line 3
4  //主函数           //Line 4
5  int main(int argv, char* args[]) //Line 5
6  {                             //Line 6
7                                   //Line 7
8      //调用函数           //Line 8
9      for (int theIntValue = 1; theIntValue < 10; theIntValue++) //Line 9
10     {                          //Line 10
11         if (theIntValue == 4) //Line 11
12             continue;        //Line 12
13         cout << theIntValue << endl; //Line 13
14     }                          //Line 14
15                                   //Line 15
16     system("pause");          //等待用户按键           //Line 16
17     return 0;                 //Line 17
18 }
```

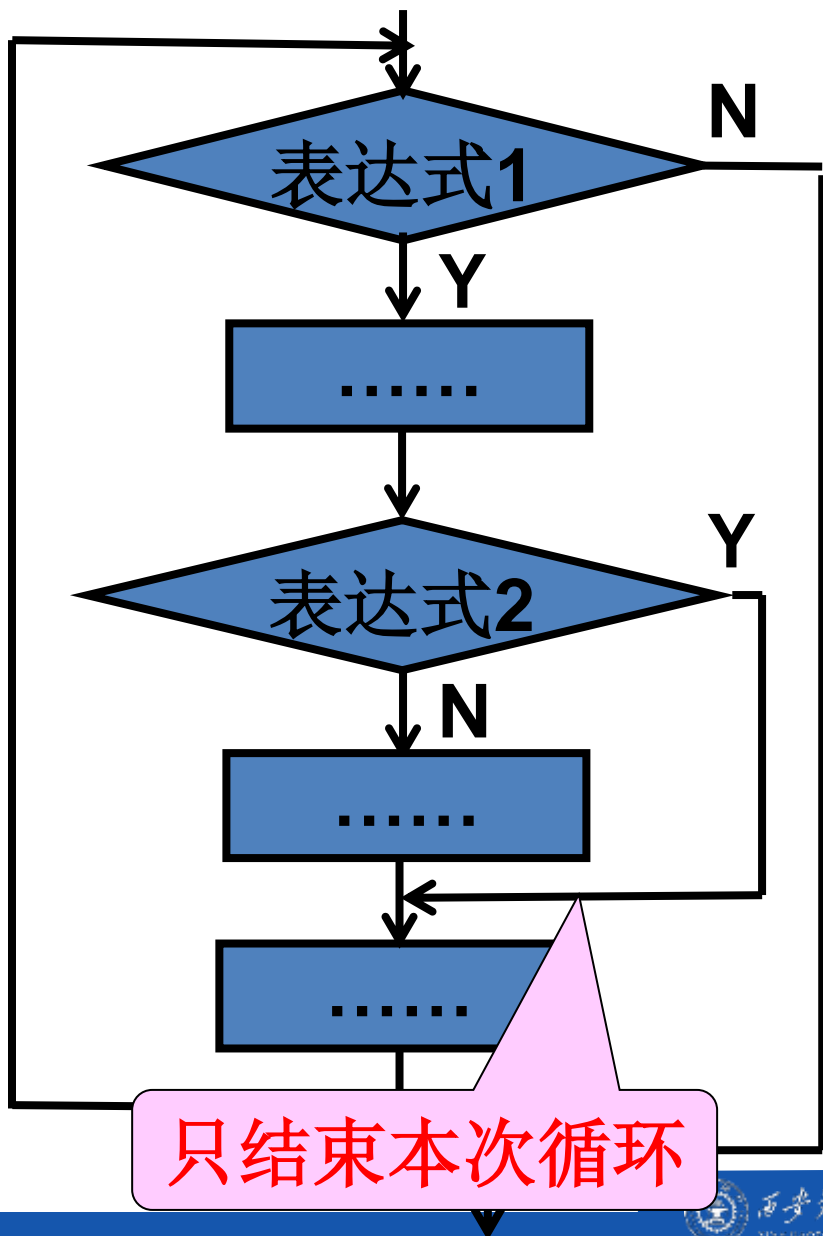
break语句和continue语句的区别

- continue语句只结束本次循环，而不是终止整个循环的执行
- break语句结束整个循环过程，不再判断执行循环的条件是否成立

break语句



continue语句



嵌套循环语句

循环语句可以嵌套来实现更复杂的情况。

```
nestedforstatement.cpp  x
HelloGeometricEngine  (全局范围)  main(int argv, char * args[])

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间               //Line 2
3                               //Line 3
4  //主函数                     //Line 4
5  int main(int argv, char* args[]) //Line 5
6  {                             //Line 6
7      int theStartPos;          //Line 7
8      //循环语句               //Line 8
9      for (int theRowNumber = 1; theRowNumber <= 20; theRowNumber += 2) //Line 9
10     {                          //Line 10
11         //当前行第一个*出现的位置 //Line 11
12         theStartPos = 11 - theRowNumber / 2; //Line 12
13         for (int theColumnNumber = 1; theColumnNumber <= theStartPos + theRowNumber; theColumnNumber++) //Line 13
14         {                      //Line 14
15             if (theColumnNumber <= theStartPos) //Line 15
16                 cout << " "; //Line 16
17             else //Line 17
18                 cout << "* "; //Line 18
19         } //Line 19
20         //换行 //Line 20
21         cout << endl; //Line 21
22     } //Line 22
23 //Line 23
24 // //Line 24
25     system("pause"); //等待用户按键 //Line 25
26     return 0; //Line 26
27 }
```

几种循环的比较

- (1) 一般情况下，3种循环可以互相代替
- (2) 在while和do---while循环中，循环体应包含使循环趋于结束的语句。
- (3) 用while和do---while循环时，循环变量初始化的操作应在while和do---while语句之前完成。而for语句可以在表达式1中实现循环变量的初始化。

循环例子

```
#include <iostream>
#define SUM 100000
int main()
{
    float amount, aver, total;    int i;
    for (i=1, total=0; i<=1000; i++)
    {
        cout << "please enter amount:";
        cin >> amount;
        total= total+amount;
        if (total>=SUM) break;
    }
    aver=total / i ;
    return aver;
}
```

循环例子

计算以下两个 4×5 的矩阵的和。

10	20	30	40	50
20	40	60	80	100
30	60	90	120	150
40	80	120	160	200

1	5	6	8	5
3	4	7	5	3
2	6	9	6	9
7	8	5	7	1

解题思路

- 解题思路：
 - 可以用循环的嵌套来处理此问题
 - 用外循环处理一行数据
 - 用内循环来处理一列数据
 - 按矩阵A和B的元素求和赋值到C对应的元素上。

循环例子

```
#include <vector>
int main()
{
    int i, j, n=0;
    int A[4][5] = {}, B, C;
    for (i=1; i<=4; i++)
        for (j=1; j<=5; j++, n++)
        {
            C[i][j] = A[i][j] + B[i][j];
        }
    cout << "\n";
    return 0;
}
```

循环例子

```
#include <vector>
int main()
{   int i, j, n=0;
    std::vector<std::vector<double>>> A, B, C;
    for (i=1; i<=4; i++)
        for (j=1; j<=5; j++, n++)
        {
            C[i][j]
        }
    cout << "\n";
    return 0;
}
```


循环例子

- 用 $\frac{\pi}{4} \approx 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$ 公式求 π 的近似值，直到发现某一项的绝对值小于 10^{-6} 为止(该项不累计加)。

循环例子

- 每项的分子都是1
- 后一项的分母是前一项的分母加2
- 第1项的符号为正，从第2项起，每一项的符号与前一项的符号相反

$$\frac{1}{n} \quad \rightarrow \quad -\frac{1}{n+2}$$

循环例子

sign=1, pi=0, n=1, term=1

当term $\geq 10^{-6}$

pi=pi+term

n=n+2

sign=-sign

term=sign/n

pi=pi*4

输出pi

循环例子

```
#include <iostream>
#include <math.h>
int main()
{   int sign=1; double pi=0, n=1, term=1;
    while(fabs(term)>=1e-6)
    {   pi=pi+term;
        n=n+2;
        sign=-sign;
        term=sign/n;
    }
    pi=pi*4;
    cout << pi;
    return 0;
}
```

循环例子

- 求费波那西 (Fibonacci) 数列的前40个数。这个数列有如下特点：第1、2两个数为1、1。从第3个数开始，该数是其前面两个数之和。

$$\begin{cases} F_1 = 1 & (n = 1) \\ F_2 = 1 & (n = 2) \\ F_n = F_{n-1} + F_{n-2} & (n \geq 3) \end{cases}$$

循环例子

- 输入一个大于3的整数 n ，判定它是否是素数 (prime, 又称质数)。
- 解题思路：
 - 让 n 被 i 整除 (i 的值从2变到 $n-1$)
 - 如果 n 能被 $2 \sim (n-1)$ 之中任何一个整数整除，则表示 n 肯定不是素数，不必再继续被后面的整数除，因此，可以提前结束循环
 - 注意：此时 i 的值必然小于 n

循环例子

- 求100~200间的全部素数。
- 解题思路：
- 增加一个外循环，先后对100~200间的全部整数一一进行判定即可

循环例子

- 编写一个程序，输出所有的两位正整数（10到99）之间，所有能够被5整除同时不被3整除的数，每输出6个数换一行。

浮点（重新考虑）

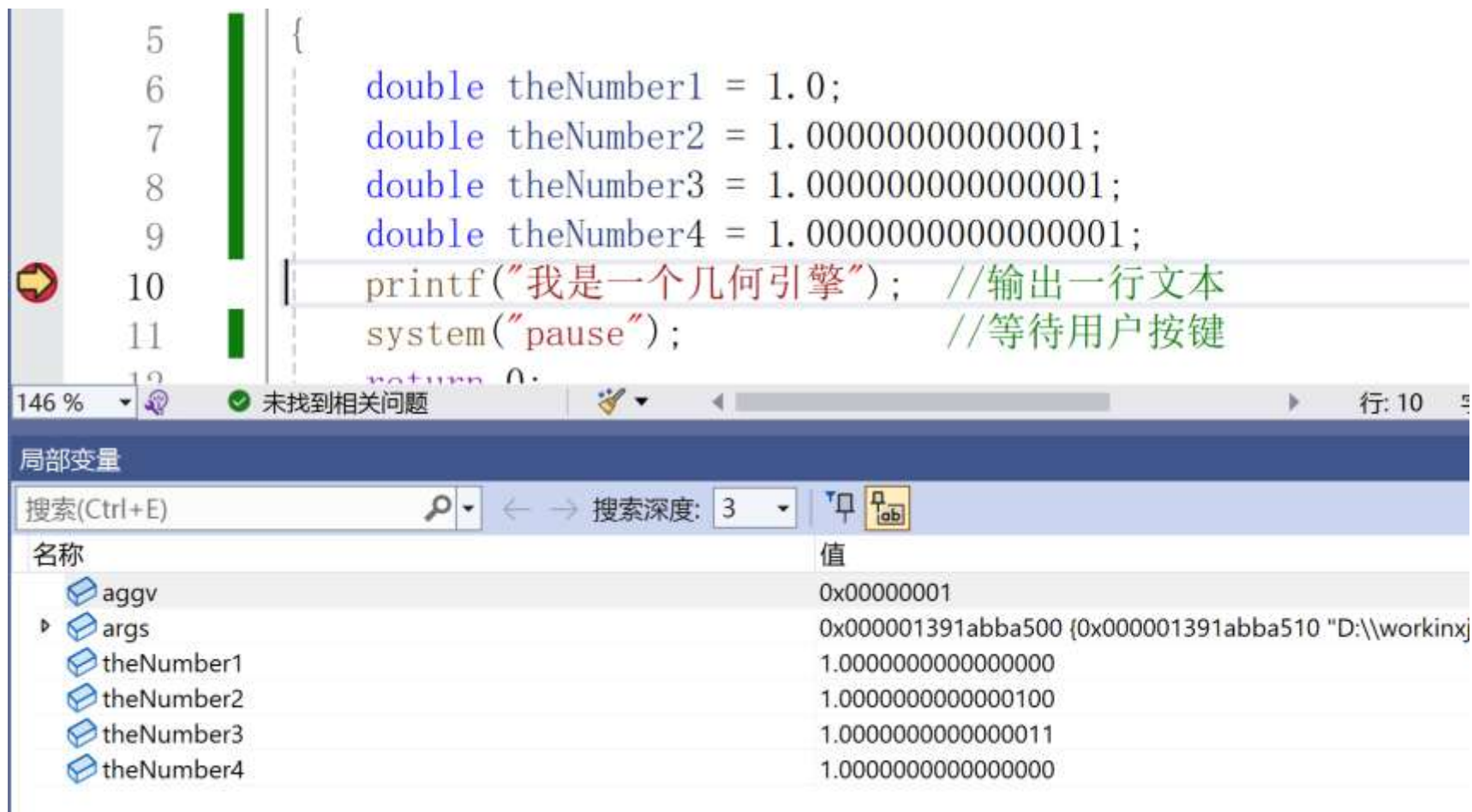
浮点型数据是用来表示具有小数点的实数，c语言中分为单精度、双精度和长双精度浮点数。浮点类型使用 IEEE-754 表示形式在各种数量级上提供小数值的近似值。

64位双精度表示的10进制的有效位数是16位，小数点后15位，最大浮点表示精度 10^{-15} 。



类 型	符号符	阶码	尾数码	总位数	偏置值	
					十六进制	十进制
短浮点数	1	8	23	32	7FH	127
长浮点数	1	11	52	64	3FFH	1023
临时短浮点数	1	15	64	80	3FFFH	16383

浮点数



The screenshot shows a C++ program in a code editor. The program defines four double variables: theNumber1, theNumber2, theNumber3, and theNumber4. It then prints a message and pauses execution. The variable watch window at the bottom shows the values of these variables.

