第1次直播课： 2~3章 例题分析

清华大学 郑莉

通过案例串讲2、3章要点及学习方法

- 例1_1：计算商品总价（整数、实数、运算）
- 例1_2：逻辑运算及短路现象
- 例1_3：判断一个数是否为质数（选择结构）
- 例1_4：求正整数的质因子（循环与选择嵌套）
- 例1_5：寻找并输出11~999之间的数m，它满足m、m2和m3均为回文数（函数）
- 例1_6：递归过程的跟踪——以汉诺塔问题为例

例1_1：计算商品总价（整数、实数、运算）

知识点：整数与实数、常量、变量、初始化、C风格字符串常量、算数运算、赋值运算、表达式、标准输入/输出

```
//z1_1.cpp
#include<iostream>
using namespace std;
int main()
{
  const double price1 = 25.5, price2 = 10.3, price3 = 12.5;
  double total=0;
  int number1=0, number2=0, number3=0;
  cout <<"三种商品价格为:"<< price1 << ','<< price2<< ','<<price3<< '\n';
  cout <<"请问每样买几件？" << endl;
  cin >> number1 >> number2 >> number3;
  total = number1*price1 + number2*price2 + number3*price3;
  cout <<"应付款总额：" << total << endl;
  return 0;
}
```

例1_2：逻辑运算及短路现象

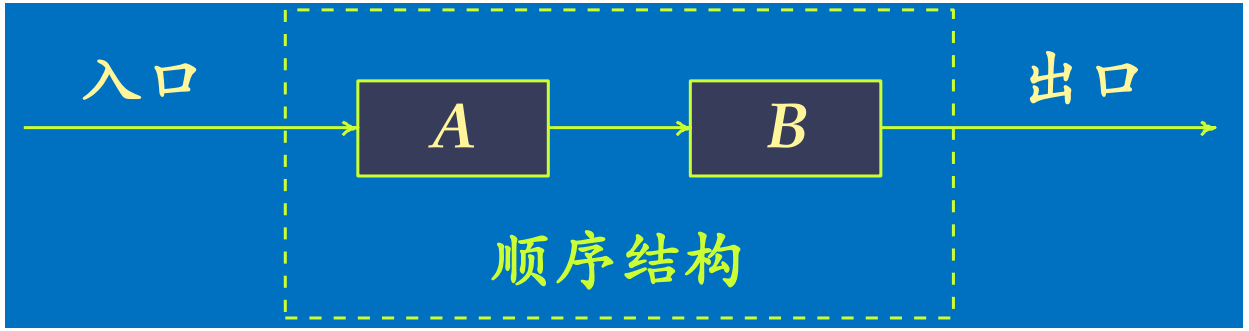
```
//z1_2
#include <iostream>
using namespace std;

int main()
{
        int a, b;
        cin>>a>>b;
        cout << ((a > 2) && (++b)) << endl;
        cout << "b = " << b << endl;
        cout << ((a > 2) || (++b)) << endl;
        cout << "b = " << b << endl;
        return 0;
}
```

