程序设计语言与方法(C语言)

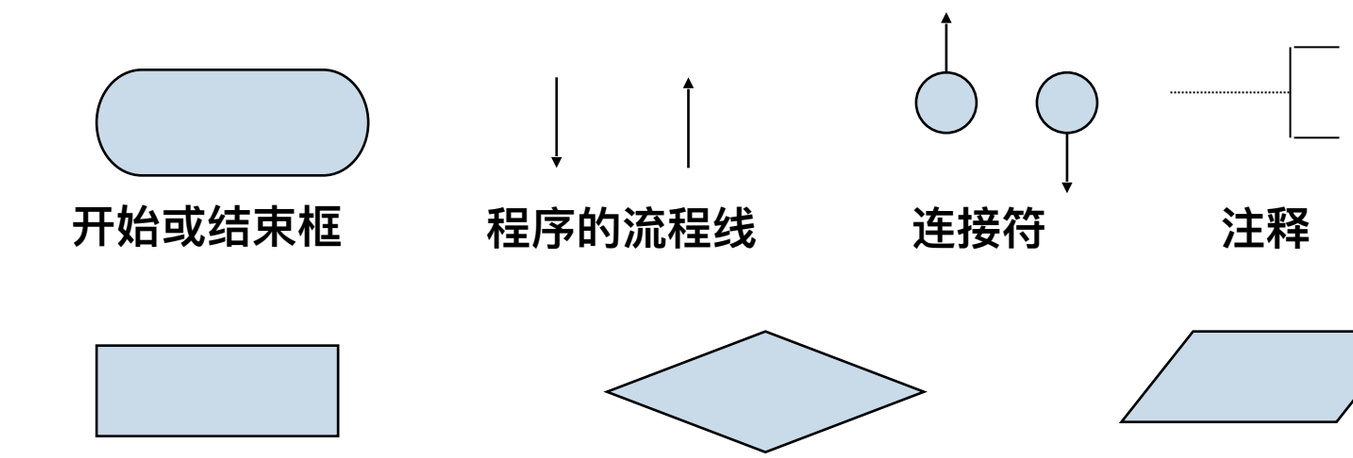
第七章 函数与结构化程序设计

程序过程的文字描述

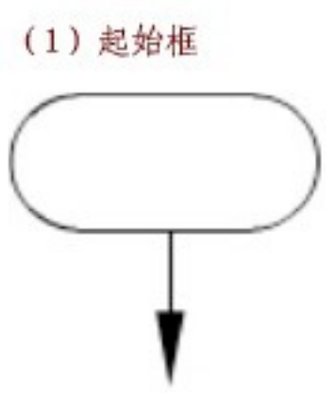
- 特点
  - 待操作的数据
  - 结果
  - 有穷性：操作步骤有限
  - 确定性：操作有确切的目标
  - 有效性：操作可以有效完成
- 问题
  - 一次成形
  - 适用于小问题
  - 大问题描述的困难