```

5  {
6      double theNumber1 = 1.0;
7      double theNumber2 = 1.0000000000000001;
8      double theNumber3 = 1.0000000000000001;
9      double theNumber4 = 1.0000000000000001;
10     printf("我是一个几何引擎"); //输出一行文本
11     system("pause"); //等待用户按键
12     return 0;

```

局部变量

搜索(Ctrl+E) 搜索深度: 3

名称	值
aggv	0x00000001
args	0x000001391abba500 {0x000001391abba510 "D:\\workinxj
theNumber1	1.0000000000000000
theNumber2	1.0000000000000001
theNumber3	1.0000000000000001
theNumber4	1.0000000000000000

使用统一的容差是保证造型系统一致性的一个重要因素

1. `logical is_equal(double n1, double n2);`
2. `logical is_equal(const SPAposition& p1, const SPAposition& p2);`
3. `logical is_equal(const SPAvector& v1, const SPAvector& v2);`
4. `logical is_equal(const SPAunit_vector& v1, const SPAunit_vector& v2);`
5. `logical are_parallel(const SPAunit_vector& v1, const SPAunit_vector& v2...);`
6. `logical are_parallel(const SPAvector& v1, const SPAvector& v2...);`
7. `logical are_perpendicular(const SPAunit_vector& v1, const SPAunit_vector& v2);`
8. `logical are_perpendicular(const SPAvector& v1, const SPAvector& v1);`
9. `logical is_zero(double n);`
10. `logical is_zero(const SPAposition& p);`
11. `logical is_zero(const SPAvector& v);`
12. `logical is_zero(const SPAunit_vector& p);`
13. `logical is_zero_squared(double n);`
14. `logical is_zero_mch(double n);`
15. `logical is_zero_nor(double n);`
16. `logical is_positive(double n);`
17. `logical is_negative(double n);`
18. `logical is_less_than(double n1, double n2);`
19. `logical is_greater_than(double n1, double n2);`
20. `double get_resabs();` (读音 "rez-abs")
21. `double get_resnor();` (读音 "rez-nor")
22. `double get_resfit();`
23. `double get_resmch();`
24. `const double SParesabs = 1E-6;` (读音 "rez-abs")
25. `const double SParesnor = 1E-10;` (读音 "rez-nor")
26. `const double SParesfit = 1E-3;`
27. `const double SParesmch = 1E-11;`

浮点（重新考虑）

a/b做除法前一定要判断除数 - 防止浮点溢出，可以使用
`is_valid_divisor()`判断

数组

- 前面使用的变量都属于基本类型，例如整型、字符型、浮点型数据，这些都是简单的数据类型。对于有些数据，只用简单的数据类型是不够的，难以反映出数据的特点，也难以有效地进行处理。
- 数组是相同类型的对象序列，它们占据一块连续的内存区。数组是一组有序数据的集合。
- 数组中各数据的排列是有一定规律的，下标代表数据在数组中的序号，用一个数组名和下标惟一确定数组中的元素，数组中的每一个元素都属于同一个数据类型。

数组

- 元素数量必须以整数文本或常量表达式的形式提供。这是因为，编译器必须知道要分配多少堆栈空间；它不能使用在运行时计算的值。
- `int Array[1000];`

数组

- 如果有1000名学生，每个学生有一个成绩，需要求这1000名学生的平均成绩。
- 用 $s_1, s_2, s_3, \dots, s_{1000}$ 表示每个学生的成绩，能体现内在联系。
- C语言用方括号中的数字表示下标，如用 $s[15]$ 表示。

数组

- 数组是一组有序数据的集合。数组中各数据的排列是有一定规律的，下标代表数据在数组中的序号
- 用一个数组名和下标惟一确定数组中的元素
- 数组中的每一个元素都属于同一个数据类型

怎样定义和引用一维数组

- 怎样定义一维数组
- 怎样引用一维数组元素
- 一维数组的初始化
- 一维数组程序举例

怎样定义一维数组

- 一维数组是数组中最简单的
- 它的元素只需要用数组名加一个下标，就能惟一确定
- 要使用数组，必须在程序中先定义数组

怎样定义一维数组

- 定义一维数组的一般形式为：
类型符 数组名[常量表达式];
- 数组名的命名规则和变量名相同
如 `int a[10];`

怎样定义一维数组

- 定义一维数组的一般形式为：

类型符 数组名[常量表达式]；

- 数组名的命名规则和变量名相同

如 `int a[10];`

10个元素：a[0], a[1], a[2], ..., a[9]

a[0]

a[1]

a[2]

a[3]

...

a[7]

a[8]

a[9]

怎样定义一维数组

- 定义一维数组的一般形式为：

类型符 数组名[常量表达式];

int a[4+6]; 合法

int n=10;

int a[n];

怎样定义一维数组

- 在定义数组并对其中各元素赋值后，就可以引用数组中的元素
- 注意：只能引用数组元素而不能一次整体调用整个数组全部元素的值

怎样引用一维数组元素

- 引用数组元素的表示形式为：

数组名 [下标]

如 $a[0]=a[5]+a[7]-a[2*3]$ 合法

```
int n=5, a[10];
```

```
a[n]=20;
```

怎样引用一维数组元素

- 对10个数组元素依次赋值为0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 要求按逆序输出。

解题思路：

- 定义一个长度为10的数组，数组定义为整型
- 要赋的值是从0到9，可以用循环来赋值
- 用循环按下标从大到小输出这10个元素

怎样引用一维数组元素

```
#include <isostream>
```

```
int main()
```

```
{  int i, a[10];
```

```
    for (i=0; i<=9; i++)
```

```
        a[i]=i;
```

```
    for (i=9; i>=0; i--)
```

```
        cout << a[i];
```

```
    cout << "\n";
```

```
    return 0;
```

```
}
```

使a[0]~a[9]
的值为0~9

0

1

2

3

4

5

6

7

8

9

怎样引用一维数组元素

```
#include <isostream>

int main()
{   int i, a[10];
    for (i=0; i<=9; i++)
        a[i]=i;
    for (i=9; i>=0; i--)
        cout << a[i];
    cout << "\n";
    return 0;
}
```

先输出a[9],
最后输出a[0]

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

一维数组的初始化

- 在定义数组的同时，给各数组元素赋值
- `int a[10]={0, 1, 2, 3, 4, 5, 6, 7, 8, 9};`
- `int a[10]={0, 1, 2, 3, 4};` 相当于
`int a[10]={0, 1, 2, 3, 4, 0, 0, 0, 0, 0};`
- `int a[10]={0, 0, 0, 0, 0, 0, 0, 0, 0, 0};` 相当于
`int a[10]={0};`
- `int a[5]={1, 2, 3, 4, 5};` 可写为
`int a[]={1, 2, 3, 4, 5};`

一维数组程序举例

- 用数组处理求Fibonacci数列问题
- 解题思路：
- 用简单变量处理的，缺点不能在内存中保存这些数。假如想直接输出数列中第25个数，是很困难的。
- 如果用数组处理，每一个数组元素代表数列中的一个数，依次求出各数并存放在相应的数组元素中

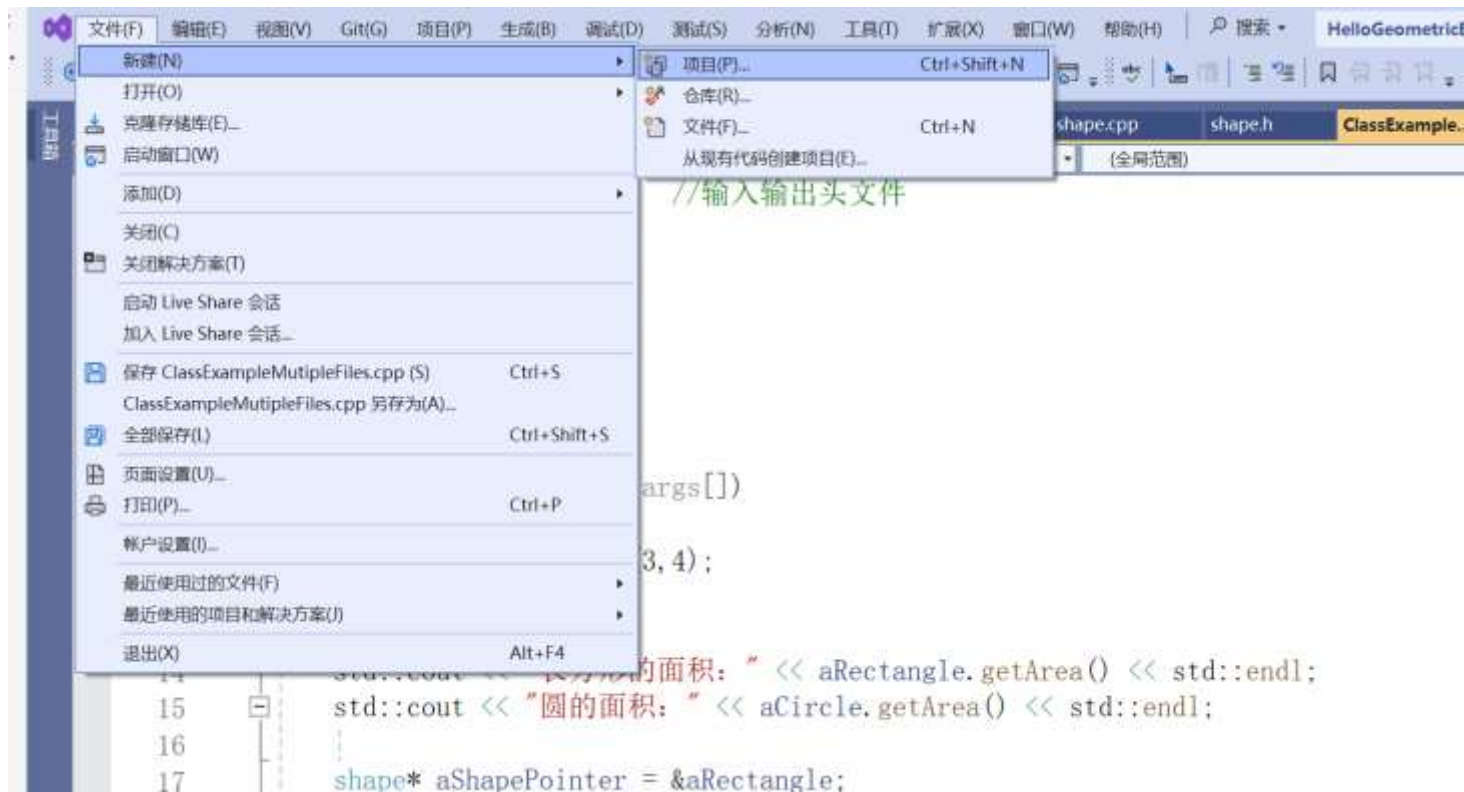
一维数组程序举例

```
#include <isostream>

Int main()
{   int i;   int f[20]={1, 1};
    for (i=2; i<20; i++)
        f[i]=f[i-2]+f[i-1];
    for (i=0; i<20; i++)
    {   if (i%5==0) cout << "\n" ;
        cout << f[i];
    }
}
```

多边形

- 几何引擎中应用非常多的一个场景，多边形处理。
- 在工业软件中，很多形状都用多边形来进行表达，比如线路板、零件草图，图形表示。



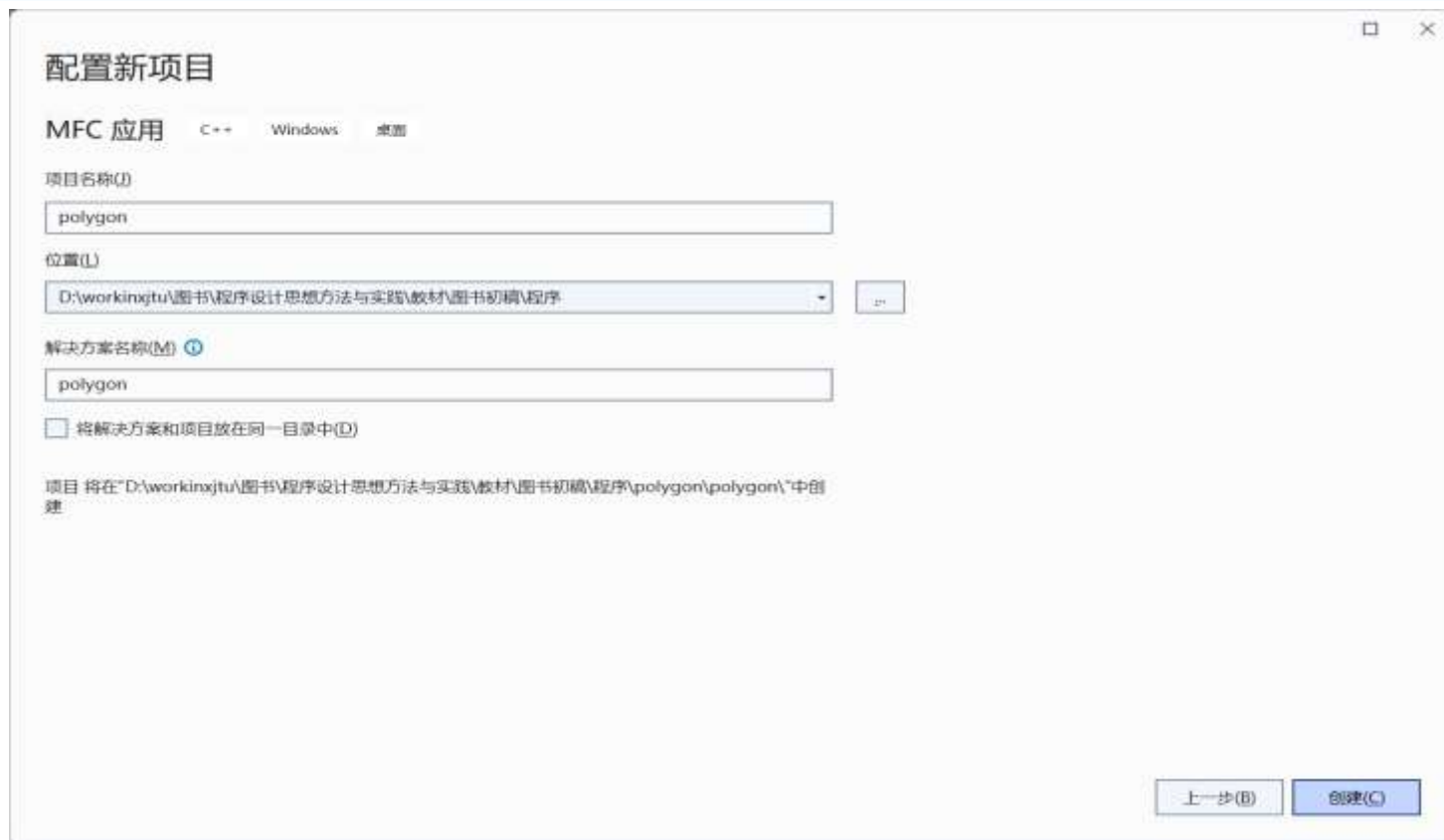
多边形

- 几何引擎中应用非常多的一个场景，多边形处理。
- 在工业软件中，很多形状都用多边形来进行表达，比如线路板、零件草图，图形表示。



多边形

- 几何引擎中应用非常多的一个场景，多边形处理。
- 在工业软件中，很多形状都用多边形来进行表达，比如线路板、零件草图，图形表示。



多边形

- 几何引擎中应用非常多的一个场景，多边形处理。
- 在工业软件中，很多形状都用多边形来进行表达，比如线路板、零件草图，图形表示。



多边形

- 几何引擎中应用非常多的一个场景，多边形处理。
- 在工业软件中，很多形状都用多边形来进行表达，比如线路板、零件草图，图形表示。



多边形

