基于 Linux 的 C++

第二讲　程序流程控制

提　纲

结构化程序设计基础

布尔数据

if 分支结构

switch 分支结构

while 循环结构

for 循环结构

结构化程序设计基础

程序的控制结构（黑箱）

单入口单出口的控制结构易于理解

三种基本控制结构

顺序结构

分支结构

循环结构

复杂控制结构

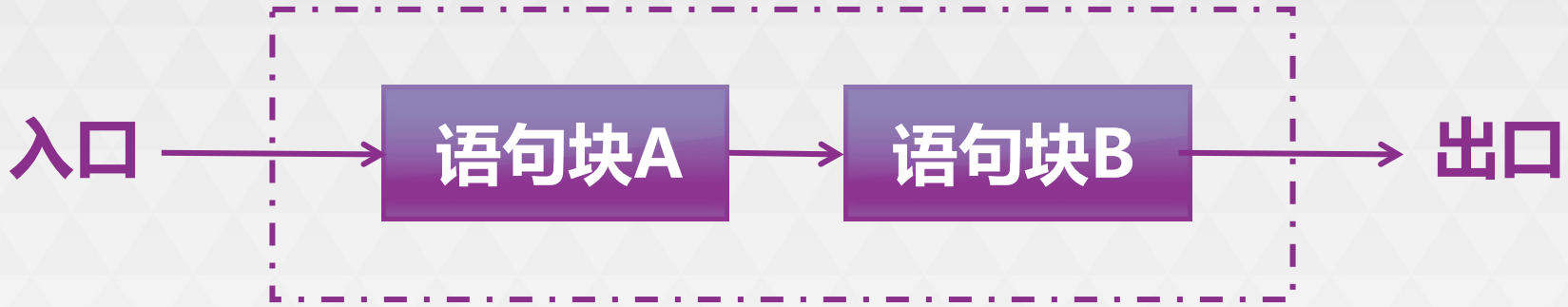
控制结构可以嵌套，以构成更复杂的控制结构

顺序结构

顺序结构

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



顺序结构示例

编写程序，接受用户输入的两个整数，输出其中较大者

```
#include <iostream>
using namespace std;
int main()
{
  int a, b, max;
  /*  输入部分  */
  cout << "The program gets two numbers and prints the greater one.\n";
  cout << "The first number: ";
  cin >> a;
  cout << "The second number: ";
  cin >> b;
  /* 计算部分  */
  max = a > b ? a : b;   /* 三元表达式 */
  /* 输出部分 */
  cout << "The greater one is " << max << "." << endl;
  return 0;
}
```

