C程序设计语言

第3章 控制流

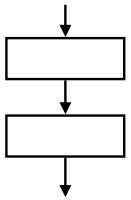
孙志岗
sun@hit.edu.cn
http://sunner.cn

三种基本结构

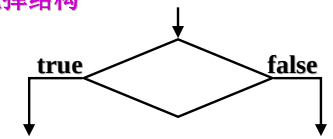
- 顺序结构、选择结构、循环结构
- 已经证明，任何程序均可只用这三种结构实现
  - BÖhm, Corrado, and Jacopini Guiseppe. "Flow diagrams, Turing machines and languages with only two formation rules." *Communication of ACM*, 9(5):366-371, May 1966.
- 只用这三种结构的程序，叫结构化程序
- 程序“必须”符合结构化规则

流程图

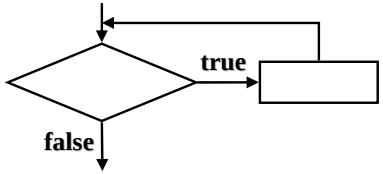
顺序结构



选择结构



循环结构

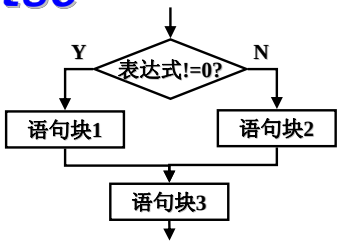


语句块（Block）

- {}括住的若干条语句构成一个语句块
- 语句块内可以定义变量
  - 变量必须在语句块的开头定义
  - 变量仅在定义它的语句块内（包括下层语句块）有效(scope.c)
  - 同一个语句块内的变量不可同名，不同语句块可以同名(homonym.c)
    - 各司其职、下层优先
    - 尽量不要在下层语句块内定义变量，也尽量不要定义同名变量
- 语句块可以用在任何可以使用语句的地方，但没有道理要乱加语句块

if-else

- 选择结构的一种最常用形式
- if (表达式)
  语句块1;
  else
  语句块2;
  语句块3

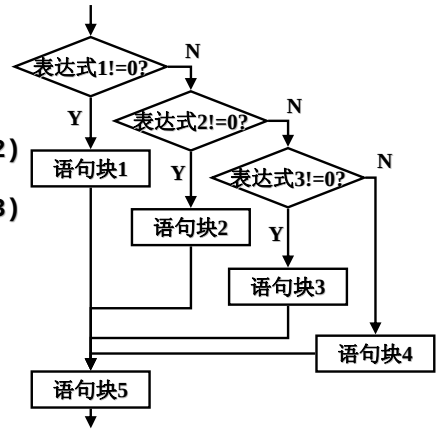


- 表达式值非0时，执行语句块1，然后语句块3；
  表达式值为0时，执行语句块2，然后语句块3
- else部分可以没有。当表达式值为0时，直接执行语句3
- if-else嵌套使用时，注意else和谁配套的问题

if.c

else-if

- if的一种扩展
- if (表达式1)
  语句块1;
  else if (表达式2)
  语句块2;
  else if (表达式3)
  语句块3;
  ……
  else
  语句块4;
  语句块5;
- else部分可以没有



switch

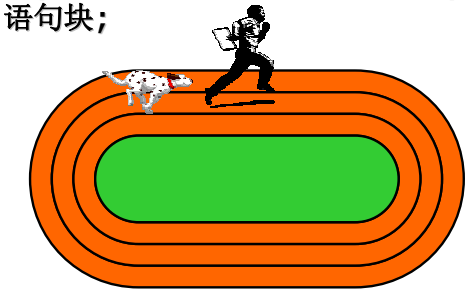
- 多路选择
- switch (表达式) {
  case 整型常数1:
  语句1;
  case 整型常数2:
  语句2;
  ……
  default:
  语句3;
  }
- default可以没有
- 现场编程完成计算器……
- 不要忘记break

switch和else-if的比较

- else-if比switch的条件控制更强大一些
  - else-if可以依照各种逻辑运算的结果进行流程控制
  - switch只能进行==判断，并且只能是整数判断
- switch比else-if更清晰
- 两者都要尽量避免用得过多、过长，尤其不要嵌套得太多
  - 它们大大增加程序的分支，使逻辑关系显得混乱，不易维护，易出错

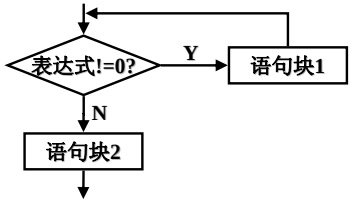
循环——while，for

- while (表达式)
  语句块;
- for (表达式1; 表达式2; 表达式3)
  语句块;



while

- while (表达式)
  语句块1;
  语句块2;
- 只要表达式的值为非0，就重复执行语句块1，直到表达式值为0时止，开始执行语句块2

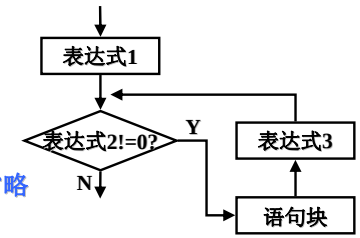


for

- for (表达式1; 表达式2; 表达式3)
  语句块;
- 首先执行表达式1。如果表达式2的值为非0，就重复执行语句块和表达式3，直到表达式2的值为0时止
- 相当于：

```
表达式1;
while (表达式2){
   语句块;
   表达式3;
}
```