```
新增功能    polygonView.cpp  X
polygon
55
56  void CpolygonView::OnDraw(CDC* /*pDC*/)
57  {
58      CpolygonDoc* pDoc = GetDocument();
59      ASSERT_VALID(pDoc);
60      if (!pDoc)
61          return;
62
63      // TODO: 在此处为本机数据添加绘制代码
64      CClientDC dc(this);
65      dc.MoveTo(10, 10);
66      dc.LineTo(100, 100);
67      dc.LineTo(200, 100);
68      dc.LineTo(300, 200);
69      dc.LineTo(400, 10);
70      dc.LineTo(10, 10);
71  }
```

多边形



多边形

- 用数组进行重写绘制代码。

```
功能 polygonView.cpp X
olygon ↓ CpolygonView
71 CPoint anPointArray[5] = { {10,10}, {100,100}, {200,100}, {300,200}, {400,10} };
72
73 CClientDC dc(this);
74 dc.MoveTo(anPointArray[0]);
75 for (int i = 1; i < 5; i++)
76 {
77     dc.LineTo(anPointArray[i]);
78 }
79 dc.LineTo(anPointArray[0]);
80 }
```

多边形

- 利用标准库进行重写绘制代码。

polygonView.cpp

↓ CpolygonView

```
std::vector<CPoint> anPointArray = {{10, 10}, {100, 100}, {200, 100}, {300, 200}, {400, 10}};  
CClientDC dc(this);  
dc.MoveTo(anPointArray[0]);  
for (int i = 1; i < 5; i++)  
{  
    dc.LineTo(anPointArray[i]);  
}  
dc.LineTo(anPointArray[0]);
```

数组优缺点

- 传统的 C 样式数组是许多 bug 的根源，但至今仍很常用，尤其是在较旧的代码库中。 在新式 C++ 中，我们强烈建议使用 `std::vector` 或 `std::array`，而不是本部分所述的 C 样式数组。
- 这两种标准库类型都将其元素存储为连续的内存块。 但是，它们提供更高的类型安全性，并支持保证指向序列中有效位置的迭代器。。

数组排序

- 有10个地区的面积，要求对它们按由小到大的顺序排列
- 解题思路：
 - 排序的规律有两种：一种是“升序”，从小到大；另一种是“降序”，从大到小
 - 把题目抽象为：“对 n 个数按升序排序”
 - 采用冒泡法排序
- 为什么要对数组进行排序？有什么好处？
- 从数组中查找一个给定的值，怎么查找，其时间复杂度是什么？
- 最快的排序算法是什么？其时间复杂度是什么？

数组

```
for(i=0;i<5;i++)  
    if (a[i]>a[i+1])  
    { t=a[i];a[i]=a[i+1];a[i+1]=t; }
```

a[0]	9	8	8	8	8	8
a[1]	8	9	5	5	5	5
a[2]	5	5	9	4	4	4
a[3]	4	4	4	9	2	2
a[4]	2	2	2	2	9	0
a[5]	0	0	0	0	0	9

大数沉淀，小数起泡

```
for(i=0;i<4;i++)  
    if (a[i]>a[i+1])  
        { t=a[i];a[i]=a[i+1];a[i+1]=t; }
```

a[0]	8	5	5	5	5
a[1]	5	8	4	4	4
a[2]	4	4	8	2	2
a[3]	2	2	2	8	0
a[4]	0	0	0	0	8
a[5]	9	9	9	9	9

```
for(i=0;i<3;i++)  
    if (a[i]>a[i+1])  
    { t=a[i];a[i]=a[i+1];a[i+1]=t; }
```

a[0]	5	4	4	4
a[1]	4	5	2	2
a[2]	2	2	5	0
a[3]	0	0	0	5
a[4]	8	8	8	8
a[5]	9	9	9	9

```
for(i=0;i<2;i++)  
    if (a[i]>a[i+1])  
        { t=a[i];a[i]=a[i+1];a[i+1]=t; }
```

a[0]	4	2	2
a[1]	2	4	0
a[2]	0	0	4
a[3]	5	5	5
a[4]	8	8	8
a[5]	9	9	9

```
for(i=0;i<1;i++)  
    if (a[i]>a[i+1])  
    { t=a[i];a[i]=a[i+1];a[i+1]=t; }
```

a[0]

a[1]

a[2]

a[3]

a[4]

a[5]

2

0

0

2

4

4

5

5

8

8

9

9

```
for(i=0;i<5;i++)  
    if (a[i]>a[i+1])  
    { .....
```

```
for(i=0;i<4;i++)  
    if (a[i]>a[i+1])  
    { .....
```

.....

```
for(i=0;i<1;i++)  
    if (a[i]>a[i+1])  
    { .....
```

```
for(j=0;j<5;j++)  
    for(i=0;i<5-j;i++)  
        if (a[i]>a[i+1])  
        { .....
```

```
input 10 numbers :  
34 67 90 43 124 87 65 99 132 26
```

```
int a[10];  
printf("the sorted numbers :  
26 34 43 65 67 87 90 99 124 132  
for (i=0;i<10;i++)  
printf("\n");  
for(j=0;j<9;j++)  
    for(i=0;i<9-j;i++)  
        if (a[i]>a[i+1])  
            {t=a[i];a[i]=a[i+1];a[i+1]=t;}  
printf("the sorted numbers :\n");  
for(i=0;i<10;i++) printf("%d ",a[i]);  
printf("\n");
```

查找元素

- 一个排序的数组，从中找到给定的数值
- 解题思路：
 - 顺序查找法：遍历所有的元素，直到找到给定的元素
 - 二分查找法：首先和中间的元素进行比较，根据比较结果，舍弃一半的元素，然后依次类推，直到找到对应的元素。
 - 这两种方法有什么不同？

怎样定义和引用二维数组

队员1 队员2 队员3 队员4 队员5 队员6

1分队	2456	1847	1243	1600	2346	2757
2分队	3045	2018	1725	2020	2458	1436
3分队	1427	1175	1046	1976	1477	2018

```
float pay[3][6];
```

怎样定义和引用二维数组

- 怎样定义二维数组
- 怎样引用二维数组的元素
- 二维数组的初始化
- 二维数组程序举例

怎样定义二维数组

- 二维数组定义的一般形式为
类型符 数组名[常量表达式][常量表达式];
如: `float a[3][4], b[5][10];`
- 二维数组可被看作是一种特殊的一维数组:
它的元素又是一个一维数组
- 例如, 把a看作是一个一维数组, 它有3个元素:
`a[0]`、`a[1]`、`a[2]`
- 每个元素又是一个包含4个元素的一维数组

怎样定义二维数组

a[0]

a[0][0]

a[0][1]

a[0][2]

a[0][3]

a[1]

a[1][0]

a[1][1]

a[1][2]

a[1][3]

a[2]

a[2][0]

a[2][1]

a[2][2]

a[2][3]

怎样定义二维数组

逻辑存储

$a[0][0]$	$a[0][1]$	$a[0][2]$	$a[0][3]$
$a[1][0]$	$a[1][1]$	$a[1][2]$	$a[1][3]$
$a[2][0]$	$a[2][1]$	$a[2][2]$	$a[2][3]$

内存中的存储顺序

怎样引用二维数组的元素

- 二维数组元素的表示形式为：

数组名 [下标] [下标]

- `b[1][2]=a[2][3]/2` 合法

- `for (i=0; i<m; i++)`

`printf(“%d, %d\n”, a[i][0], a[0][i]);` 合法

二维数组的初始化

```
int a[3][4]={ {1, 2, 3, 4}, {5, 6, 7, 8},  
              {9, 10, 11, 12} } ;
```

```
int a[3][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} ;
```

```
int a[3][4]={ {1}, {5}, {9} } ; 等价于
```

```
int a[3][4]={ {1, 0, 0, 0}, {5, 0, 0, 0},  
              {9, 0, 0, 0} } ;
```

```
int a[3][4]={ {1}, {5, 6} } ; 相当于
```

```
int a[3][4]={ {1}, {5, 6}, {0} } ;
```

二维数组的初始化

```
int a[3][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```

等价于：

```
int a[ ][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```

```
int a[][4]={ {0, 0, 3}, { }, {0, 10} };合法
```


二维数组程序举例

将一个二维数组行和列的元素互换，存到另一个二维数组中。

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \longrightarrow b = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

二维数组程序举例

- 解题思路：
 - 可以定义两个数组：数组a为2行3列，存放指定的6个数
 - 数组b为3行2列，开始时未赋值
 - 将a数组中的元素 $a[i][j]$ 存放到b数组中的 $b[j][i]$ 元素中
 - 用嵌套的for循环完成

二维数组程序举例

```
#include <iostream>

int main()
{   int a[2][3]={ {1, 2, 3}, {4, 5, 6} };
    int b[3][2], i, j;
    cout << "array a:\n";
    for (i=0; i<=1; i++)
    {   for (j=0; j<=2; j++)
        {   cout << a[i][j];
            b[j][i]=a[i][j];
        }
        cout << "\n";
    }
}
```

二维数组程序举例

```
cout << "array b:\n";  
for (i=0; i<=2; i++)  
{   for (j=0; j<=1; j++)  
        cout << b[i][j];  
        cout << "\n";  
}  
return 0;  
}
```

二维数组程序举例

有一个 3×4 的矩阵，要求编程序求出其中值最大的那个元素的值，以及其所在的行号和列号。

解题思路：采用“打擂台算法”

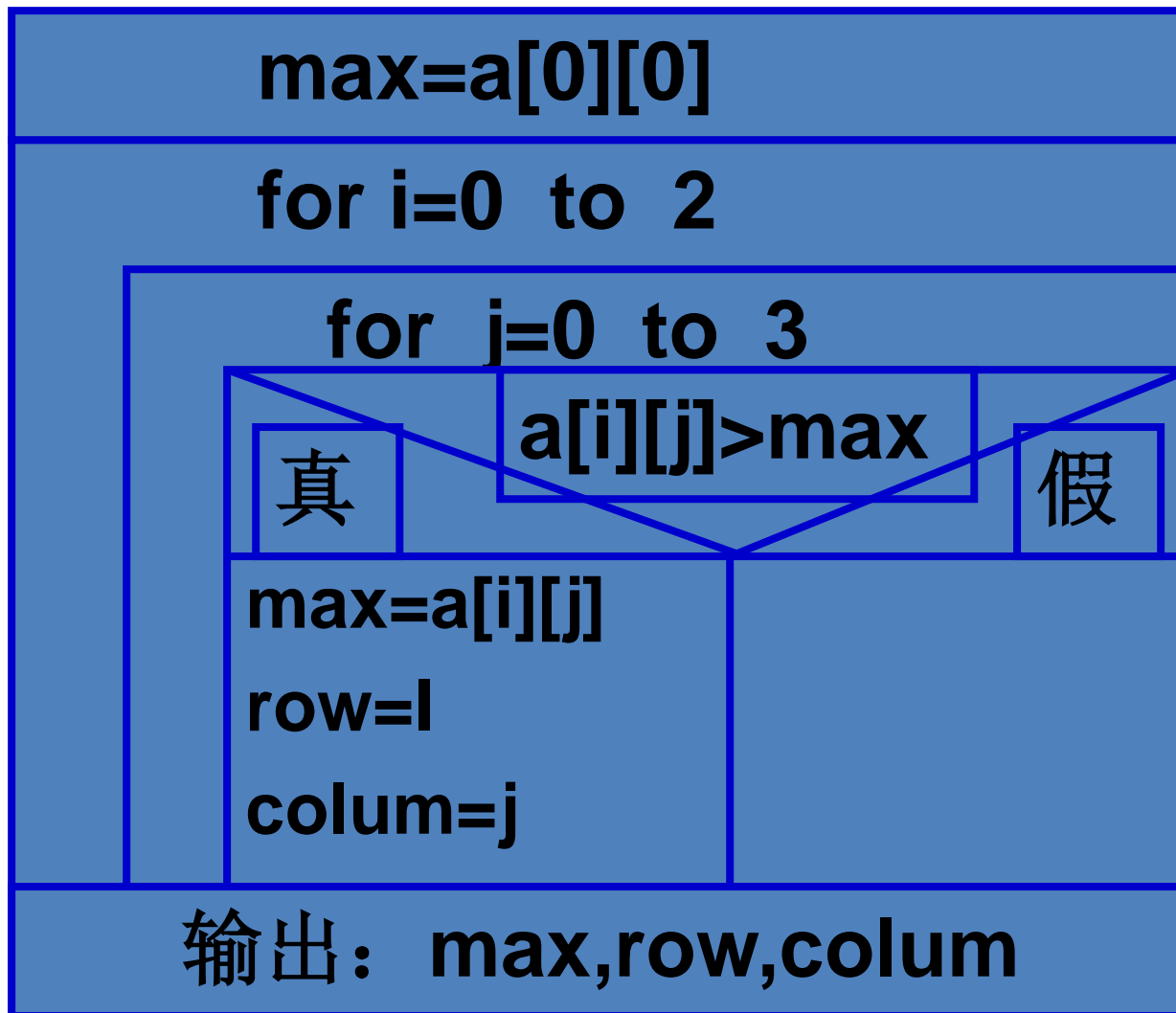
- 先找出任一人站在台上，第2人上去与之比武，胜者留在台上
- 第3人与台上的人比武，胜者留台上，败者下台
- 以后每一个人都是与当时留在台上的人比武，直到所有人都上台比为止，最后留在台上的是冠军

二维数组程序举例

有一个 3×4 的矩阵，要求编程序求出其中值最大的那个元素的值，以及其所在的行号和列号。

- 解题思路：采用“打擂台算法”
 - 先把 $a[0][0]$ 的值赋给变量max
 - max用来存放当前已知的最大值
 - $a[0][1]$ 与max比较，如果 $a[0][1] > \text{max}$ ，则表示 $a[0][1]$ 是已经比过的数据中值最大的，把它的值赋给max，取代了max的原值
 - 以后依此处理，最后max就是最大的值

二维数组程序举例



二维数组程序举例

.....