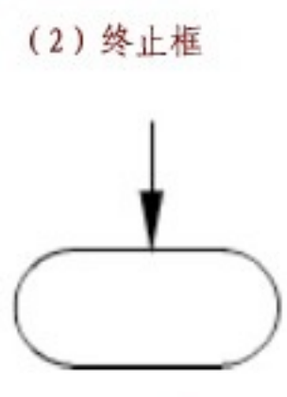
程序的图形化表示——程序流程图

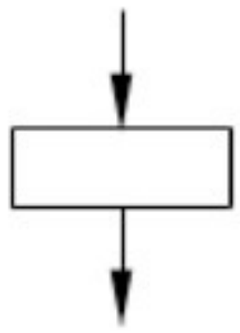
- 比文字表述更易于理解
- 图形符号

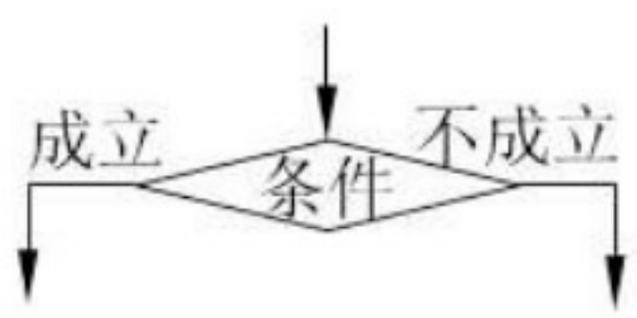


程序流程图

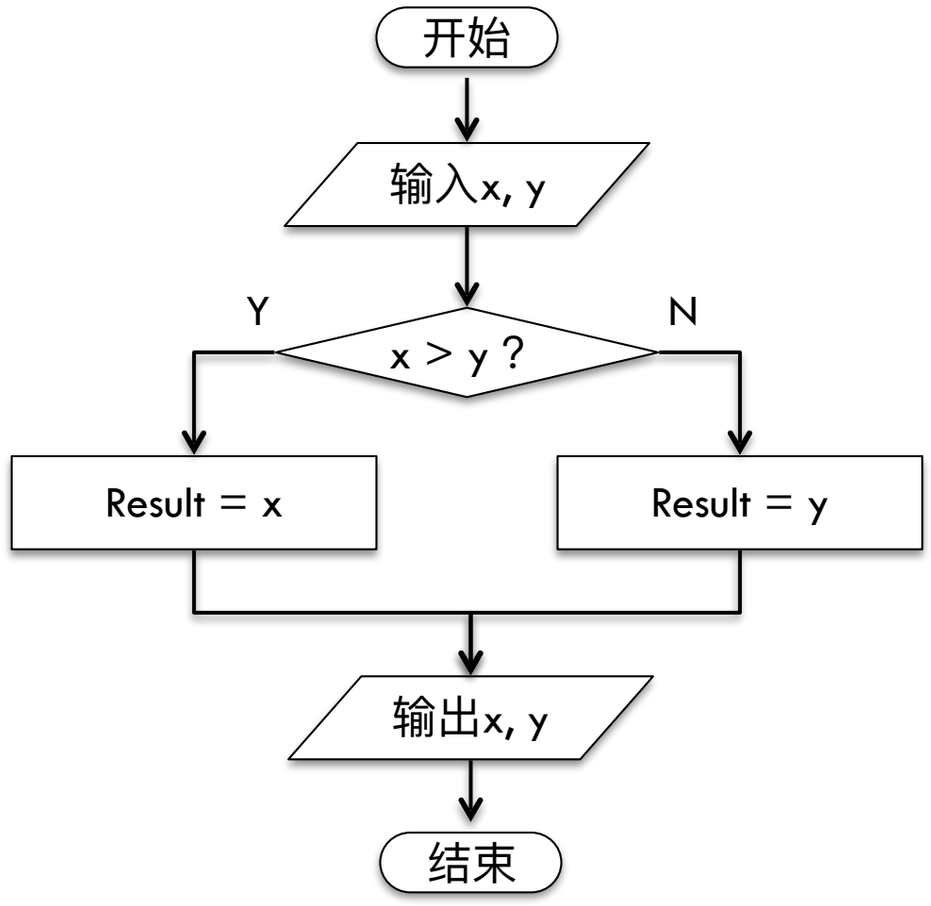






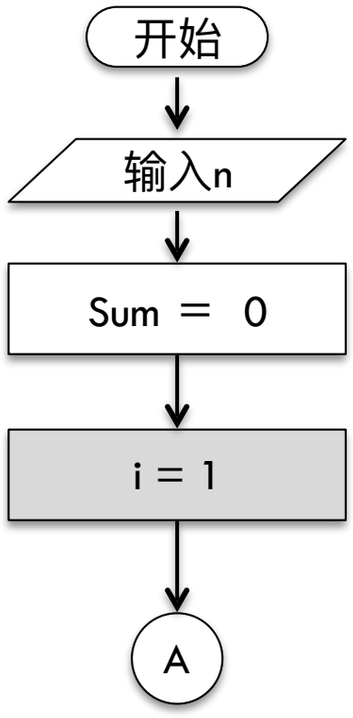


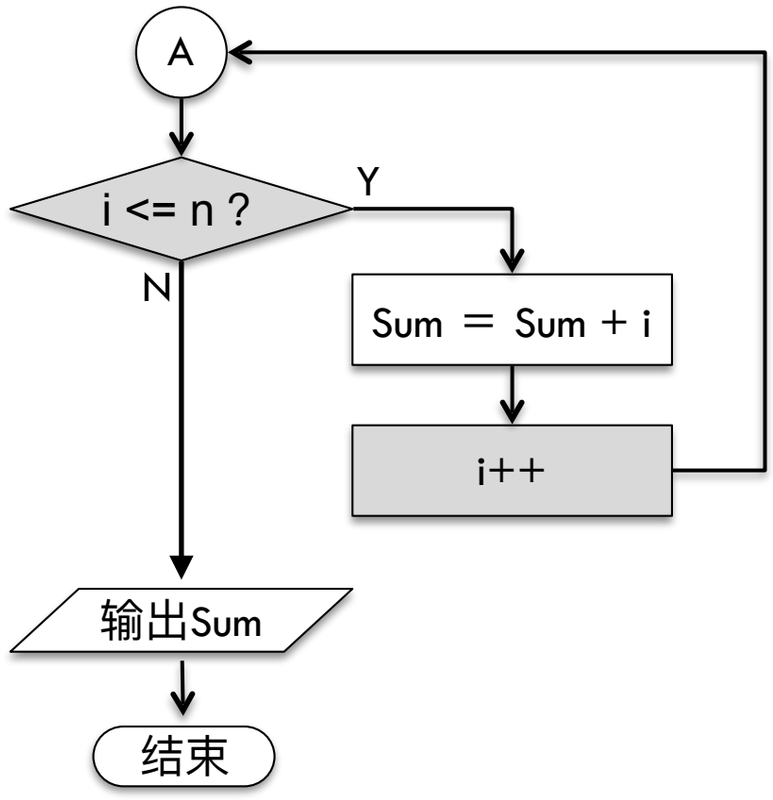
程序流程图：求两个数的较大数



程序流程图：循环表示(1..N的累加和)