算法的流程控制

顺序结构
选择结构
循环结构

顺序结构

- 顺序结构的含义
  - 由一组顺序执行的处理块组成，每个处理块可能包含一条或一组语句，完成一项任务
  - 顺序结构是最基本的算法结构



- 语句与复合语句（语句块）
  - 三种语句结构：单语句（表达式;）、空语句（;）、复合语句（{语句序列}）

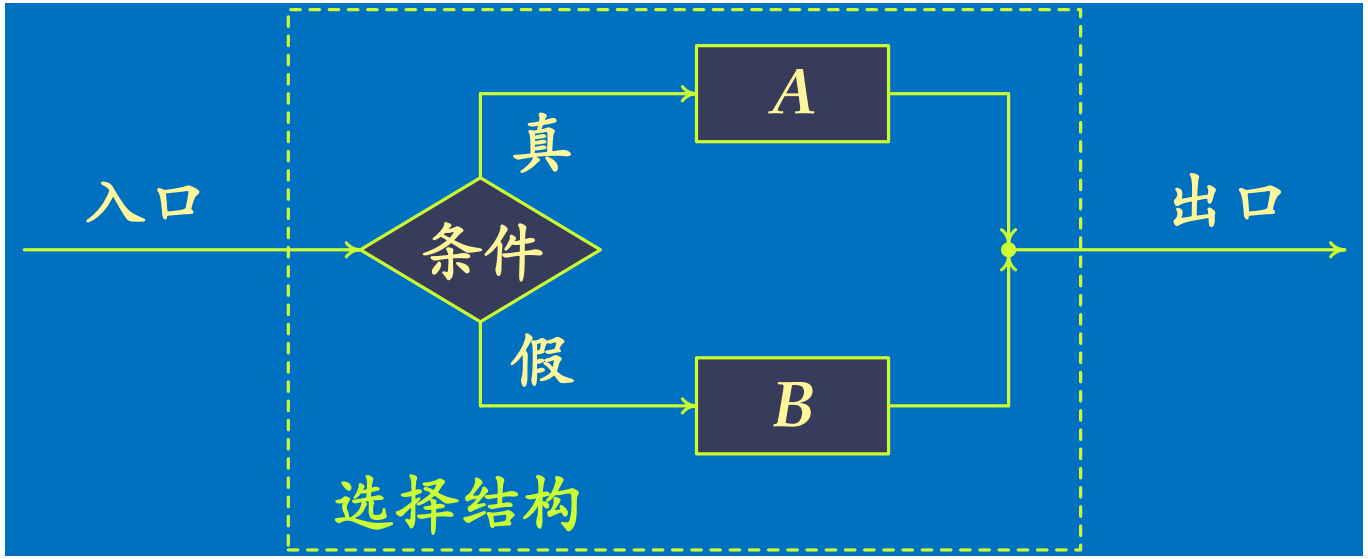
选择结构——if语句

- 选择结构的含义
  - 根据某一条件的判断结果，确定程序的流程，即选择哪一个程序分支中的处理块去执行
  - 最基本的选择结构是二路选择结构

语法形式：

if (表达式) 语句1 else 语句2

例：if (x > y) cout << x;
else cout << y;