三元表达式

格　式

表达式1？表达式2：表达式3

计算过程

先计算表达式1的值，若为真，则结果为表达式2的值，否则为表达式3的值

示　例

```
x = ( a > b ) ? a : b;
```

等价于：`if( a > b )  x = a;  else  x = b;`

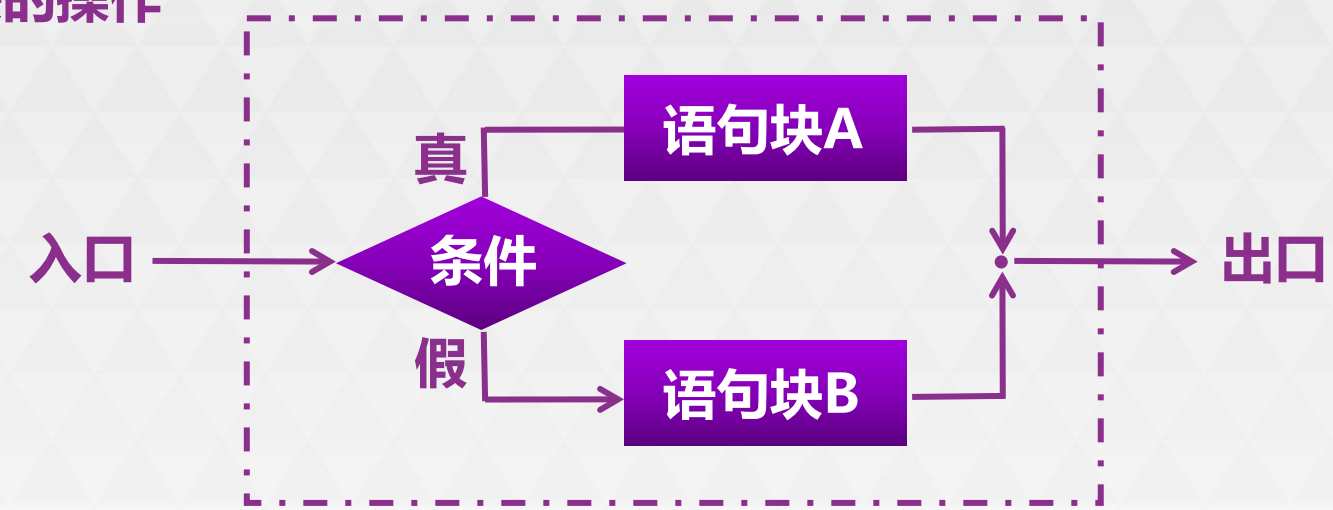
分支结构

分支结构（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

最基本的分支结构是二路分支结构

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



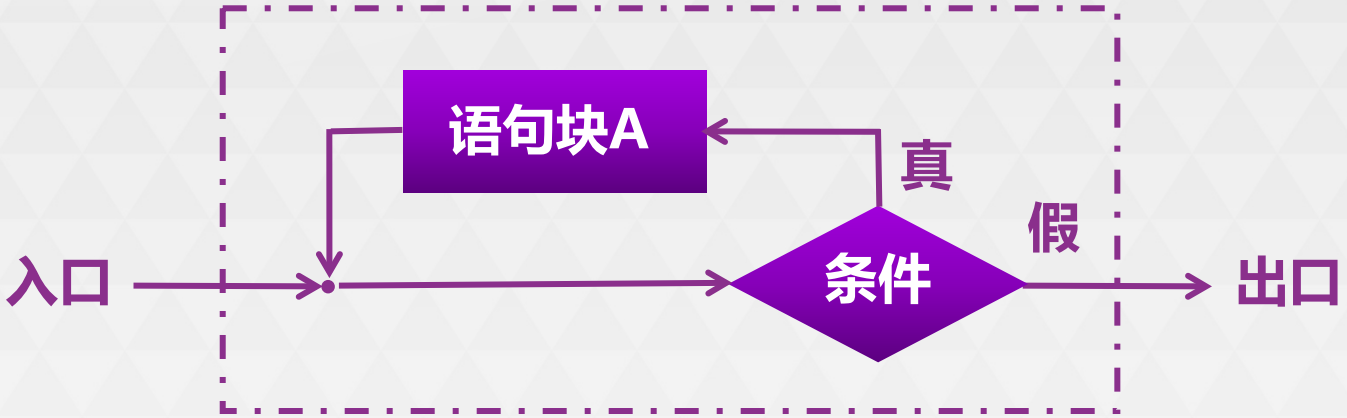
循环结构

循环结构的含义

根据某一条件的判断结果，反复执行某一处理块的过程

最基本的循环结构是当循环

进入循环结构，判断循环条件，如果循环条件的结果为真，则执行A处理块的操作，即循环一次，然后再次判断循环条件，当循环条件为假时，循环结束



布尔数据

- 枚举类型
- 用户自定义数据类型
- 关系表达式
- 逻辑表达式
- 逻辑表达式的求值

枚举类型

枚举类型的声明

格式：enum 枚举名 { 元素名1, 元素名2, …, 元素名n };

例：enum MONTH{ JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };

枚举类型声明中的元素为枚举文字，不是变量

枚举类型变量的定义

例：MONTH month;