```
int i,j,row=0,column=0,max;
int a[3][4]={{1,2,3,4},{9,8,7,6},
             {-10,10,-5,2}};
max=a[0][0];
for (i=0;i<=2;i++)
    for (j=0;j<=3;j++)
        if (a[i][j]>max)
            { max=a[i][j]; row=i; column=j; }
printf("max=%d\nrow=%d\n
       column=%d\n",max,row,column);
```

.....

字符数组

怎样定义字符数组

字符数组的初始化

怎样引用字符数组中的元素

字符串和字符串结束标志

怎样定义字符数组

用来存放字符数据的数组是字符数组

字符数组中的一个元素存放一个字符

定义字符数组的方法与定义数值型数组的方法类似

怎样定义字符数组

```
char c[10];
```

```
c[0]=' l' ;    c[1]='  ' ;
```

```
c[2]=' a' ;    c[3]=' m' ;
```

```
c[4]='  ' ;    c[5]=' h' ;
```

```
c[6]=' a' ;    c[7]=' p' ;
```

```
c[8]=' p' ;    c[9]=' y' ;
```

c[0]c[1]c[2]c[3]c[4]c[5]c[6]c[7]c[8]c[9]

l		a	m		h	a	p	p	y
----------	--	----------	----------	--	----------	----------	----------	----------	----------

字符数组的初始化

```
char c[10]={ 'l', ' ', 'a', 'm', ' ', 'h', 'a', 'p', 'p', 'y' };
```

```
char c[10]={ 'c', ' ', 'p', 'r', 'o', 'g', 'r', 'a', 'm' };
```

c[0]c[1]c[2]c[3]c[4]c[5]c[6]c[7]c[8]c[9]

l		a	m		h	a	p	p	y
---	--	---	---	--	---	---	---	---	---

c[0]c[1]c[2]c[3]c[4]c[5]c[6]c[7]c[8]c[9]

c		p	r	o	g	r	a	m	\0
---	--	---	---	---	---	---	---	---	----

字符数组的初始化

```
char diamond[5][5]={{' ',' ',' ','*'},  
                    {' ','*',' ',' ','*'},  
                    {'*',' ',' ',' ','*'},  
                    {' ','*',' ',' ','*'},  
                    {' ',' ',' ','*'} };
```

怎样引用字符数组中的元素

输出一个已知的字符串。

解题思路：

定义一个字符数组，并用“初始化列表”对其赋以初值
用循环逐个输出此字符数组中的字符

怎样引用字符数组中的元素

```
#include <iostream>

int main()

{ char c[15]={ 'I' , ' ' , 'a' , 'm' , ' ' , 'a' ,
               's', 't', 'u', 'd', 'e', 'n', 't', '.' };

  int i;

  for (i=0; i<15; i++)

    cout << c[i];

  cout << "\n";

  return 0;

}
```

怎样引用字符数组中的元素

输出一个菱形图。

解题思路：

定义一个字符型的二维数组，用“初始化列表”进行初始化
用嵌套的for循环输出字符数组中的所有元素。

怎样引用字符数组中的元素

```
#include <iostream>

int main()
{ char diamond[][5]={{ ' ', ' ', '*' },
                      { ' ', '*', ' ', '*' }, { '*', ' ', ' ', ' ', '*' },
                      { ' ', '*', ' ', '*' }, { ' ', ' ', '*' }};

  int i, j;
  for (i=0; i<5; i++)
    {for (j=0; j<5; j++)
      cout << diamond[i][j];
      cout << "\n";
    }
  return 0;
}
```

字符串和字符串结束标志

在C语言中，是将字符串作为字符数组来处理的

关心的是字符串的有效长度而不是字符数组的长度

为了测定字符串的实际长度，C语言规定了字符串结束标志

' \0'

字符串和字符串结束标志

' \0' 代表ASCII码为0的字符

从ASCII码表可以查到，ASCII码为0的字符不是一个可以显示的字符，而是一个“空操作符”，即它什么也不做

用它作为字符串结束标志不会产生附加的操作或增加有效字符，只起一个供辨别的标志

字符串和字符串结束标志

```
char c[]={ " I am happy" };
```

可写成

```
char c[]=" I am happy" ;
```

相当于

```
char c[11]={ " I am happy" };
```

字符串和字符串结束标志

```
char c[10]={ "China" };
```

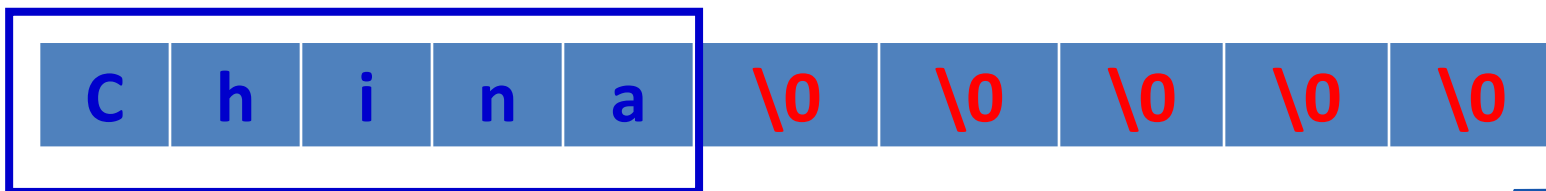
可写成

```
char c[10]= "China" ;
```

从c[5]开始，元素值均为\0

只显示

```
cout << c;
```



为什么要用函数

- 如果程序的功能比较多，规模比较大，把所有代码都写在main函数中，就会使主函数变得庞杂、头绪不清，阅读和维护变得困难
- 有时程序中要多次实现某一功能，就需要多次重复编写实现此功能的程序代码，这使程序冗长，不精炼

为什么要用函数

- 解决的方法：用模块化程序设计的思路
 - 采用“组装”的办法简化程序设计的过程
 - 事先编好一批实现各种不同功能的函数
 - 把它们保存在函数库中，需要时直接用

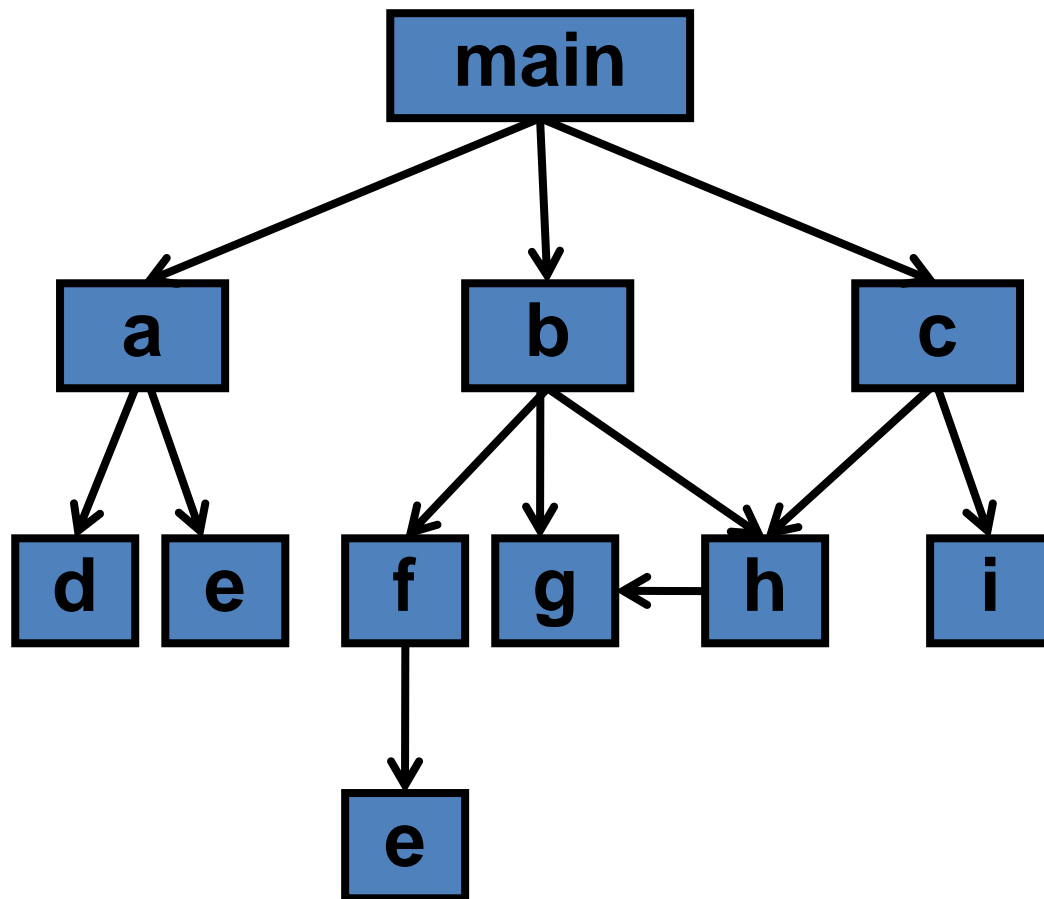
为什么要用函数

- 解决的方法：用模块化程序设计的思路
 - 函数就是功能
 - 每一个函数用来实现一个特定的功能
 - 函数的名字应反映其代表的功能

为什么要用函数

- 在设计一个较大的程序时，往往把它分为若干个程序模块，每一个模块包括一个或多个函数，每个函数实现一个特定的功能
- C 程序可由一个主函数和若干个其他函数构成
- 主函数调用其他函数，其他函数也可以互相调用
- 同一个函数可以被一个或多个函数调用任意多次

为什么要用函数



为什么要用函数

- 可以使用库函数
- 可以使用自己编写的函数
- 在程序设计中要善于利用函数，可以减少重复编写程序段的工作量，同时可以方便地实现模块化的程序设计

函数

- 函数是执行某种操作的代码块。 函数可以选择性地定义使调用方可以将实参传递到函数中的输入形参。
- 函数可以选择性地返回值作为输出。
- 函数可用于在单个可重用块中封装常用操作。
- 对于函数长度没有实际限制，不过良好的设计应以执行单个明确定义的任务的函数为目标。
- 复杂算法应尽可能分解成易于理解的更简单函数。模块化设计基本单元是函数，通过将复杂任务分解为意义明确的子单元，通过调用子单元对应的函数最终实现复杂任务。

定义函数

- C语言要求，在程序中用到的所有函数，必须“先定义，后使用”
- 指定函数名字、函数返回值类型、函数实现的功能以及参数的个数与类型，将这些信息通知编译系统。

定义函数

- 指定函数的名字，以便以后按名调用
- 指定函数类型，即函数返回值的类型
- 指定函数参数的名字和类型，以便在调用函数时向它们传递数据
- 指定函数的功能。这是最重要的，这是在函数体中解决的

定义函数

- 对于库函数，程序设计者只需用`#include`指令把有关的头文件包含到本文件模块中即可
- 程序设计者需要在程序中自己定义想用的而库函数并没有提供的函数

定义函数

定义无参函数

- 定义无参函数的一般形式为：

类型名 函数名()

{

函数体

}

类型名 函数名(void)

{

函数体

}

包括声明部分和
语句部分

定义函数

定义无参函数

- 定义无参函数

指定函数值的类型

类型名 函数名()

{

函数体

}

类型名 函数名(void)

{

函数体

}

定义函数

定义有参函数

定义有参函数的一般形式为：

类型名 函数名（形式参数表列）

{

函数体

}

定义函数

定义空函数

定义空函数的一般形式为：

类型名 函数名 ()

{
}

先用空函数占一个位置，以后逐步扩充

好处：程序结构清楚，可读性好，以后扩充新功能方便，对程序结构影响不大

函数调用的形式

函数调用的一般形式为：

函数名（实参表列）

如果是调用无参函数，则“实参表列”可以没有，但括号不能省略

如果实参表列包含多个实参，则各参数间用逗号隔开

函数

- 程序中只能有一个main函数，称为程序的主入口。
- 可以从程序中任意位置调用函数。
- 传递给函数的值是实参，其类型必须与函数定义中的形式参数类型兼容。

```
SumFunction.cpp  + x
HelloGeometricEngine  (全局范围)
1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间             //Line 2
3  //定义求和函数               //Line 3
4  int MyIntSum(const int& theIntValueA, const int& theIntValueB) //Line 4
5  {                             //Line 5
6      return theIntValueA + theIntValueB;               //Line 6
7  }                             //Line 7
8  //主函数                     //Line 8
9  int main(int argv, char* args[]) //Line 9
10 {                             //Line 10
11     int theIntValue = MyIntSum(10, 20);                //Line 11
12     cout << "10和20的和为: " << theIntValue << endl; //Line 12
13     system("pause");                                     //Line 13
14     return 0;                                           //Line 14
15 }
```

函数

- 程序中只能有一个main函数，称为程序的主入口。
- 可以从程序中任意位置调用函数。
- 传递给函数的值是实参，其类型必须与函数定义中的形式参数类型兼容。