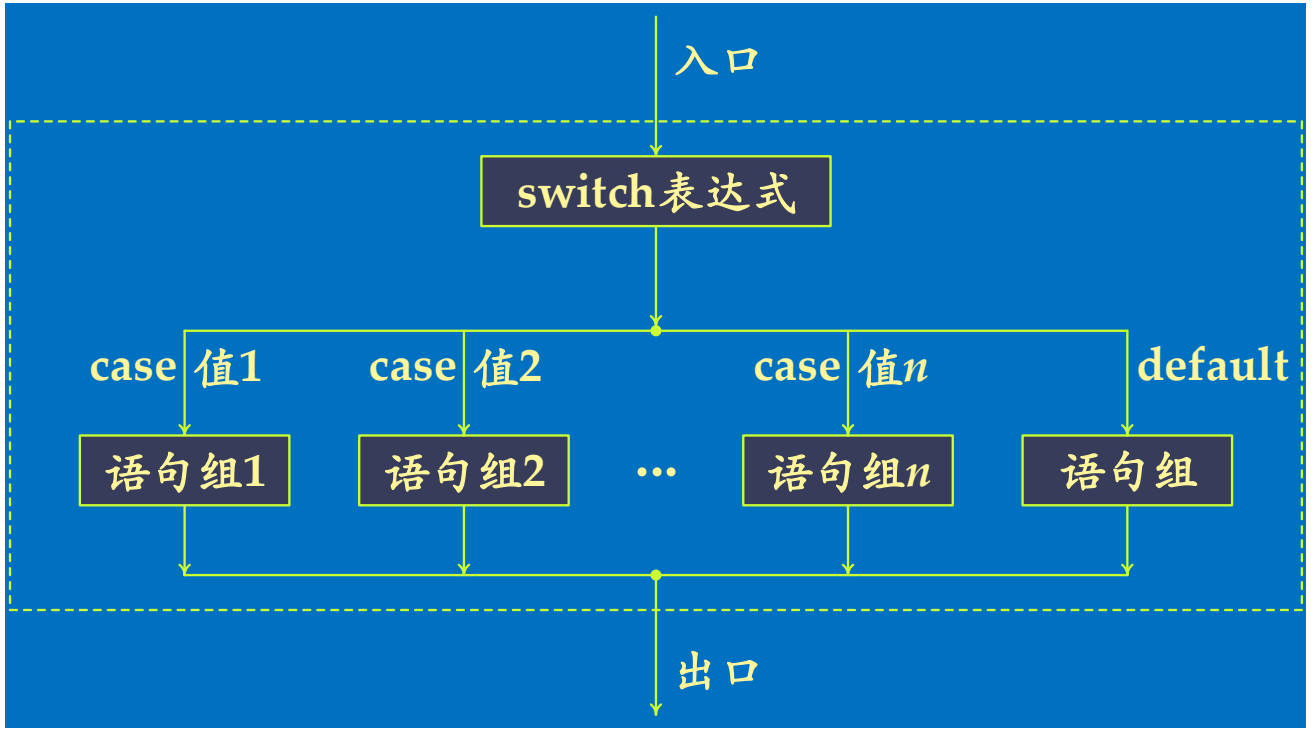


  - 以条件判断为起点，如果判断结果为真，则执行A处理块的操作，否则执行B处理块的操作

switch 语句

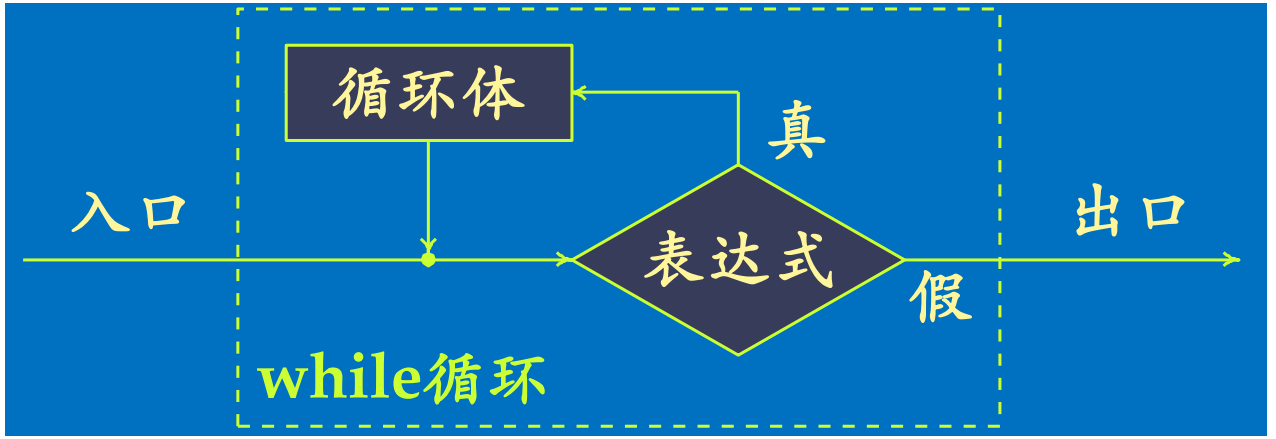
- 语法形式

```
switch (表达式)
  { case   常量表达式 1：语句1
    case   常量表达式 2：语句2
             ┆
    case   常量表达式 n：语句n
    default :          语句n+1
  }
```



循环结构——while 语句

- while 循环格式：while(表达式) 循环体
- while 循环流程
  - 先判断后执行：表达式为真时，执行一遍循环体（一次迭代），返回重新计算表达式的值以确定是否重复执行循环体；若表达式为假，则终止循环
  - 为保证循环终止，循环体内应有能改变表达式值的语句

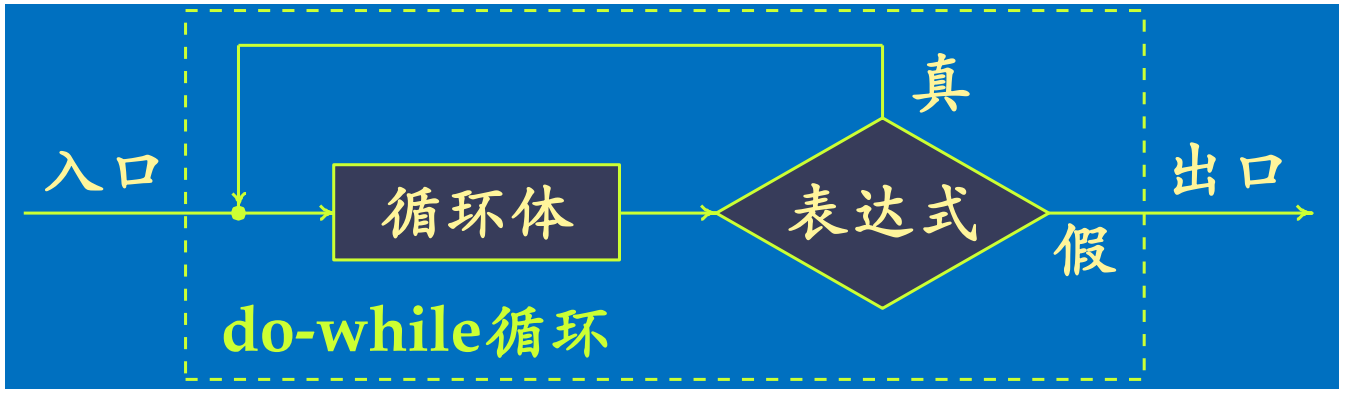


循环结构——do-while 语句

- 语法形式

  do 语句 ← 可以是复合语句，其中必须含有改变条件表达式值的语句。

  while (表达式)