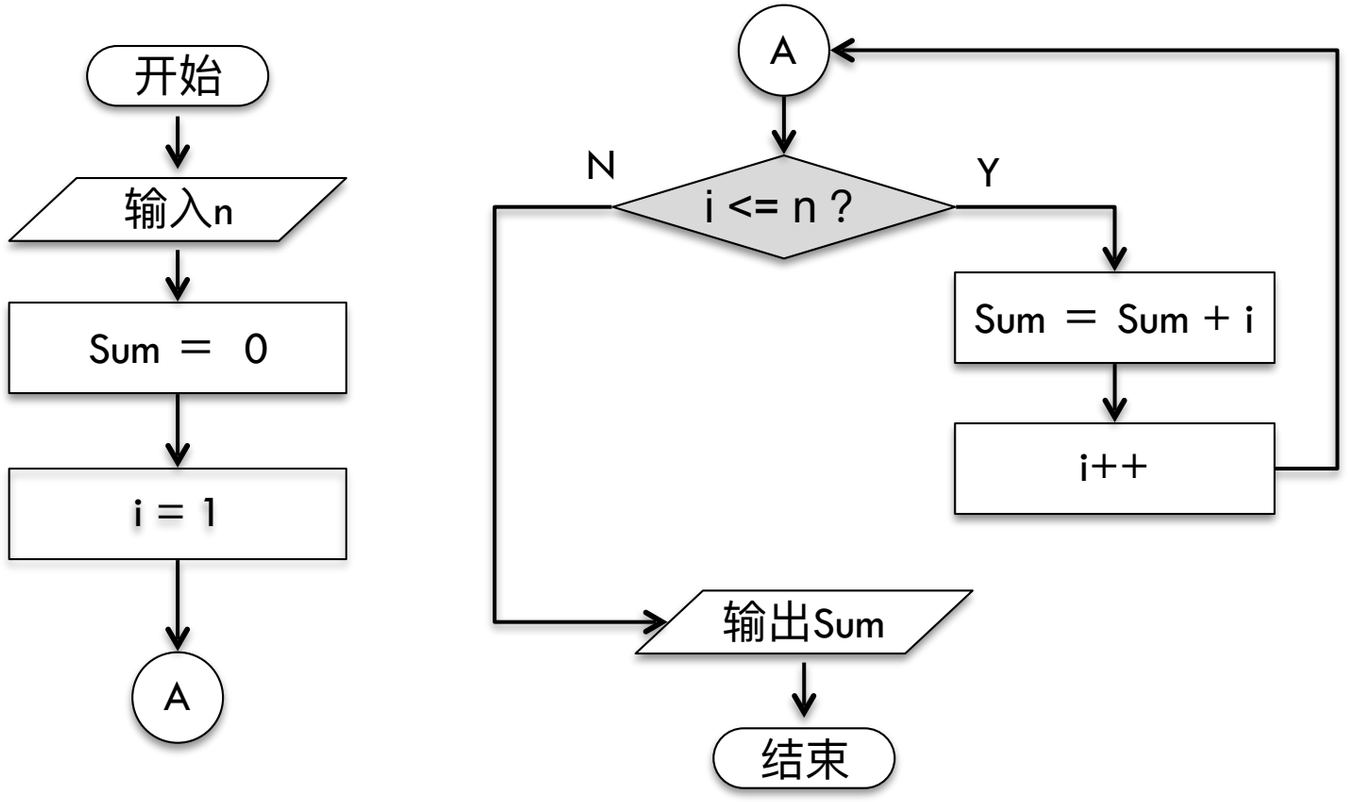
- for(i=1; i <= n; i++) {
  - Sum += i;
- }





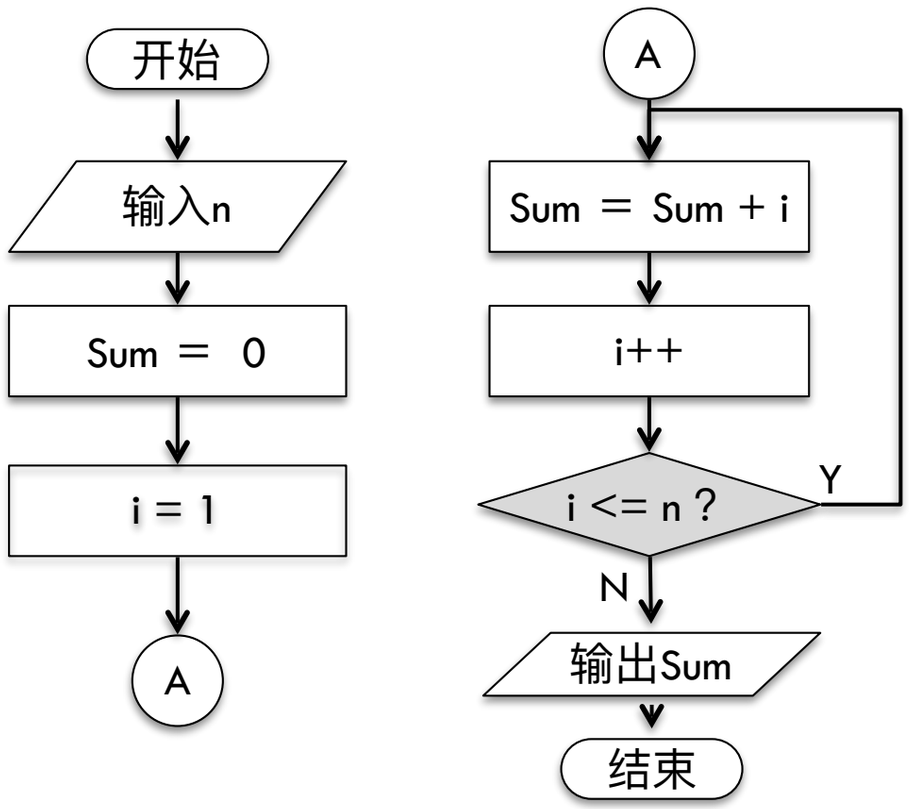
程序流程图：循环表示(1..N的累加和)