枚举类型的意义

将多个文字组织在一起，表达从属于特定类型的性质

取代魔数，使源代码更易理解

用户自定义类型

自定义类型的格式

格式：typedef 原类型标识 新类型名;

例一：typedef int DWORD;

自定义类型的性质

新类型与原类型相同，并未产生新类型，重新命名的自定义类型使程序更易理解

若整数可以用于表示两类不同数据对象，使用自定义类型可以区分它们

自定义类型不是简单的类型替换，虽然它们确实等同

布尔类型

bool 类型

取值：false、true

bool 量的定义

定义：bool modified;

赋值： modified = true;

关系表达式

关系操作符

大于（>）、等于（==）、小于（<）、不大于（<=）、不小于（>=）、不等于（!=）

关系表达式

关系操作符与两个操作数构成的表达式

多个关系表达式可连接起来构成复杂关系表达式

运算结果为逻辑值：真或假

逻辑值（布尔值）

C/C++ 语言以 0 表示假，以非 0 表示真（经常以 1 表示）

尽量使用 bool 类型表示逻辑值

示　例

例：x == y , a < b

逻辑表达式

逻辑操作符：逻辑与（&&）、逻辑或（||）、逻辑非（!）

逻辑表达式

逻辑操作符与一个或两个操作数构成的表达式，结果仍为真或假

x && y：若 x、y 均为真，则结果为真，否则为假

x || y：若 x、y 均为假，则结果为假，否则为真

!x：若 x 为真，则结果为假，否则为真

复杂逻辑表达式

例：x > y || a != b && 3 <= 2

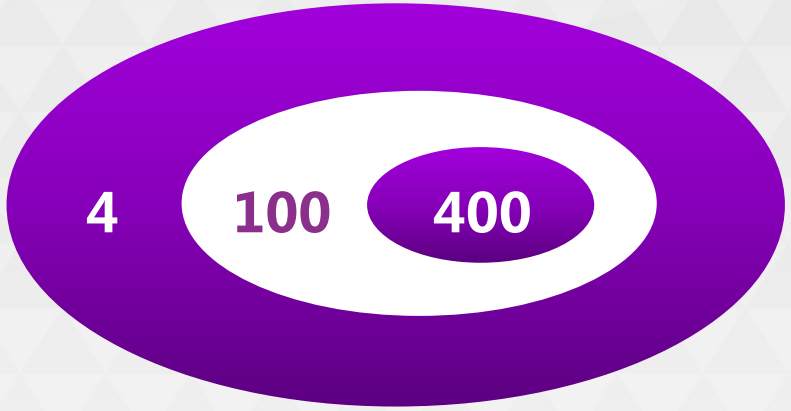
关系操作符与逻辑操作符的优先级

从高到低顺序：逻辑非“!”；小于“<”、不小于“>=”、大于“>”、不大于“<=”（同级）；等于“==”、不等于“!=”（同级）；逻辑与“&&”；逻辑或“||”

逻辑表达式的求值

给定年份 year，判断是否为闰年，闰年规定为：

- 能够被 400 整除的年份一定是闰年
- 其他能够被 100 整除的年份一定不是闰年
- 其他能够被 4 整除的年份一定是闰年




```
year % 4 == 0 && year % 100 != 0 || year % 400 == 0
```