- 执行顺序

  先执行语句，后判断条件。
  表达式非0时，继续执行循环体。

- while 语句与 do-while 语句的比较：

  While 语句先判断表达式的值，非 0 再执行语句

循环结构—— for 语句

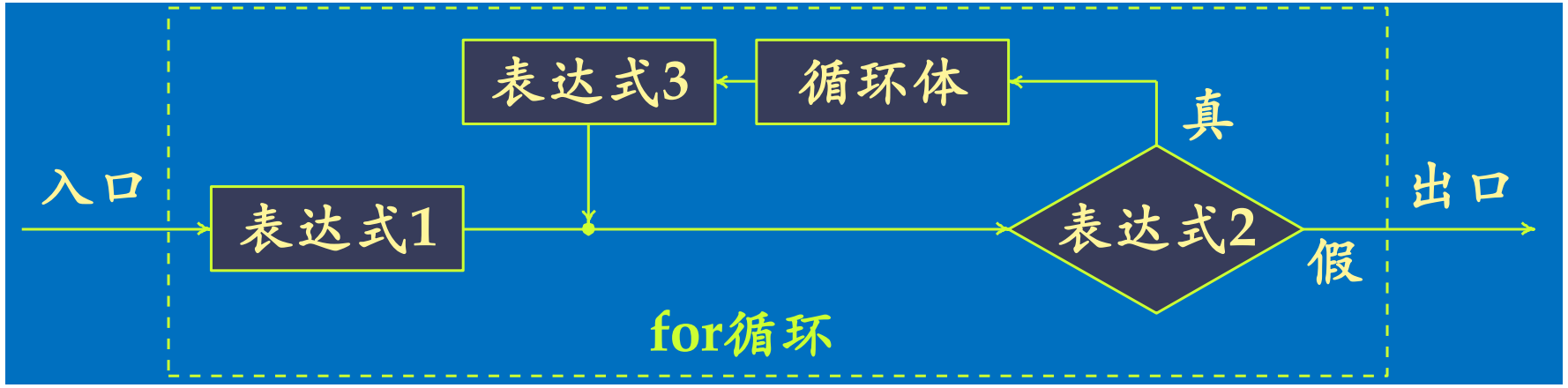
- 语法形式

for（表达式1；表达式2；表达式3）语句

循环前先求解

非0时执行循环体

每次执行完循环体后求解



例1_3：判断一个数是否为质数

```
//z1_3.cpp
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    int a;
    cout << "请输入一大于1的正整数：";
    cin >> a;
    if (a < 2) {
      cout << "输入的数字不是大于1的正整数" << endl;
      return 0;
    }
    if (a == 2) {
          cout << a << "是质数" << endl;
          return 0;
    }
    double s = sqrt(1.0 *a);
     for(int div = 2; div <= s; div ++)
       if (a % div == 0)
       {
          cout << a << "不是质数" << endl;
          return 0;
        }
    cout << a << "是质数" << endl;
    return 0;
}
```

思考：为什么a采用int而不是unsigned int？

例1_4：求正整数的质因子

设计程序，输入为一个正整数，给出这个整数的质因子分解，如：
60 = 2 * 2 * 3 * 5

思考：需要判断每个因子是否质数吗？

例1_4：求正整数的质因子

```
//z1_4.cpp
#include <iostream>
using namespace std;

int main()
{
    int n;
    cout << "输入要分解的整数：";
    cin >> n;
    if (n < 1) {
        cout << "输入不是正整数，程序退出" << endl;
        return 0;
    }
    int element = 2;
    while (n > 1 && element <= n) {
        if (n % element == 0) {
            if (n == element)
                cout << element;
            else
                cout << element << " * ";
            n /= element; // n=n/ element;
        }
        else
            element ++;
    }
    cout << endl;
    return 0;
}
```

例1_5：寻找并输出11~999之间的数m，它满足m、m^2和m^3均为回文数

- 回文：各位数字左右对称的整数。
  例如：11满足上述条件
  $11^2=121$，$11^3=1331$。
- 分析：
  - 用除以10取余的方法，从最低位开始，依次取出该数的各位数字。按反序重新构成新的数，比较与原数是否相等，若相等，则原数为回文。

例1_5

```
//z1_5.cpp
#include <iostream>
using namespace std;
//判断n是否为回文数
bool symm(unsigned n) {
  unsigned i = n;
   unsigned m = 0;
   while (i > 0) {
     m = m * 10 + i % 10;
     i /= 10;
  }
  return m == n;
}
```