```
SumFunction.cpp  + x
HelloGeometricEngine  (全局范围)
1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间             //Line 2
3  //定义求和函数               //Line 3
4  int MyIntSum(const int& theIntValueA, const int& theIntValueB) //Line 4
5  {                             //Line 5
6      return theIntValueA + theIntValueB;               //Line 6
7  }                             //Line 7
8  //主函数                     //Line 8
9  int main(int argv, char* args[]) //Line 9
10 {                             //Line 10
11     int theIntValue = MyIntSum(10, 20);                 //Line 11
12     cout << "10和20的和为: " << theIntValue << endl;   //Line 12
13     system("pause");                                     //Line 13
14     return 0;                                           //Line 14
15 }
```

函数

- `const` &形参, 表示在函数内部不能修改此变量。
- &是引用标志, 表示形式参数和实参共享同一段地址空间。
- 这是在软件工程中非常常用的一种技巧, 即避免了修改实参的风险, 又防止了实参和形参之间的值拷贝 (占用内存, 影响效率)

```
SumFunction.cpp  HelloGeometricEngine  {全局范围}
1  #include <iostream>                //输入输出头文件                //Line 1
2  using namespace std;                //std命名空间                //Line 2
3  //定义求和函数                    //Line 3
4  int MyIntSum(const int& theIntValueA, const int& theIntValueB) //Line 4
5  {                                    //Line 5
6      return theIntValueA + theIntValueB; //Line 6
7  }                                    //Line 7
8  //主函数                            //Line 8
9  int main(int argv, char* args[])    //Line 9
10 {                                    //Line 10
11     int theIntValue = MyIntSum(10, 20); //Line 11
12     cout << "10和20的和为: " << theIntValue << endl; //Line 12
13     system("pause");                  //等待用户按键                //Line 13
14     return 0;                        //Line 14
15 }
```

函数调用时的数据传递

- 形式参数和实际参数
 - 在调用有参函数时，主调函数和被调用函数之间有数据传递关系
 - 定义函数时函数名后面的变量名称为“形式参数”
 - 主调函数中调用一个函数时，函数名后面参数称为“实际参数”（简称“实参”）
- 实际参数可以是常量、变量或表达式

函数调用时的数据传递

- 实参和形参间的数据传递
 - 在调用函数过程中，系统会把实参的值传递给被调用函数的形参
 - 或者说，形参从实参拷贝一个值，一般情况下改变形参，而实参的值不变
 - 该值在函数调用期间有效，可以参加被调函数中的运算

函数调用时的数据传递

- 输入两个整数，要求输出其中值较大者。要求用函数来找到大数。
 - 解题思路：
 - (1) 函数名应是见名知意，今定名为max
 - (2) 由于给定的两个数是整数，返回主调函数的值（即较大数）应该是整型
 - (3) max函数应当有两个参数，以便从主函数接收两个整数，因此参数的类型应当是整型

函数调用时的数据传递

先编写max函数:

```
int max(int x,int y)
{
    int z;
    z=x>y?x:y;
    return(z);
}
```

函数调用时的数据传递

- 在max函数上面，再编写主函数
- `#include <iostream>`
- `Int main()`
- `{ int max(int x, int y);`
- `int a, b, c;`
- `cout << "two integer numbers: ";`
- `cin >> a >> b;`
- `c=max(a, b);`
- `cout << "max is %d\n" ;`
- `}`

函数调用时的数据传递

`c=max(a,b);`

(main函数)

`int max(int x, int y)`

(max函数)

{

`int z;`

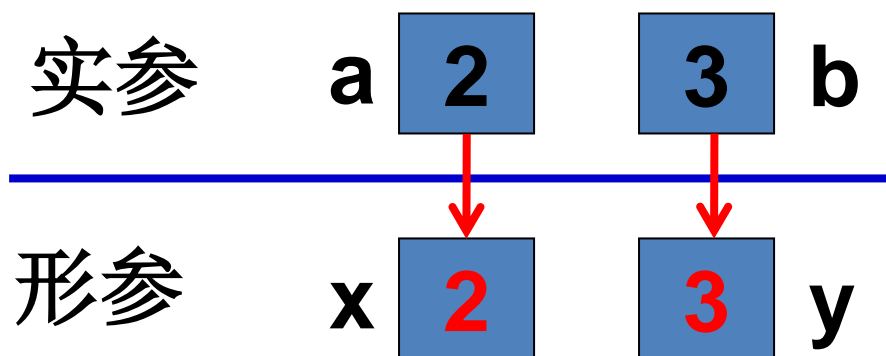
`z=x>y?x:y;`

`return(z);`

}

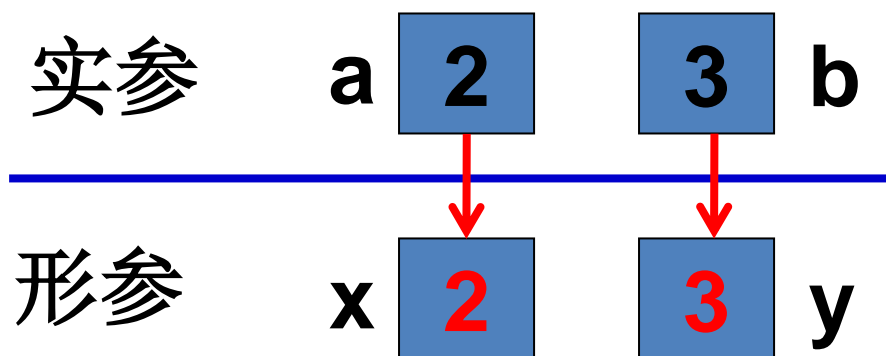
函数调用的过程

- 在定义函数中指定的形参，在未出现函数调用时，它们并不占内存中的存储单元。在发生函数调用时，函数max的形参被临时分配内存单元。



函数调用的过程

- 调用结束，形参单元被释放
- 实参单元仍保留并维持原值，没有改变
- 如果在执行一个被调用函数时，形参的值发生改变，不会改变主调函数的实参的值



函数的返回值

- 通常，希望通过函数调用使主调函数能得到一个确定的值，这就是函数值(函数的返回值)
- 函数的返回值是通过函数中的return语句获得的。
- 一个函数中可以有一个以上的return语句，执行到哪一个return语句，哪一个就起作用
- return语句后面的括号可以不要
- 通常，希望通过函数调用使主调函数能得到一个确定的值，这就是函数值(函数的返回值)
- 函数值的类型。应当在定义函数时指定函数值的类型
- 通常，希望通过函数调用使主调函数能得到一个确定的值，这就是函数值(函数的返回值)

函数的返回值

- 在定义函数时指定的函数类型一般应该和return语句中的表达式类型一致
- 如果函数值的类型和return语句中表达式的值不一致，则以函数类型为准

函数的返回值

- 将在max函数中定义的变量z改为float型。函数返回值的类型与指定的函数类型不同，分析其处理方法。
- 解题思路：如果函数返回值的类型与指定的函数类型不同，按照赋值规则处理。

```
#include <iostream.h>
```

```
int main()
```

```
{ int max(float x,float y);
```

```
float a,b; int c;
```

```
scanf("%f,%f",&a,&b);
```

```
c=max(a,b);
```

变为2

```
printf("max is %d\n",c);
```

```
return 0;
```

```
}
```

```
int max(float x,float y)
```

```
{ float z;
```

```
z=x>y?x:y;
```

```
return( z );
```

```
}
```

1.5

2.6



2.6



2

```
1.5,2.6  
max is 2
```

对被调用函数的声明和函数原型

- 分别编写add函数和main函数，它们组成一个源程序文件
- main函数的位置在add函数之前
- 在main函数中对add函数进行声明



```
#include <iostream>
int main()
{ float add(float x, float y);
  float a,b,c;
  cout << "Please enter a and b:";
  cin >> a >> b;
  c=add(a,b);
  printf("sum is %f\n",c);
  return 0;
}
```

对**add**函数声明

```
float add(float x,float y)
{ float z;
  z=x+y;
  return(z);
}
```

求两个实数之和，
函数值也是实型



```
#include <stdio.h>
```

```
int main()
```

```
{ float add(float x, float y);
```

```
    float a, b, c;
```

```
    printf("Please enter a and b:");
```

```
    scanf("%f, %f", &a, &b);
```

```
    c=add(a, b);
```

```
    printf("sum is %f\n", c);
```

```
    return 0;
```

```
}
```

只差一个分号

```
float add(float x, float y)
```

```
{ float z;
```

```
    z=x+y;
```

```
    return(z);
```

```
}
```



```
#include <iostream>
int main()
```

```
{ float add(float x, float y);
```

```
float a,b,c;
```

```
cout << "Please enter a and b:" ;
```

```
cin >> a >> b;
```

```
c=add(a,b);
```

```
cout << "sum is %f\n";
```

```
return 0;
```

```
}
```

```
Please enter a and b:3.6,6.5
sum is 10.100000
```

调用**add**函数

定义**add**函数

```
float add(float x,float y)
```

```
{ float z;
```

```
z=x+y;
```

```
return(z);
```

```
}
```



函数单个定义规则 (ODR)

- 函数定义必须仅在程序中出现一次。
- 函数的声明
 - 1. 返回类型，指定函数将返回的值的类型，如果不返回任何值，则为 `void`。在 C++11 中，`auto` 是有效返回类型，可指示编译器从返回语句推断类型。在 C++14 中，还允许使用 `decltype(auto)`。
 - 2. 函数名，必须以字母或下划线开头，不能包含空格。一般而言，标准库函数名中的前导下划线指示私有成员函数，或不是供你的代码使用的非成员函数。
 - 3. 参数列表（一组用大括号限定、逗号分隔的零个或多个参数），指定类型以及可以用于在函数体内访问值的可选局部变量名。

函数

- 指示编译器将对函数的每个调用替换为函数代码本身。
- 在某个函数快速执行并且在性能关键代码段中重复调用的情况下，内联可以帮助提高性能。
- 函数定义由声明和函数体组成，括在大括号中，其中包含变量声明、语句和表达式。
- 函数体内声明的变量称为局部变量。它们会在函数退出时超出范围；因此，函数应永远不返回对局部变量的引用！
- 函数具有零种或多种类型的逗号分隔参数列表，其中每个参数都具有可以用于在函数体内访问它的名称。默认情况下，参数通过值传递给函数，这意味着函数会收到所传递的对象的副本。
- 对于大型对象，创建副本可能成本高昂，并非始终必要。若要使自变量通过引用（特别是左值引用）进行传递，请向参数添加引用限定符&。

函数

- 在函数主体内声明的变量称为局部变量。非静态局部变量仅在函数体中可见，如果它们在堆栈上声明，会在函数退出时超出范围。
- 构造局部变量并通过值返回它时，编译器通常可以执行所谓的返回值优化以避免不必要的复制操作。
- 如果通过引用返回局部变量，则编译器会发出警告，因为调用方为使用该引用而进行的任何尝试会在局部变量已销毁之后进行。

函数

- 一个C程序由一个或多个程序模块组成，每一个程序模块作为一个源程序文件。对较大的程序，一般不希望把所有内容全放在一个文件中，而是将它们分别放在若干个源文件中，由若干个源程序文件组成一个C程序。这样便于分别编写、分别编译，提高调试效率。一个源程序文件可以为多个C程序共用。
- 一个源程序文件由一个或多个函数以及其他有关内容（如预处理指令、数据声明与定义等）组成。
- C程序的执行是从main函数开始的，如果在main函数中调用其他函数，在调用后流程返回到main函数，在main函数中结束整个程序的运行。
- 所有函数都是平行的，即在定义函数时是分别进行的，是互相独立的。
- 从用户使用的角度看，函数有两种。
 - 库函数，它是由系统提供的，用户不必自己定义而直接使用它们。应该说明，不同的C语言编译系统提供的库函数的数量和功能会有一些不同，当然许多基本的函数是共同的。
 - 用户自己定义的函数。它是用以解决用户专门需要的函数。

函数

从函数的形式看，函数分两类。

- ① 无参函数。无参函数一般用来执行指定的一组操作。无参函数可以带回或不带回函数值，但一般以不带回函数值的居多。
- ② 有参函数。在调用函数时，主调函数在调用被调用函数时，通过参数向被调用函数传递数据，一般情况下，执行被调用函数时会得到一个函数值，供主调函数使用。

函数重载

- C++ 允许在同一范围内指定多个同名函数。这些函数称为重载函数或重载。
- 利用重载函数，你可以根据参数的类型和数量为函数提供不同的语义。

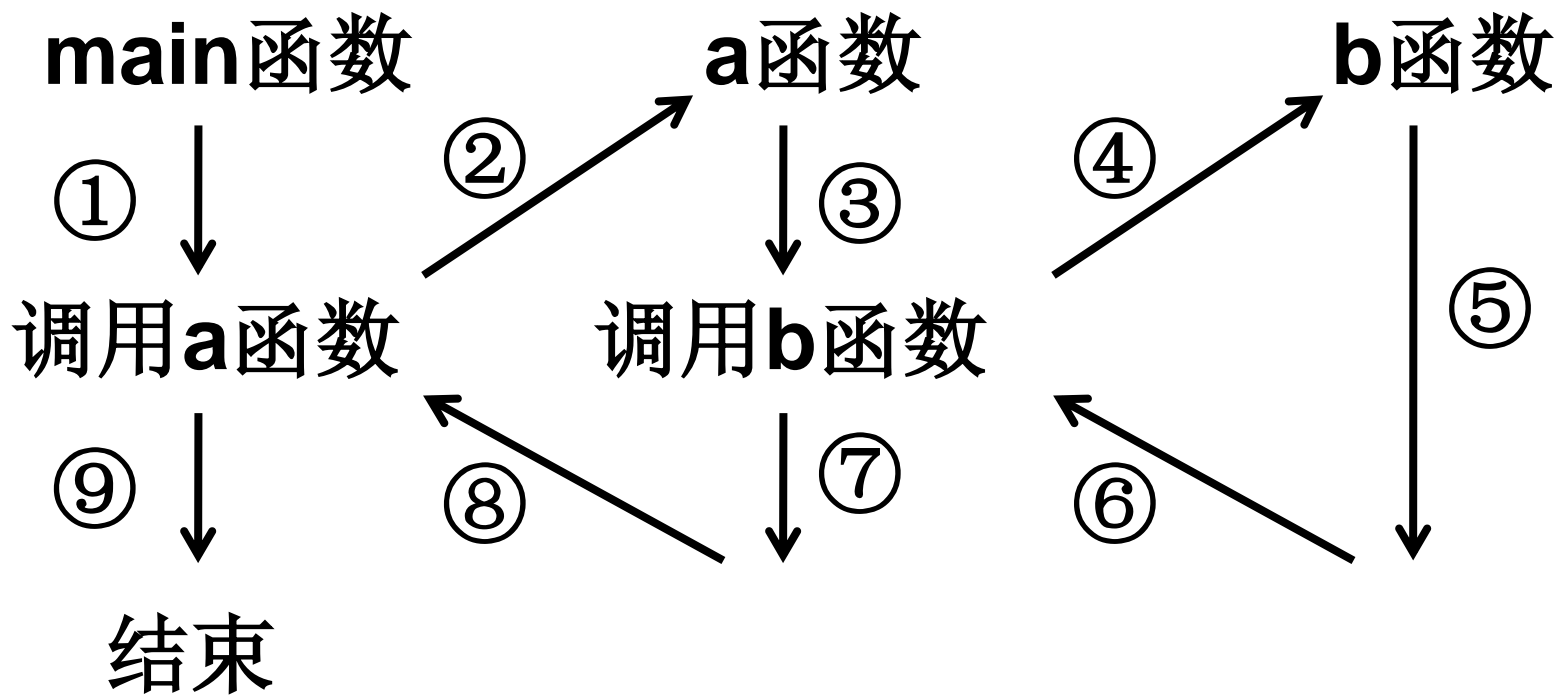
```
SumFunctionOverload.cpp  x
HelloGeometricEngine  (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间           //Line 2
3  //定义求和函数           //Line 3
4  int MySum(const int& theIntValueA, const int& theIntValueB) //Line 4
5  {                             //Line 5
6      return theIntValueA + theIntValueB;           //Line 6
7  }                             //Line 7
8  //重载函数           //Line 8
9  auto MySum(const double& theIntValueA, const double& theIntValueB) //Line 9
10 {                             //Line 10
11     return theIntValueA + theIntValueB;           //Line 11
12 }                             //Line 12
13 //主函数           //Line 13
14 int main(int argv, char* args[]) //Line 14
15 {                             //Line 15
16     int theIntValue = MySum(10, 20);           //Line 16
17     cout << "10和20的和为: " << theIntValue << endl; //Line 17
18     auto theDoubleValue = MySum(11.1, 20.2); //Line 18
19     cout << "11.1和20.2的和为: " << theDoubleValue << endl; //Line 19
20     system("pause");           //等待用户按键           //Line 20
21     return 0;                 //Line 21
22 }
```

函数的嵌套调用

- C 语言的函数定义是互相平行、独立的
- 即函数不能嵌套定义
- 但可以嵌套调用函数
- 即调用一个函数的过程中，又可以调用另一个函数

函数的嵌套调用



函数的嵌套调用

- 输入4个整数，找出其中最大的数。用函数的嵌套调用来处理。
- 解题思路：

main中调用max4函数，找4个数中最大者

max4中再调用max2，找两个数中的大者

max4中多次调用max2，可找4个数中的大者，然后把它作为函数值返回

main函数

main函数中输出结果

函数的嵌套调用

主函数