- while(i<=n) {
  - Sum += i;
  - i++;
- }



程序流程图：循环表示(1..N的累加和)

- do {
  - ……
- }while(i<=n);



伪代码（类语言的程序表述模式）

- 看起来更象是某种语言编写的程序
- 采用某种语言的语法和程序结构(通常会与你将要使用的编程语言一致)
  - 类C
  - 类Pascal
  - 类C++
  - 类Java
  - …
- 易于编写成相应语言的源程序
  - 变量赋值的表示形式
    - x←10;　//x = 10;
- 后续课程（如算法等）中会用到，建议自己学习相关处理过程的表示方法，以及整体的表示结构和书写规则

结构化程序设计

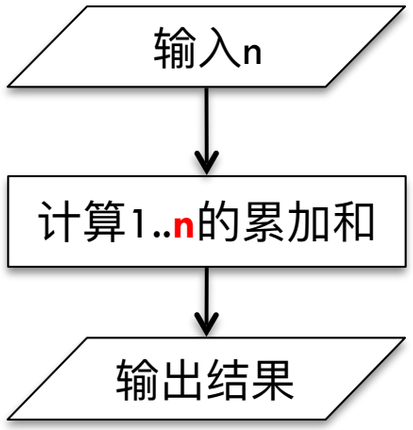
- 基本思想
  - 问题分解
- 基本方法
  - 自顶向下
  - 逐层细化
- 模块化编程

示例：求1..N的累加和

- 求1..n的累加和
  - 算法描述
    - 第1步：开始
    - 每2步：输入n　　/* scanf */
    - 每3步：计算1到n的累加和　/* ??　*/
    - 每4步：输出结果　　/* printf */
    - 每5步：结束

/*？？*/
要是能有这个就好了（简单多了）！
　　谁帮我做一个，我也就有现成的啦！
没办法，自己做吧，☹
　　该如何做呢?

示例（流程图）：求1..N的累加和

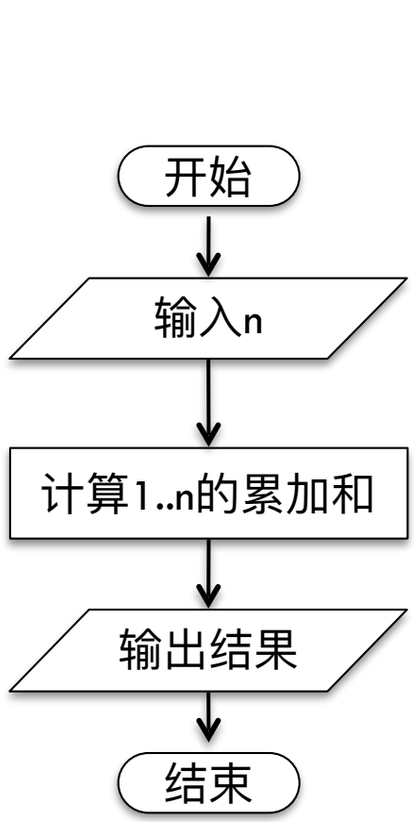


1. 是否有可以实现这个步骤的东西?
2. 我自己是否可以解决第2个问题？如果可以，如何解决?

过程描述

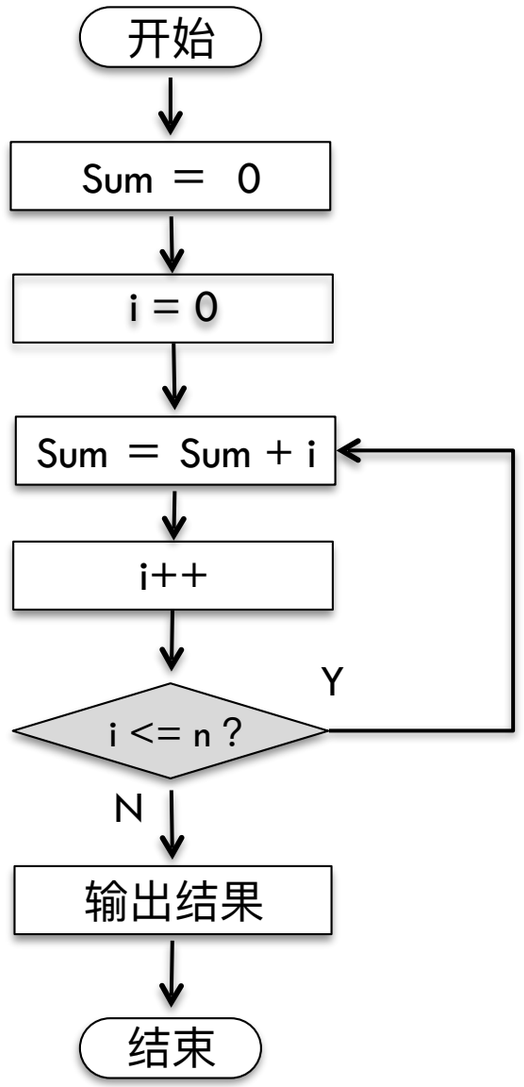
- 第3步，计算1..n的累加和
  - 迭代法的求解过程
    - 第1步：开始；
    - 第2步：result ← 0；
    - 第3步：i←1；
    - 第4步：if i > n，转第7步，否则，继续；
    - 第5步：result ← result + i；
    - 第6步：i++;
    - 第7步：输出结果；
    - 第8步：结束；