例1_5

```
int main() {
   for(unsigned m = 11; m < 1000; m++)
     if (symm(m) && symm(m * m) && symm(m * m * m)) {
       cout << "m = " << m;
       cout << "  m * m = " << m * m;
       cout << "  m * m * m = "
            << m * m * m << endl;
     }
   return 0;
}
```

例1_5

运行结果：

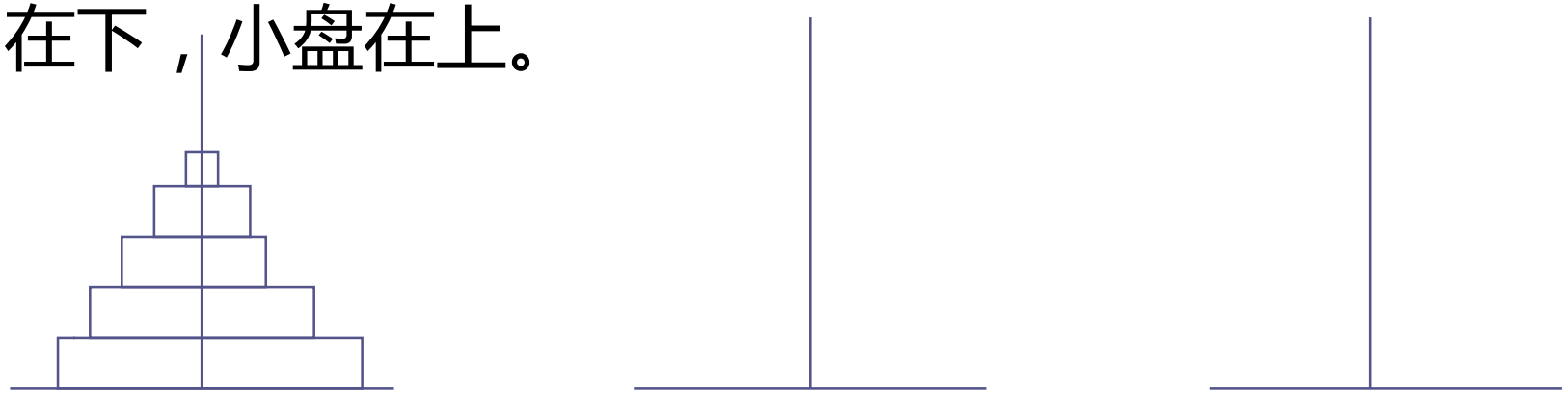
m=11 m*m=121 m*m*m=1331

m=101 m*m=10201 m*m*m=1030301

m=111 m*m=12321 m*m*m=1367631

例1_6：递归过程的跟踪——以汉诺塔问题为例

- 有三根针A、B、C。A针上有N个盘子，大的在下，小的在上，要求把这N个盘子从A针移到C针，在移动过程中可以借助B针，每次只允许移动一个盘，且在移动过程中在三根针上都保持大盘在下，小盘在上。



例1_6（续）

将n 个盘子从A针移到C针可以分解为三个步骤：

①将A 上n-1个盘子移到 B针上（借助C针）；

②把A针上剩下的一个盘子移到C针上;

③将n-1个盘子从B针移到C针上（借助A针）；

上面三个步骤包含两种操作：

①将多个盘子从一个针移到另一个针上，这是一个递归的过程。 hanoi函数实现。

②将1个盘子从一个针上移到另一针上。用move函数实现。

例1_6（续）

```
//zl_6.cpp
#include <iostream>
using namespace std;
//将src针的最上面一个盘子移动到dest针上
void move(char src, char dest) {
   cout << src << " --> " << dest << endl;
}
//将n个盘子从src针移动到dest针，以medium针作为中转
void hanoi(int n, char src, char medium, char dest)
{
  if (n == 1)
     move(src, dest);
  else {
     hanoi(n - 1, src, dest, medium);
     move(src, dest);
     hanoi(n - 1, medium, src, dest);
  }
}
```

```
int main() {
   int m;
   cout << "Enter the number of diskes: ";
   cin >> m;
   cout << "the steps to moving " << m
   << " diskes:" << endl;
   hanoi(m,'A','B','C');
   return 0;
}
```

运行结果：

```
Enter the number of diskes:3
the steps to moving 3 diskes:
A --> C
A --> B
C --> B
A --> C
B --> A
B --> C
A --> C
```

递归问题的描述与跟踪

- 汉诺塔问题的执行过程：