```
include <iostream>
```

```
int main()
```

```
{  
    int max4(int a, int b, int c, int d);  
    int a, b, c, d, max;  
    cout << "4 interger numbers:";  
    cin >> a >> b >> c >> d;  
    max=max4(a, b, c, d);  
    cout << max;  
    return 0;  
}
```

对max4 函数声明

函数的嵌套调用

主函数

```
include <iostream>
```

```
int main()
```

```
{
```

```
    int max4(int a, int b, int c, int d);
```

```
    int a, b, c, d, max;
```

```
    cout << "4 interger numbers: ";
```

```
    cin >> a >> b >> c >> d;
```

```
    max=max4(a, b, c, d);
```

```
    cout << max;
```

```
    return 0;
```

```
}
```

对max4 函数声明

输入4个整数

函数的嵌套调用

主函数

```
include <iostream>
```

```
int main()
```

```
{  
    int max4(int a, int b, int c, int d);  
    int a, b, c, d;  
    cout << "4个整数中最大者: ";  
    cin >> a >> b >> c >> d;  
    max=max4(a, b, c, d);  
    cout << max;  
    return 0;  
}
```

对max4 函数声明

调用后肯定是4个整数中最大者

输出最大者

max4函数

```
int max4(int a, int b, int c, int d)
{
    int max2(int a, int b);
    int m;
    m=max2(a, b);
    m=max2(m, c);
    m=max2(m, d);
    return(m);
}
```

对max2 函数声明

max4函数

```
int max4(int a, int b, int c, int d)
{  int max2(int a, int b);
```

```
    int m;
```

```
    m=max2(a, b);
```

a,b中较大者

```
    m=max2(m, c);
```

a,b,c中较大者

```
    m=max2(m, d);
```

a,b,c,d中最大者

```
    return(m);
```

```
}
```

max4函数

```
int max4(int a, int b, int c, int d)
{
    int max2(int a, int b);
    int m;
    m=max2(a, b);
    m=max2(m, c);
    m=max2(m, d);
    return(m);
}
```

max2函数

```
int max2(int a,int b)
{
    if(a>=b)
        return a;
    else
        return b;
}
```

找a,b中较大者

max4函数

```
int max4(int a, int b, int c, int d)
{
    int max2(int a, int b);
    int m;
    m=max2(a, b);
    m=max2(m, c);
    m=max2(m, d);
    return(m);
}
```

max2函数

```
int max2(int a,int b)
{
    if(a>=b)
        return a;
    else
        return b;
}
```

return(a>b?a:b);

max4函数

```
int max4(int a, int b, int c, int d)
{
    int max2(int a, int b);
    int m;
    m=max2(a, b);
    m=max2(m, c);
    m=max2(m, d);
    return(m);
}
```

```
int max2(int a,int b)
{
    return(a>b?a:b);
}
```


max4函数

```
int max4(int a, int b, int c, int d)
{
    int max2(int a, int b);
    int m;
    m=max2(a, b);
    m=max2(m, c);
    m=max2(m, d);
    return(m);
}
```

m=max2(max2(a,b),c);

```
int max2(int a,int b)
{
    return(a>b?a:b);
}
```

max4函数

```
int max4(int a, int b, int c, int d)
```

```
{  int max2(int
```

```
    int m;
```

```
    m=max2(a, b);
```

```
    m=max2(m, c);
```

```
    m=max2(m, d);
```

```
    return(m);
```

m=max2(max2(max2(a,b),c),d);

```
}
```

```
int max2(int a,int b)
```

```
{  return(a>b?a:b); }
```

max4函数

```
int max4(int a, int b, int c, int d)
{
    return max2(max2(max2(a,b),c),d);
    int m;
    m=max2(a, b);
    m=max2(m, c);
    m=max2(m, d);
    return(m);
}
```

```
int max2(int a,int b)
{ return(a>b?a:b); }
```

```
#include <stdio.h>
```

```
4 integer numbers:12 45 -6 89  
max=89
```

```
int main()
```

```
{ .....
```

```
max=max4(a,b,c,d);
```

```
.....
```

```
}
```

```
int max4(int a, int b, int c, int d)
```

```
{ int max2(int a, int b);
```

```
return max2(max2(max2(a, b), c), d);
```

```
}
```

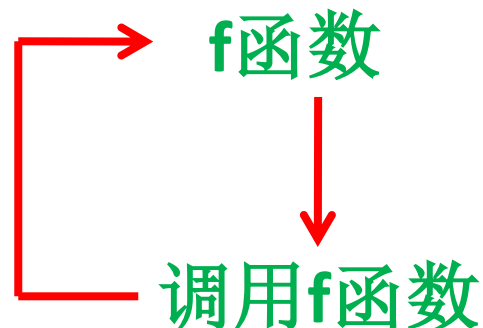
```
int max2(int a, int b)
```

```
{ return(a>b?a:b); }
```

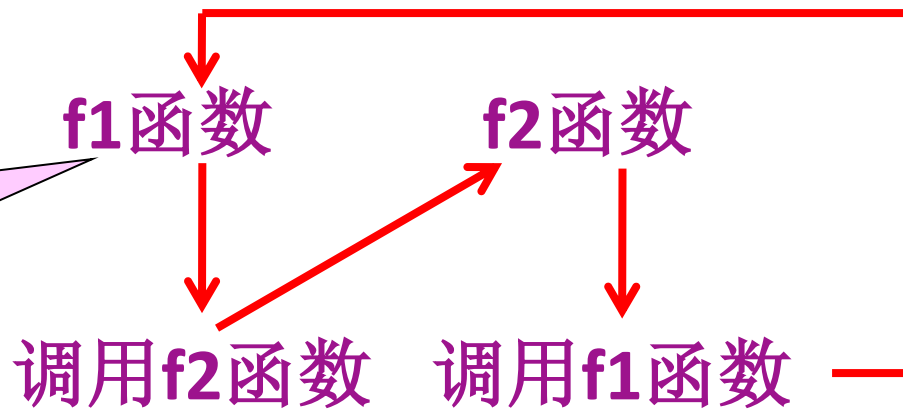
函数的递归调用

```
int f(int x)
{
    int z;
    return (2*z);
}
```

直接调用本函数



间接调用本函数



应使用if语句控制结束调用

函数递归

```
FactorialFunction.cpp  x
HelloGeometricEngine  (全局范围)

1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间               //Line 2
3  //定义阶乘函数               //Line 3
4  int MyFactorialFunction(const int& theIntValue)           //Line 4
5  {                             //Line 5
6      if (theIntValue == 1)     //Line 6
7          return 1;            //Line 7
8      else                     //Line 8
9          return MyFactorialFunction(theIntValue - 1) * theIntValue; //Line 9
10 }                             //Line 10
11 //主函数                     //Line 11
12 int main(int argv, char* args[]) //Line 12
13 {                             //Line 13
14     int theIntValue;           //Line 14
15     //调用函数                 //Line 15
16     theIntValue = MyFactorialFunction(10); //Line 16
17     //输出结果                 //Line 17
18     cout << "10的阶乘为: " << theIntValue << endl; //Line 18
19     system("pause");           //Line 19
20     return 0;                 //Line 20
21 }                             //Line 21
22                               //Line 22
23                               //Line 23
24                               //Line 24
```

函数递归

FactorialFunction.cpp

HelloGeometricEngine (全局范围)

```
1  #include <iostream>           //输入输出头文件           //Line 1
2  using namespace std;         //std命名空间           //Line 2
3  //定义阶乘函数           //Line 3
4  int MyFactorialFunction(const int& theIntValue)           //Line 4
5  {           //Line 5
6      if (theIntValue == 1)           //Line 6
7          return 1;           //Line 7
8      //Line 8
9      return MyFactorialFunction(theIntValue - 1) * theIntValue; //Line 9
10 //Line 10
11 //Line 11
12 int main(int argv, char* args[]) //Line 12
13 { //Line 13
14     int theIntValue; //Line 14
15     //Line 15
16     //调用函数 //Line 16
17     theIntValue = MyFactorialFunction(10); //Line 17
18     //Line 18
19     //输出结果 //Line 19
20     cout << "10的阶乘为: " << theIntValue << endl; //Line 20
21     //Line 21
22     system("pause"); //等待用户按键 //Line 22
23     return 0; //Line 23
24 }
```

删除断点(E)
禁用断点(D) Ctrl+F9
条件(C)... Alt+F9, C
操作(A)...
编辑标签(L)... Alt+F9, L
导出(X)...

函数递归

FactorialFunction.cpp

HelloGeometricEngine (全局范围) MyFacto