求解过程（流程图）



问题求解过程

细化



第3步求解过程

新的问题

- 计算步骤和问题的区别在哪里?
- 怎么做?
- 怎么用?

问题分析

- 问题中的n和计算步骤中的n
  - 不确定 VS 确定
  - 前者是用户输出的，后者是计算步骤中已知的
  - 过程的输入和输出
    - 输入：n； 输出：结果
- 假设有一个过程sum，可以完成计算
  - 如何将待处理的数据送给过程
  - 如何从过程获得计算结果
  - 如何在程序中使用这个过程

函数

- C语言中数据处理过程的统称
  - 表示：函数名称
  - 输入：参数列表（0..n)
  - 输出：函数的值，称为函数的返回值(0..1)
  - 使用：函数调用
    - 函数名称(参数列表);
    - 参数列表 ＝ 参数1[,参数2[,参数3[……]]]
- 标准库函数
  - scanf, printf, puts, getch, system, ……
- main
- 自定义函数

函数示例

- printf
- double fabs(double x); //求x的绝对值
  - 函数名：fabs；<math.h>
  - 形式参数：double x，接收一个双精度实数
  - 返回值的类型：double，一个双精度实数（正数）
  - 函数调用
    - fabs(y);
      - y是一个类型可转换为double的表达式
      - 实际运行时，参与计算的是表达式的值
      - 可能会有隐式类型转换，将表达式的值转换为double

函数调用示例

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

int main()
{
   double y;
   double x = -12.23;

   printf("%lf\n", fabs(x));  //double fabs(double x);
   y = fabs(x);
   printf("%lf\n", y);

   y = 1 + fabs(x * 5 + 2) / 4;
   printf("%lf, %lf\n", y, 1 + fabs(x * 5 + 2) / 4);

   return 0;
}
```

函数：1..N的累加和

- 有计算过程时
  - res = sum(n); /* 得到计算结果 */
- 自己做
  - 我要做一个sum过程可以完成1..n的累加和的计算
  - 输入(参数）/输出（返回值）及它们的表示
    - n， 累加和
    - 为输出/输出选择合适的数据类型：int, long
    - 限制说明，能做什么，不能做什么
- long sum(int n)
  - /* 能做的和不能做的 */
- 如何让函数输出结果（不是打印到屏幕）
  - 返回一个值：使用return 语句

函数声明和定义

- 函数原型
  - long sum(int n);
- 函数定义
  - long sum(int n)
  - {
  - int i;
  - long res = 0;
  - for(i = 1; i <= n; i++) {
  - res += i;
  - }
  - return res;
  - }

应用：函数调用

实参与形参一一对应
形式参数和实际参数的名称可以不同

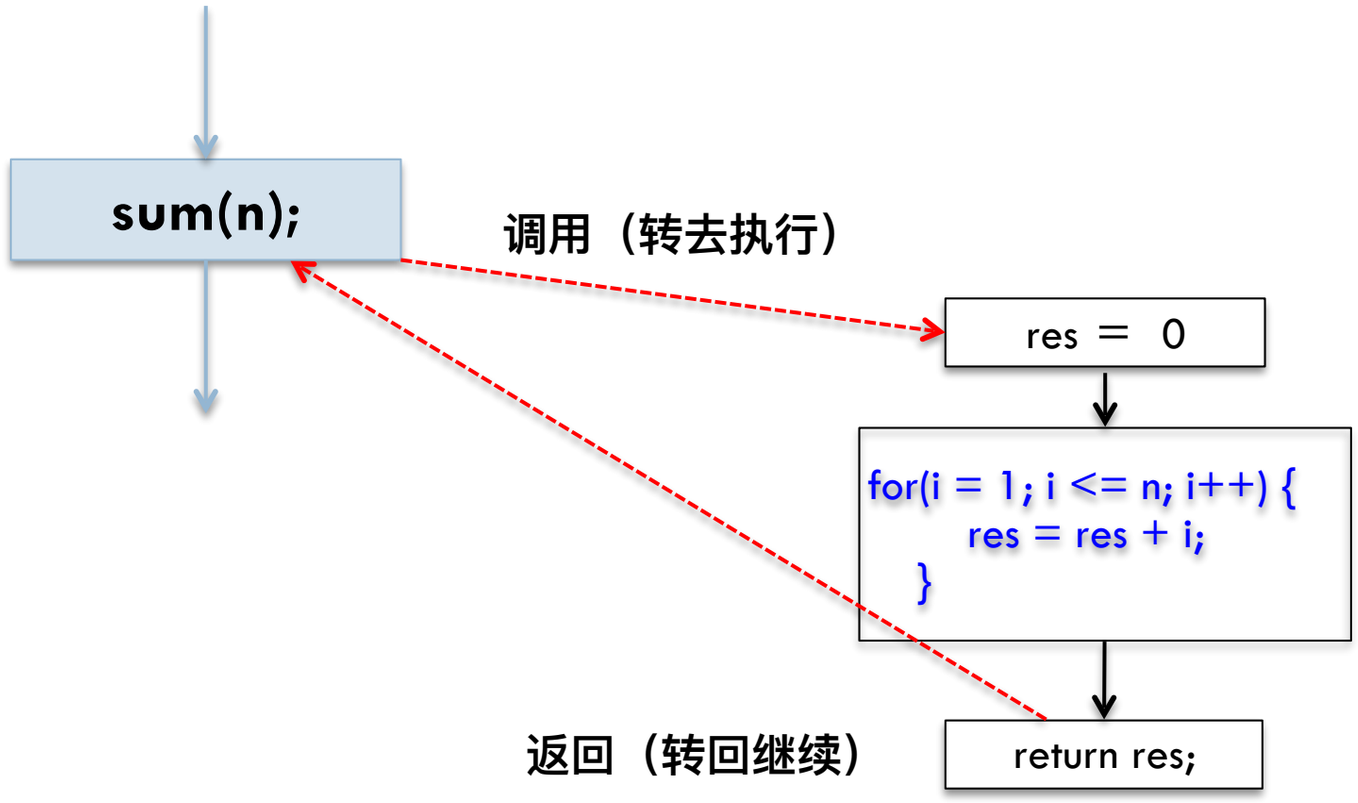
```
#include <stdio.h>