if分支结构

If 分支结构的三种格式

格式一：if(条件表达式) { 语句序列 }

格式二：if(条件表达式) { 语句序列 1 } else { 语句序列 2 }

格式三：if(条件表达式 1) { 语句序列 1 } else if(条件表达式 2) { 语句序列 2 } ...
else { 语句序列 n }

特别说明

条件表达式必须位于括号内，一般为关系或逻辑表达式

先计算条件表达式值，若为真则执行语句序列 1，否则执行语句序列 2

语句序列 1 与语句序列 2 可以为复合语句、单语句或空语句

语句序列 1 与语句序列 2 只能有一个被执行

若仅用于确定某条语句是否执行，else 分支可以省略

简单if语句

编写程序，接受用户输入的整数，如果该整数为奇数则将其乘 3 加 1 后输出，偶数直接输出

```
#include <iostream>
using namespace std;
int main()
{
  int a, result;
  cout << "The program gets a number.\nIf it is an even, output it directly, \n";
  cout << "otherwise multiply it by 3 then plus 1.\n";
  cout << "The number: ";
  cin >> a;
  result = a;
  if( a % 2 == 1 )
    result = a * 3 + 1;
  cout << "The result is " << result << "." << endl;
  return 0;
}
```

if-else 语句

编写程序，接受用户输入的整数，如果该整数为奇数则将其乘 3 加 1 后输出，偶数除以 2 后输出

```
#include <iostream>
using namespace std;
int main()
{
  int a, result;
  cout << "The program gets a number.\nIf it is an even, divide it by 2, \n";
  cout << "otherwise multiply it by 3 then plus 1.\n";
  cout << "The number: ";
  cin >> a;
  if( a % 2 == 1 )
    result = a * 3 + 1;
  else
    result = a / 2;
  cout << "The result is " << result << "." << endl;
  return 0;
}
```

if-else if-else 语句

已知2006年12月1日为星期五。编制程序，接受用户输入的1~31之间的整数，按照下述格式将该日星期几信息打印在对应栏下。例如，2006年12月1日打印在星期五“Fr”下面：

```
Calendar 2006-12
_____________________________
Su   Mo   Tu   We   Th   Fr   Sa
_____________________________
                          1
_____________________________
```

程序代码

```
#include <iostream>
#include <iomanip>
using namespace std;
typedef enum { SUN, MON, TUE, WED, THU, FRI, SAT } WEEKDAY;
int main()
{
  int date;
  const WEEKDAY date_1 = FRI;
  WEEKDAY weekday;
  /* 输入部分 */
  cout << "The program gets a date(1~31), \n";
  cout << " and prints a calendar of 2006-12 (just the date).\n";
  cout << "The date: ";
  cin >> date;
  if( date < 1 || date > 31 )
  {  /* 日期输入错误，给出错误信息，退出执行 */
    cout << "Date error!\n";
    return 1;
  }
```

程序代码

```
/* 计算部分，得到该日的星期几信息 */
 weekday = (WEEKDAY)((date + (int)date_1 - 1) % 7);
 /* 输出部分 */
 cout << "Calendar 2006-12\n";
 cout << "------------------------\n";
 cout << "Su  Mo  Tu  We  Th  Fr  Sa\n";
 cout << "------------------------\n";
 /* 在指定位置输出该日的星期几信息 */
 if( weekday == SUN )          cout << setw(2) << date;
 else if( weekday == MON )     cout << setw(6) << date;
 else if( weekday == TUE )     cout << setw(10) << date;
 else if( weekday == WED )     cout << setw(14) << date;
 else if( weekday == THU )     cout << setw(18) << date;
 else if( weekday == FRI )     cout << setw(22) << date;
 else                          cout << setw(26) << date;
 cout << endl << "------------------------\n";
 return 0;
}
```