- for的所有表达式均可省略

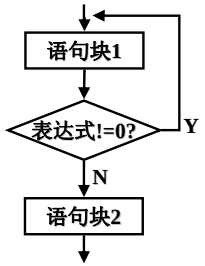


注意

- 在for和while语句之后一般没有分号
  - 有分号表示循环体就是分号之前的内容，即循环体不存在
- while (i < 100);
  i++;
- for (i = 0; i < 100; i++);
  printf("%d", i);
- for通常有一个循环变量控制循环的次数，不要在循环体内改变这个变量

循环——do-while

- do
  语句块1;
  while (表达式);
  语句块2;
- 首先执行语句，然后判断表达式的值。如果表达式为0，继续向下执行，否则，再次执行语句，再次判断表达式的值
- 语句块1会被执行至少一次



选择三种循环的一般思路

- 如果循环次数已知，用for
- 如果循环次数未知，用while
- 如果循环体至少要执行一次，用do-while
- 只是思路，不是定律

死循环

- 永远不会退出的循环为死循环
  - for (;;) {}
  - while (1) {}
  - do {} while (1)
  - 除非确实需要死循环，否则不要使用这样的形式。它们使循环的中止条件变得不明朗
- 一般情况下，要极力避免死循环
  - 绝大多数程序不需要死循环。如果出现，往往都是bug
  - 时间过长的循环会造成“假死”现象，也要考虑解决

break和continue

- 对for、while、do-while循环进行内部手术
- break，退出循环
- continue，中断此次循环的执行，开始下一次
- break和continue少用为妙
  - 它们增加了循环执行的分支，break更增加了循环的出口
  - 它们可以用来处理程序异常，而尽量不要用来处理正常流程

goto与标号（label）

- 标号举例
  - Error:
  - 同变量、函数的命名规则一样，后面加上一个冒号，一般顶格书写
- goto举例
  - goto Error;

Dijkstra与goto

- Edsger W. Dijkstra，生于1930年，卒于2002年8月6日
- 软件体系结构，最短路径算法，PV原语，结构化程序设计，向量，堆栈……大师赐予我们许多深邃的简单

Dijkstra与goto

- Edsger W. Dijkstra.
  "Letters to the editor: Go to statement considered harmful."
  Communication of ACM, 11(3):147–148, March 1968
- *"Goto considered harmful"*，Dijkstra在1968年就告诉了我们
  - "I became convinced that the go to statement should be abolished from all "higher level" programming languages."
  - "The go to statement … is too much an invitation to make a mess of one's program. "
- 现代观点认为，混乱根源不在goto，而在标号
- 任何程序都可以不用goto就实现其功能
- 但在某些情况下，使用goto可以让程序更清晰

糟糕的goto

```
START_LOOP:
if (fStatusOk) {
    if (fDataAvaiable) {
        i = 10;
        goto MID_LOOP;
    } else {
        goto END_LOOP;
    }
} else {
    for (i = 0; i < 100; i++) {
MID_LOOP:
        // lots of code here
        … …
    }
    goto START_LOOP;
}
END_LOOP:
```

糟糕的goto

```
START_LOOP:
if (fStatusOk) {
    if (fDataAvaiable) {
        i = 10;
        goto MID_LOOP;
    } else {
        goto END_LOOP;
    }
} else {
    for (i = 0; i < 100; i++) {
MID_LOOP:
        // lots of code here
        … …
    }
    goto START_LOOP;
}
END_LOOP:
```


这个代码怎么样？

```
HRESULT Init()
{
  pszMyName =
   (CHAR*)malloc(256);
  if (pszMyName == NULL)
  {
    return hr;
  }
  pszHerName =
   (CHAR*)malloc(256);
  if (pszHerName == NULL)
  {
    free(pszMyName);
    return hr;
  }
  pszHisName =
   (CHAR*)malloc(256);
  if (pszHisName == NULL)
  {
    free(pszMyName);
    free(pszHerName);
    return hr;
  }

  ……

  free(pszMyName);
  free(pszHerName);
  free(pszHisName);
  return hr;
}
```


用goto修改之后

```
HRESULT Init()
{
  pszMyName
   =(CHAR*)malloc(…);
  if (pszMyName == NULL)
    goto Exit;
  pszHeName
   =(CHAR*)malloc(…);
  if (pszHeName == NULL)
    goto Exit;
  pszHiName
   =(CHAR*)malloc(…);
  if (pszHiName == NULL)
    goto Exit;

  … …
Exit:
  if (pszMyName)
    free(pszMyName);
  if (pszHeName)
    free(pszHeName);
  if (pszHiName)
    free(pszHiName);

  return hr;
}
```


使用goto的原则

- 使用之后，程序仍然是单入口，单出口
- 不要使用一个以上的标号
- 不要用**goto**往回走，要向下走
- 不要让**goto**制造出永远不会被执行的代码

Dijkstra说过的话

- 编程的艺术就是处理复杂性的艺术
- 优秀的程序员很清楚自己的能力是有限的，所以他对待编程任务的态度是完全谦卑的，特别是，他们会象逃避瘟疫那样逃避“聪明的技巧”。——1972年图灵奖演讲
- 简单是可靠的先决条件
- 我们所使用的工具深刻地影响我们的思考习惯，从而也影响了我们的思考能力
- 实际上如果一个程序员先学了BASIC，那就很难教会他好的编程技术了：作为一个可能的程序员，他们的神经已经错乱了，而且无法康复
- 就语言的使用问题：根本不可能用一把钝斧子削好铅笔，而换成十把钝斧子会使事情变成大灾难