long sum(int n)  /*形式参数，声明与定义*/
{
    int i;
    long res = 0;
    for(i = 1; i <= n; i++) {
        res += i;
    }
    return res;
}

int main()
{
    int n;
    long r;
    printf("input a integer: ");
    scanf("%d", &n);
    r = sum(n);  /* 实际参数 */
    printf("1+...+%d = %d\r\n", n, r);
}
```

函数体

```
#include <stdio.h>

long sum(int m);          /*形式参数， 仅声明函数*/

int main()
{
    int n;
    long r;
    printf("input a integer: ");
    scanf("%d", &n);
    r = sum(n);           /* 实际参数 */
    printf("1+...+%d = %d\r\n", n, r);
}

long sum(int m)           /*形式参数，函数定义*/
{
    int i;
    long res = 0;
    for(i = 1; i <= n; i++) {
        res = res + i;
    }
    return res;
}
```

函数体

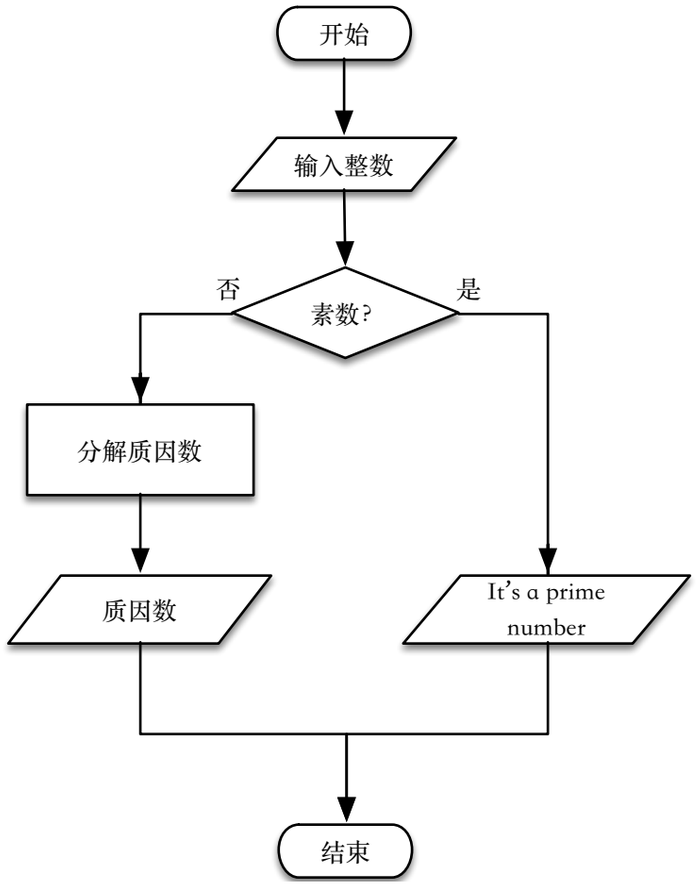
函数的执行过程

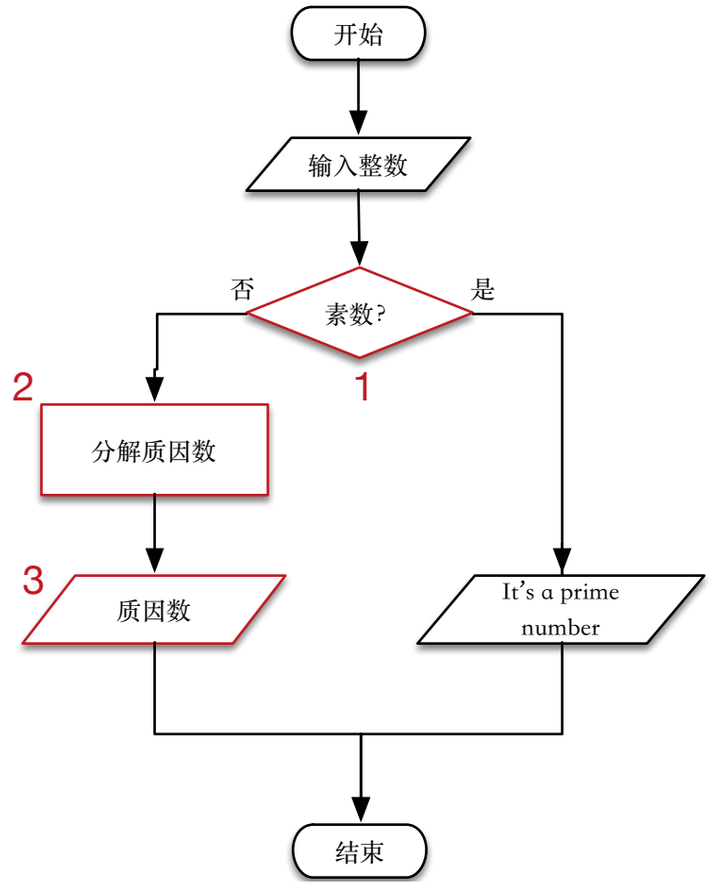


示例2：整数分解

- 对用户给定的整数进行质因数分解
  - 如果输入的整数是素数
    - 输出“It is a prime number.”
  - 如果输入的整数不是素数
    - 对其进行质因数分解
    - 并将质因数从小到大顺序排列的乘积形式输出
- 允许用户多次输入
  - 输入非0的整数，表示结束程序运行

整数分解流程

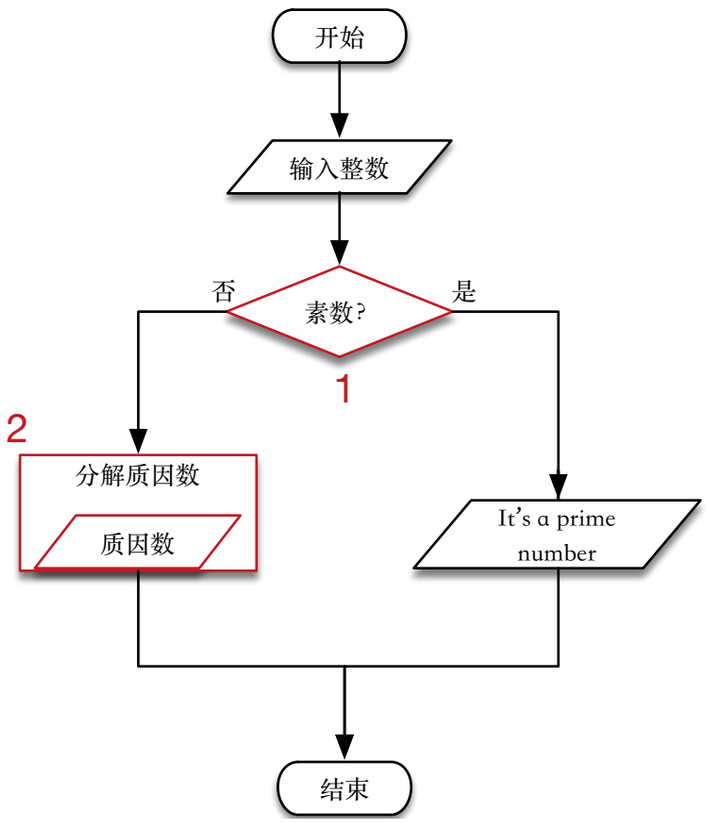

开始
输入整数
否
素数?
是
分解质因数
质因数
It's a prime number
结束

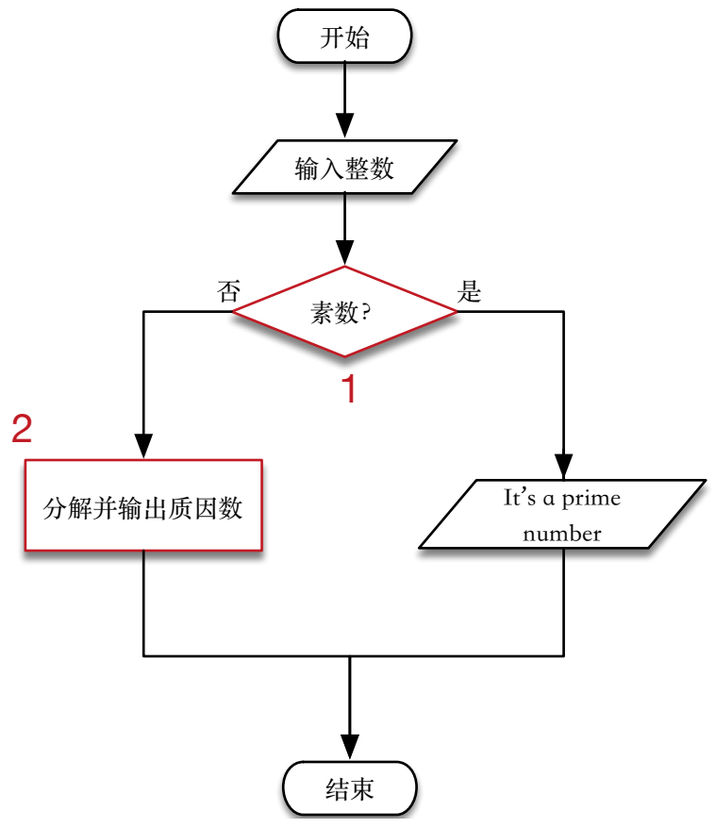

开始
输入整数
否
素数?
是
1
2
分解质因数
3
质因数
It's a prime number
结束

整数分解