```
1 #include <iostream> //输入输出头文件 //Line 1
2 using namespace std; //std命名空间 //Line 2
3 //定义阶乘函数 //Line 3
4 int MyFactorialFunction(const int& theIntValue) //Line 4
5 { //Line 5
6     if (theIntValue == 1) //Line 6
7         return 1; //Line 7
8     else //Line 8
9         return MyFactorialFunction(theIntValue - 1) * theIntValue; //Line 9
10 } //Line 10
11 //主函数 //Line 11
12 int main(int argv, char* args[]) //Line 12
13 { //Line 13
14     int theIntValue; //Line 14
15     //Line 15
16     //调用函数 //Line 16
17     theIntValue = MyFactorialFunction(10); //Line 17
18     //Line 18
19     //输出结果 //Line 19
20     cout << "10的阶乘为: " << theIntValue << endl; //Line 20
21     //Line 21
22     system("pause"); //等待用户按键 //Line 22
23     return 0; //Line 23
24 }
```

调用堆栈

搜索(Ctrl+E) 查看所有线程 显示外部代码

名称	语言
HelloGeometricEngine.exe!MyFactorialFunction(const int & theIntValue) 行 6	C++
HelloGeometricEngine.exe!MyFactorialFunction(const int & theIntValue) 行 9	C++
HelloGeometricEngine.exe!MyFactorialFunction(const int & theIntValue) 行 9	C++
HelloGeometricEngine.exe!MyFactorialFunction(const int & theIntValue) 行 9	C++
HelloGeometricEngine.exe!MyFactorialFunction(const int & theIntValue) 行 9	C++
HelloGeometricEngine.exe!main(int argv, char ** args) 行 17	C++
(外部代码)	

调用堆栈 断点 异常设置 命令窗口 即时窗口 输出

函数的递归调用

- 有5个学生坐在一起
- 问第5个学生多少岁？他说比第4个学生大2岁
- 问第4个学生岁数，他说比第3个学生大2岁
- 问第3个学生，又说比第2个学生大2岁
- 问第2个学生，说比第1个学生大2岁
- 最后问第1个学生， he说是10岁
- 请问第5个学生多大

函数的递归调用

- 解题思路：
- 要求第 5 个年龄，就必须先知道第 4 个年龄
- 要求第 4 个年龄必须先知道第 3 个年龄
- 第 3 个年龄又取决于第 2 个年龄
- 第 2 个年龄取决于第 1 个年龄
- 每个学生年龄都比其前 1 个学生的年龄大 2

函数的递归调用

解题思路:

$$\text{age}(5) = \text{age}(4) + 2$$

$$\text{age}(4) = \text{age}(3) + 2$$

$$\text{age}(3) = \text{age}(2) + 2$$

$$\text{age}(2) = \text{age}(1) + 2$$

$$\text{age}(1) = 10$$

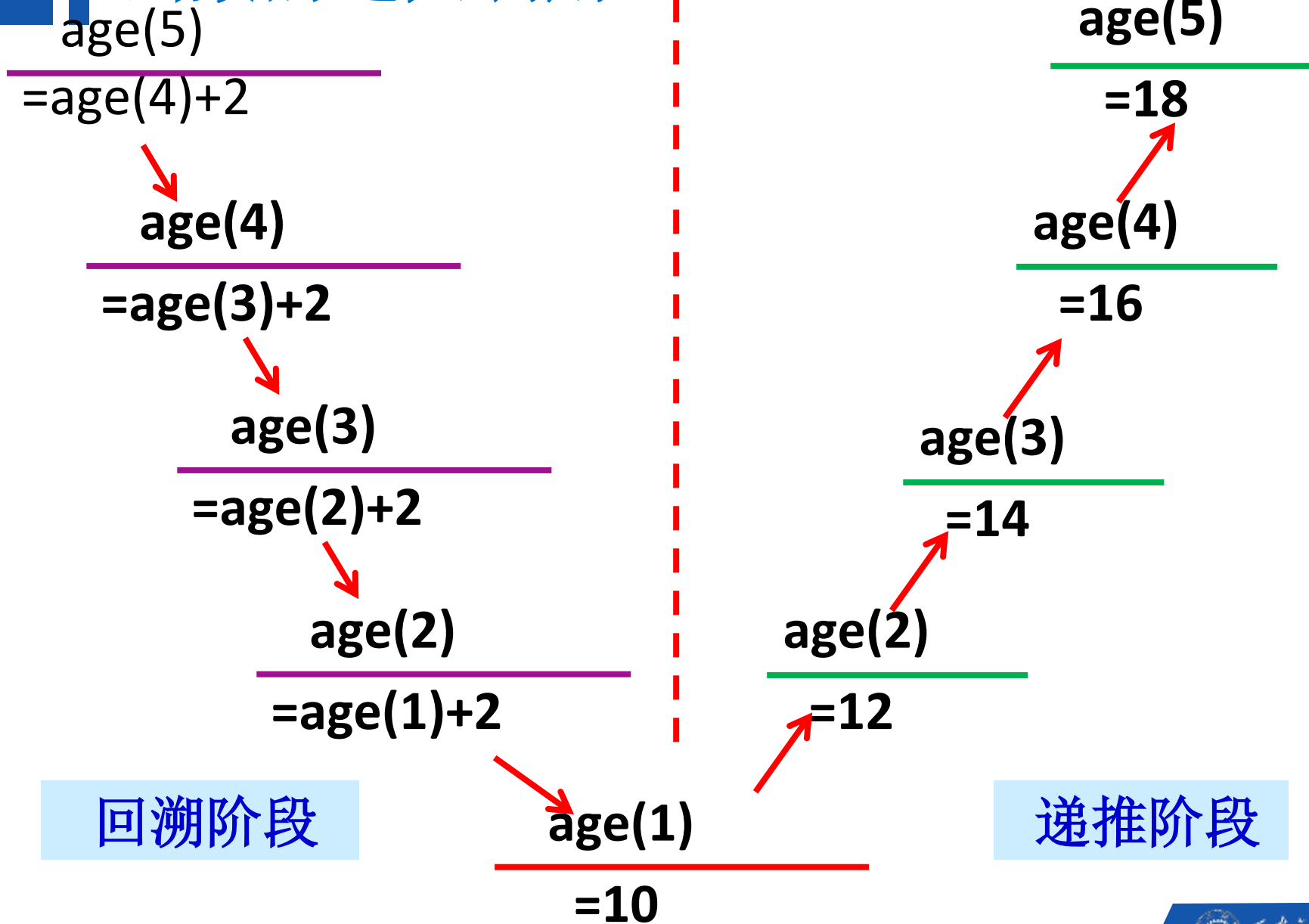


$$\text{age}(n) = 10 \quad (n = 1)$$

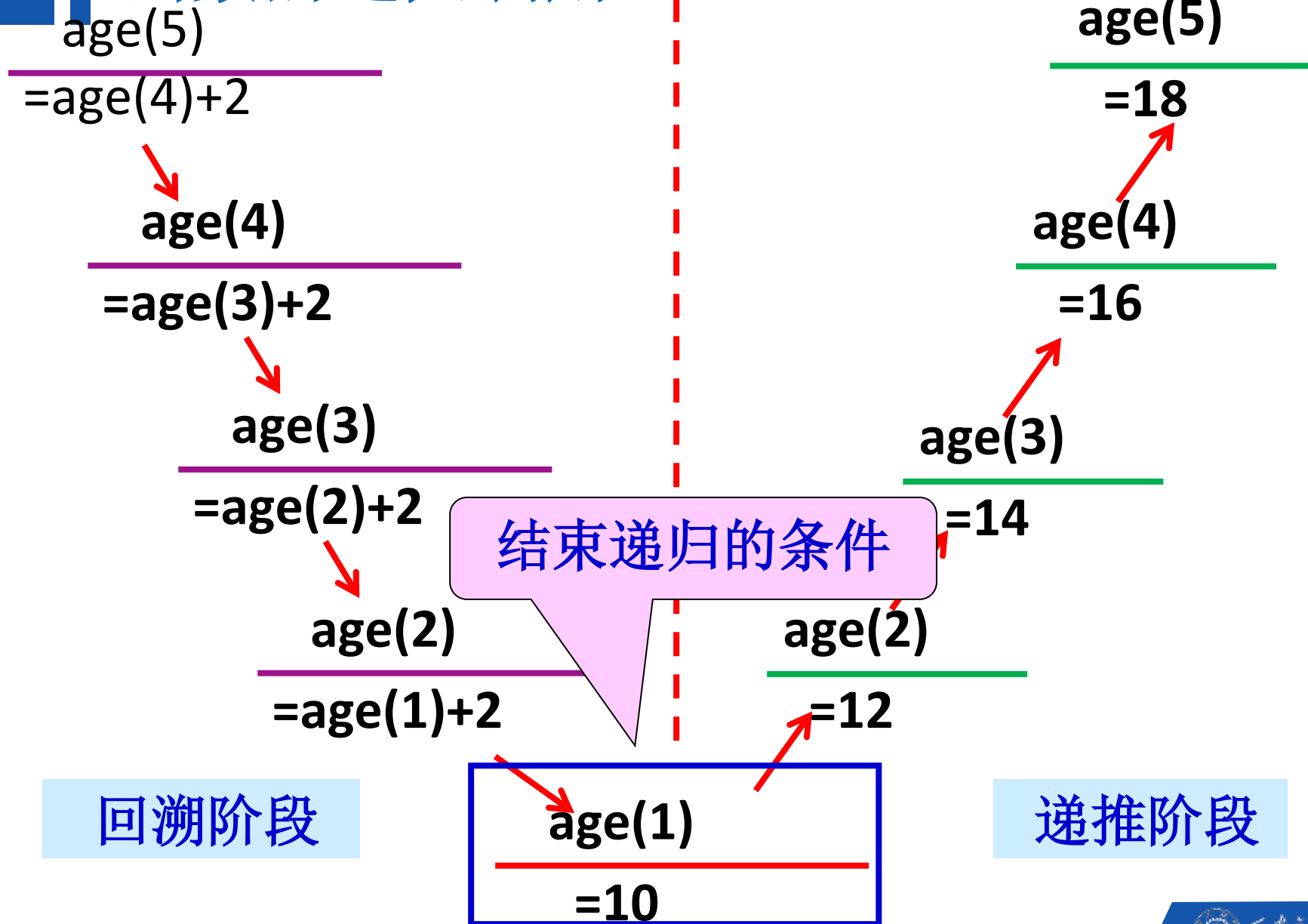
$$\text{age}(n) = \text{age}(n - 1) + 2 \quad (n > 1)$$



函数的递归调用



函数的递归调用



函数的递归调用

```
#include <iostream>
int main()
{ int age(int n);
  cout << age(5);
  return 0;
}
int age(int n)
{ int c;
  if(n==1) c=10;
  else c=age(n-1)+2;
  return(c);
}
```

递归函数

- 用递归方法求 $n!$ 。
- 解题思路：
- 求 $n!$ 可以用递推方法：即从 1 开始，乘 2，再乘 3……一直乘到 n 。
- 递推法的特点是从一个已知的事实(如 $1!=1$) 出发，按一定规律推出下一个事实(如 $2!=1!*2$)，再从这个新的已知的事实出发，再向下推出一个新的事实($3!=3*2!$)。 $n!=n*(n-1)!$ 。

递归函数

用递归方法求 $n!$ 。

解题思路：

求 $n!$ 也可以用递归方法，即 $5!$ 等于 $4! \times 5$ ，
而 $4! = 3! \times 4 \dots$ ， $1! = 1$

可用下面的递归公式表示：

$$n! = \begin{cases} n! = 1 & (n = 0, 1) \\ n \bullet (n - 1) & (n > 1) \end{cases}$$

递归函数

```
#include <iostream>

Int main()
{int fac(int n);
 int n; int y;
 cout << "input an integer number:";
 cin >> n;
 y=fac(n);
 cout << n << y;
 return 0;
}
```

递归函数

```
int fac(int n)
{
    int f;
    if(n<0)
        cout << "n<0,data error!";
    else if(n==0 | | n==1)
        f=1;
    else f=fac(n-1)*n;
    return(f);
}
```

数组作为函数参数

数组元素作函数实参

数组名作函数参数

多维数组名作函数参数



数组元素作函数实参

输入10个数，要求输出其中值最大的元素和该数是第几个数。

数组元素作函数实参

解题思路:

- 定义数组a，用来存放10个数
- 设计函数max，用来求两个数中的大者
- 在主函数中定义变量m，初值为a[0]，每次调用max函数后的返回值存放在m中
- 用“打擂台”算法，依次将数组元素a[1]到a[9]与m比较，最后得到的m值就是10个数中的最大者

```
include <iostream>
int main()
{  int max(int x, int y);
    int a[10], m, n, l;
    cout << "10 integer numbers:\n";
    for (i=0; i<10; i++)
        cin >> a[i];
    cout << "\n";
```

```
10 integer numbers:
4 7 0 -3 4 34 67 -42 31 -76
```

```
for (i=1, m=a[0], n=0; i<10; i++)
```

```
{ if (max(m, a[i])>m)
```

```
{ m=max(m, a[i]);
```

```
  n=i;
```

```
}
```

```
}
```

```
cout << "largest number is " << m;
```

```
cout << n+1 << "th number.\n";
```

```
}
```

largest number is 67
7th number.

```
int max(int x,int y)
```

```
{ return(x>y?x:y); }
```

数组名作函数参数

- 除了可以用数组元素作为函数参数外，还可以用数组名作函数参数(包括实参和形参)
- 用数组元素作实参时，向形参变量传递的是数组元素的值
- 用数组名作函数实参时，向形参传递的是数组首元素的地址

7.7.2数组名作函数参数

有一个一维数组score，内放10个学生成绩，求平均成绩。

解题思路：

- 用函数average求平均成绩，用数组名作为函数实参，形参也用数组名
- 在average函数中引用各数组元素，求平均成绩并返回main函数

```
#include <iostream>
int main()
{ float average(float array[10]);
  float score[10], aver;  int i;
  cout << "input 10 scores:\n";
  for (i=0; i<10; i++)
    cin >> score[i];
  cout << "\n";
  aver=average(score);
  cout << aver;
  return 0;
}
```

定义实参数组


定义形参数组

```
float average(float array[10])  
{   int i;  
    float aver, sum=array[0];  
    for (i=1; i<10; i++)  
        sum=sum+array[i];  
    aver=sum/10;  
    return (aver);  
}
```

相当于score[0]

相当于score[i]

```
input 10 scores:  
100 56 78 98 67.5 99 54 88.5 76 58  
  
77.50
```



有两个班级，分别有35名和30名学生，编写一个**average**函数，分别调用求这两个班的学生们的平均成绩。

解题思路:

- 需要解决怎样用同一个函数求两个不同长度的数组的平均值的问题
- 定义**average**函数时不指定数组的长度，在形参表中增加一个整型变量*i*
- 从主函数把数组实际长度从实参传递给形参*i*
- 这个*i*用来在**average**函数中控制循环的次数
- 为简化，设两个班的学生数分别为5和10

```
include <iostream>
int main()
{ float average(float array[ ], int n);
  float score1[5]={98.5, 97, 91.5, 60, 55};
  float score2[10]={67.5, 89.5, 99, 69.5,
                    77, 89.5, 76.5, 54, 60, 99.5};
  cout << average(score1, 5);
  cout << average(score2, 10);
  return 0;
}
```

调用形式为average(score1,5)时

```
float average(float array[ ], int n)
{ int i;
  float aver; sum=array[0];
  for (i=1; i<n; i++)
    sum=sum+array[i];
  aver=sum/n;
  return (aver);
}
```

相当于5

相当于score1[0]

相当于score1[i]



调用形式为average(score2,10)时