```
int main()
{
    int x;
    scanf("%d", &x);
    if(isPrime(x)) {                          /* x是素数吗? */
        printf("It is a prime number.\n”);    /* 是 */
    } else {
        get_factors(x);                       /* 不是 */
        print_factors();
    }
    return 0;
}
```

整数分解


开始
输入整数
否
素数?
是
1
2
分解质因数
质因数
It's a prime number
结束


开始
输入整数
否
素数?
是
1
2
分解并输出质因数
It's a prime number
结束

整数分解

```
int main()
{
    int x;
    scanf("%d", &x);
    if(isPrime(x)) {
        printf("It is a prime number.\n");
    } else {
        get_factors(x);
        print_factors();
    }
    return 0;
}
```

```
#include <stdio.h>

int isPrime(int x);  /*  1 */
void print_factors(int x); /*  2  */

int main()
{
    int x;
    scanf("%d", &x);
    if(isPrime(x)) {
        printf("It is a prime number.\n");
    } else {
        print_factors(x);
    }
    return 0;
}
```

素数判定

- 决定**给定的整数**是**不是**素数
  - 给定整数　=〉输入
  - 判定结果：是 I 不是　=〉输出
- 函数原型
  - 函数名：isPrime
  - 参数：整数，int x
  - 返回值：布尔值，int
  - int isPrime(int x);

素数判定（函数定义）

```
int isPrime(int x)
{
        int i;
        int r = 1;
        for(i = 2; i < x / 2; i++) {
                if(x % i == 0) {
                        r = 0;
                        break;
                }
        }
        return r;
}
```

```
int isPrime(int x)
{
        int i = 2;
        while((x % i != 0) && (i < x / 2)) {
                i++;
        }
        return !(i < x / 2);
}
```

条条道路通罗马！！

分解质因数

- 分解**给定的整数**的质因数
  - 给定整数　=〉输入
  - 直接打印在屏幕上 =〉输出呢?
- 函数原型
  - 函数名：PrintFactors
  - 参数：整数，int x
  - 返回值：无，void，因为直接打印屏幕，所以无返回值
  - void print_factors(int x);

分解质因数

```
void print_factors(int x)
{
    int i = 2;
    printf("%d = ", x);
    while(i <= x) {
        if(x % i == 0) {
            x = x / i;
            printf("*%d", i);
            continue;
        }
        i++;
    }
}
```

函数设计的基本原则

- 模块化
- 信息隐藏
- 重用
- 防御性
- 函数设计的基本原则
  - 功能单一，规模要小（不多于50行）
  - 一个入口和一个出口
  - 错误控制
  - void, 无返回值
  - ……

问题：求N的阶乘

- 求n!
  - n!　＝ n * (n–1)!
  - (n–1)! = (n–1) * (n–2)!
  - ……
  - 2! = 2 * 1!
  - 1! = 1
- 假设有一计算n!的函数fact(n)
  - 上面的过程如何完成?
  - 如果我自己做函数fact(n)
  - 如何完成

分析与过程描述

- 分析
  - if(n == 1)
    - 结果等于1;
  - 否则
    - 结果等于n * (n–1)的阶乘
    - 函数声明
      - long fact(int n);
    - 函数定义
      - 合适吗?

```
long fact(int n)
{
   if(n == 1) {
      return 1;
   } else {
      return n * fact(n-1);
   }
}
```

递归函数

- 如果一个对象部分地由它自己组成或按它自己定义，则我们称它是递归的。
- 在函数体中调用函数自己的函数
  - 一种可根据其自身来定义或求解问题的编程技术
    - 一般情况
      - 由其自身定义的与原始问题类似的更小规模的子问题，它使递归过程持续进行
    - 基线情况
      - 递归调用的最简形式，它是一个能够用来结束递归调用过程的条件
- 动手完成求n!的程序

用递归法求解问题

- 1. 求1..n的累加和
- 2. 求Fibonacci数列
- 3. 汉诺塔问题

变量的使用域与存储类型

- 作用域
  - 全局变量
  - 局部变量
- 存储类型
  - 自动变量
  - 静态变量
  - 外部变量
  - 寄存器变量