```
float average(float array[], int n)
{ int i;
  float aver, sum=array[0];
  for (i=1; i<n; i++)
    sum=sum+array[i];
  aver=sum/n;
  return (aver);
}
```

相当于10

相当于score2[0]

相当于score2[i]

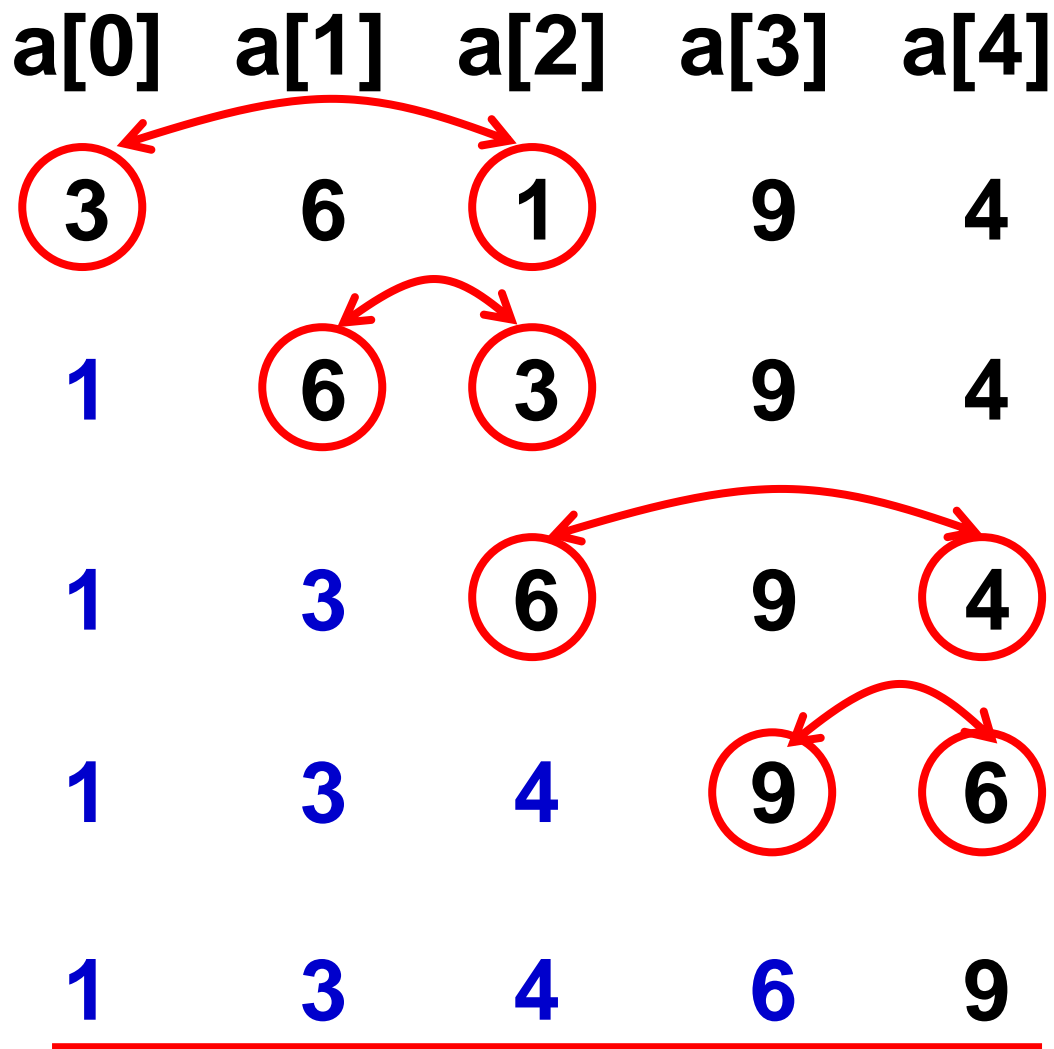
80.40
78.20



用选择法对数组中10个整数按由小到大排序。

解题思路：

- 所谓选择法就是先将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



小到大排序

```
#include <iostream>
int main()
{ void sort(int array[], int n);
  int a[10], i;
  cout << "enter array:\n";
  for (i=0; i<10; i++)  cin >> a[i];
  sort(a, 10);
  cout << "The sorted array:\n";
  for (i=0; i<10; i++)  cin >> a[i];
  cout << "\n";
  return 0;
}
```

```
void sort(int array[], int n)
```

```
{ int i, j, k, t;
```

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

```
  { k=i;
```

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

```
      if (array[j]<array[k])      k=j;
```

```
    t=array[k];
```

```
    array[k]=array[i];
```

```
    array[i]=t;
```

```
  }
```

```
}
```

在sort[i]~sort[9]中，
最小数与sort[i]对换

```
enter array:
```

```
45 2 9 0 -3 54 12 5 66 33
```

```
The sorted array:
```

```
-3 0 2 5 9 12 33 45 54 66
```



7.7.3 多维数组名作函数参数

有一个 3×4 的矩阵，求所有元素中的最大值。

1	3	5	7
2	4	6	8
15	17	34	12

解题思路：先使变量max的初值等于矩阵中第一个元素的值，然后将矩阵中各个元素的值与max相比，每次比较后都把“大者”存放在max中，全部元素比较完后，max 的值就是所有元素的最大值。

不能省略
要与形参数组第二维大小相同

```
include <iostream>
int main()
{ int max_value(int array[][4]);
  int a[3][4]={ {1, 3, 5, 7}, {2, 4, 6, 8},
                {15, 17, 34, 12} };
  cout << "Max value is" << max_value(a);
  return 0;
}
```

要与实参数组第二维大小相同

```
int max_value(int array[][4])
{   int i, j, max;
    max = array[0][0];
    for (i=0; i<3; i++)
        for (j=0; j<4; j++)
            if (array[i][j]>max)
                max = array[i][j];
    return (max);
}
```

局部变量和全局变量

局部变量
全局变量



局部变量

定义变量可能有三种情况：

在函数的开头定义

在函数内的复合语句内定义

在函数的外部定义

局部变量

- 在一个函数内部定义的变量只在本函数范围内有效
- 在复合语句内定义的变量只在本复合语句范围内有效
- 在函数内部或复合语句内部定义的变量称为“局部变量”

```
float f1( int a)
{ int b, c;
  .....
}
```

a、b、c仅在此函数内有效

```
char f2(int x, int y)
{ int i, j;
  .....
}
```

x、y、i、j仅在此函数内有效

```
int main( )
{ int m, n;
  .....
  return 0;
}
```

m、n仅在此函数内有效

```
float f1( int a)
{ int b, c;
  .....
}
char f2(int x, int y)
{ int i, j;
  .....
}
int main( )
{ int a, b;
  .....
  return 0;
}
```

类似于不同
班同名学生

a、b也仅在此
函数内有效

```
int main ( )  
{ int a,b;
```

.....

```
{ int c;  
  c=a+b;
```

.....

```
}
```

.....

```
}
```

a、b仅在此复合语句内有效

c仅在此复合语句内有效

全局变量

- 在函数内定义的变量是局部变量，而在函数之外定义的变量称为外部变量
- 外部变量是全局变量(也称全程变量)
- 全局变量可以为本文件中其他函数所共用
- 有效范围为从定义变量的位置开始到本源文件结束

```
int p=1, q=5
```

```
float f1(int a)
```

```
{ int b, c; ..... }
```

```
char c1, c2;
```

```
char f2 (int x, int y)
```

```
{ int i, j; ..... }
```

```
int main ( )
```

```
{ int m, n;
```

```
.....
```

```
return 0;
```

```
}
```

**p、q、c1、c2
为全局变量**

有一个一维数组，内放10个学生成绩，写一个函数，当主函数调用此函数后，能求出平均分、最高分和最低分。

解题思路：调用一个函数可以得到一个函数返回值，现在希望通过函数调用能得到3个结果。可以利用全局变量来达到此目的。


```
#include <iostream>
float Max=0, Min=0;
int main()
{ float average(float array[ ], int n);
  float ave, score[10];  int i;
  cout << "Please enter 10 scores:\n";
  for (i=0; i<10; i++)
    cin >> score[i];
  ave=average(score, 10);
  cout << Max << Min << ave;
  return 0;
}
```

```

float average(float array[ ], int n)
{ int i; float aver, sum=array[0];
  Max=Min=array[0];
  for (i=1; i<n; i++)
  { if(array[i]>Max) Max=array[i];
    else if(array[i]<Min) Min=array[i];
    sum=sum+array[i];
  }
  aver=sum/n;
  return(aver);
}

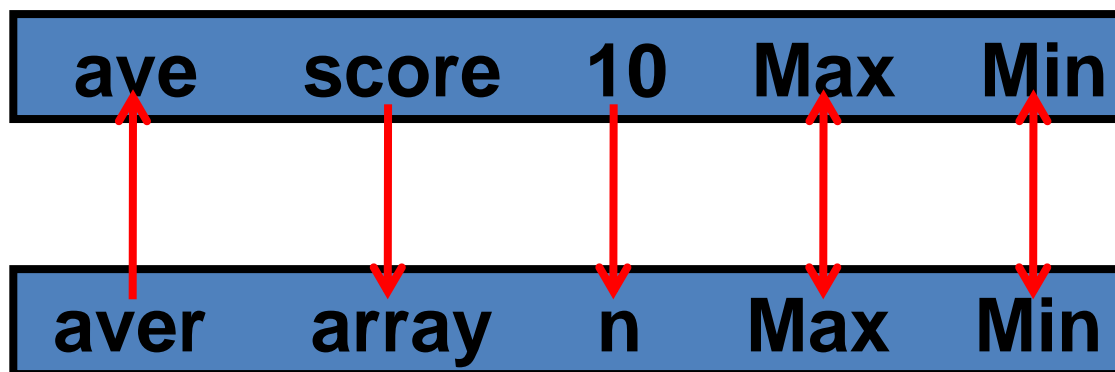
```

```

Please enter 10 scores:
89 95 87.5 100 67.5 97 59 84 73 90
max=100.00
min= 59.00
average= 84.20

```





main
函数

average
函数

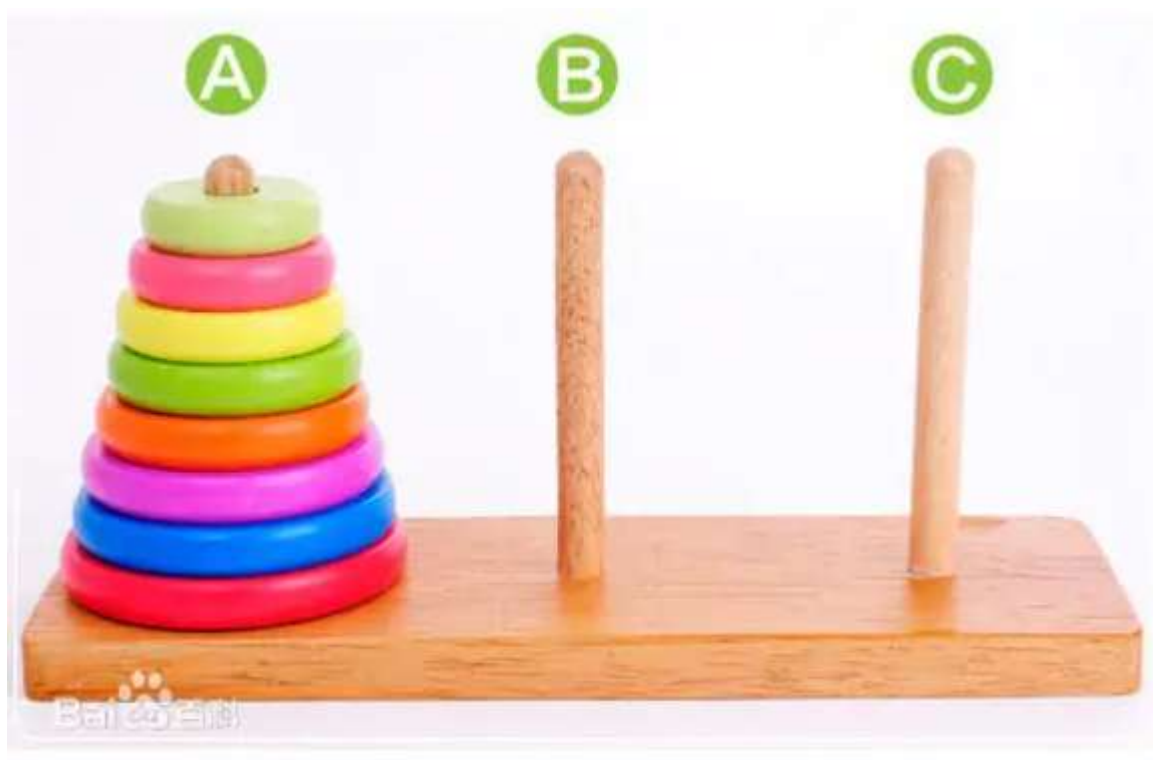
建议不在必要时不要使用全局变量

关于变量的声明和定义

- 一般为了叙述方便，把建立存储空间的变量声明称定义，而把不需要建立存储空间的声明称为声明
- 在函数中出现的对变量的声明(除了用**extern**声明的以外)都是定义
- 在函数中对其他函数的声明不是函数的定义



函数递归实现汉诺塔。



结构化程序设计

- 工业软件的规模越来越大，程序的功能也越来越强，参与项目开发的程序员也越来越多。
- 随着软件规模的扩大，程序的可维护性变得越来越重要，一个程序需要具有良好的组织架构，其功能模块和函数的划分要明确，具有实际的含义。
- 如何将大规模的程序拆解成意义明确，规模较小的功能模块，让不同的人参与进来负责不同的模块开发在软件工程中具有非常重要的意义。
- 它为大规模程序设计提供了指导性思路和设计原则，通过模块划分将大的任务分解成规模小的子任务，实现复杂任务的拆解，有利于程序的开发和后期维护。

结构化程序设计

- 结构化程序设计的难点在于任务的合理分解，每一个子任务和项目中明确完整的子业务逻辑对应，任务之间的调用关系体现了业务流程。
- 任务分解的颗粒太大，则每一个子任务的逻辑太复杂；颗粒太小则太繁琐。分解的颗粒度一般情况下将子任务分解成可单独实现的子功能为宜。
- C++通过函数调用将子任务设计成独立的子函数实现，然后在主函数中调用子函数完成整体的任务。程序员采取以下方法得到结构化的程序：
 - （1） 自顶向下；
 - （2） 逐步细化；
 - （3） 模块化设计；
 - （4） 结构化编码。

结构化程序设计

- 结构化程序设计强调程序设计风格和程序结构的规范化，提倡清晰的结构。结构化程序设计方法的基本思路是：把一个复杂问题的求解过程分阶段进行，每个阶段处理的问题都控制在人们容易理解和处理的范围内。