switch 分支结构

计算过程

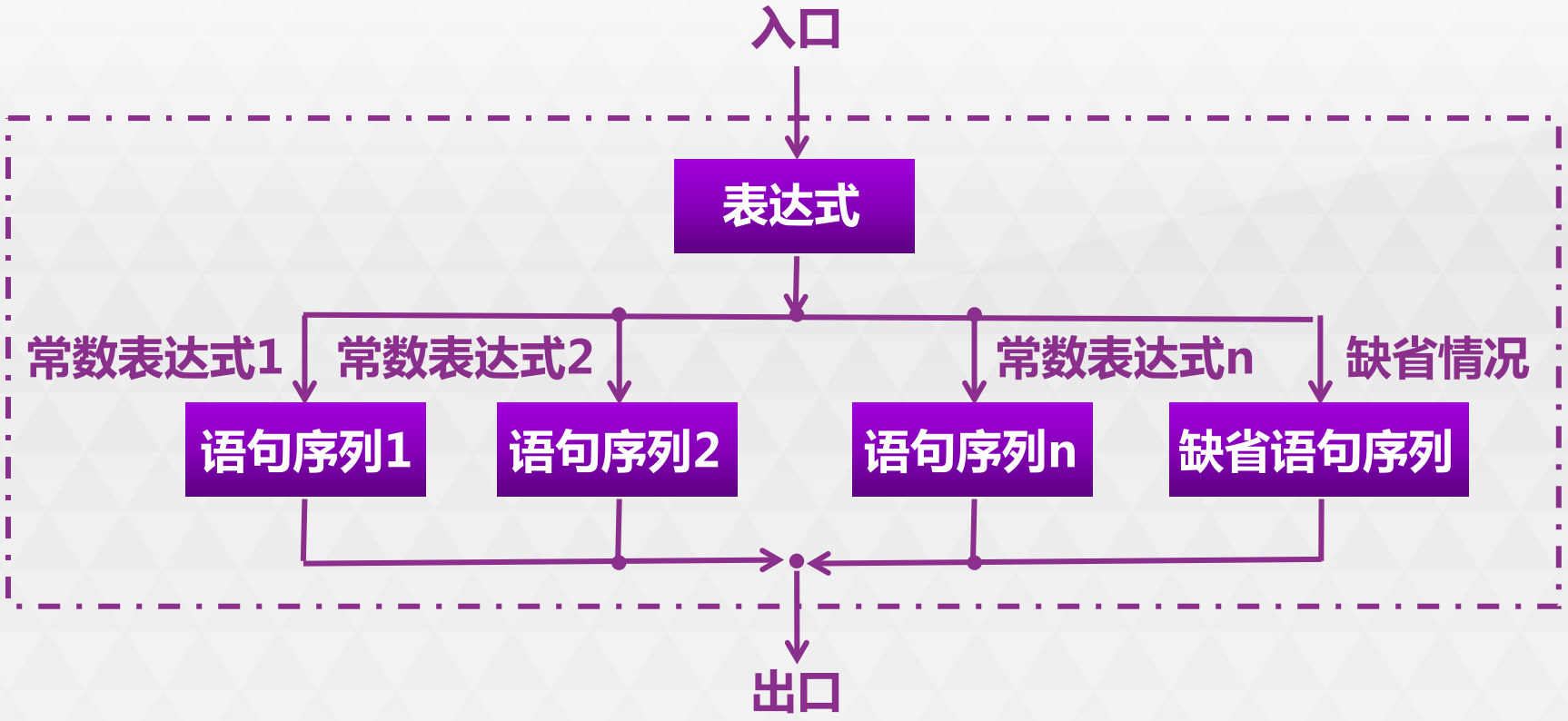
先计算表达式的值
依次与常数表达式比较
若相同则执行该分支（子句）
否则转向 default 分支
最后退出 switch 语句

```
switch( 表达式 )
{
  case 常数表达式 1:  语句序列 1
  case 常数表达式 2:  语句序列 2
  ┆
  case 常数表达式 n:  语句序列 n
  default: 默认语句序列
}
```

特别说明

switch 后面的表达式必须为整型、字符型或枚举型
case 后面必须为常量表达式，且各个 case 值必须不同
若无 default 分支，且无 case 分支匹配，则不执行
case 分支中的语句可以有多条，不需要花括号

switch 语句的执行流程



switch 示例

```
/* 输出部分 */
 cout << "Calendar 2006-12\n";
 cout << "------------------------\n";
 cout << "Su  Mo  Tu  We  Th  Fr  Sa\n";
 cout << "------------------------\n";
 /* 在指定位置输出该日的星期几信息 */
 switch( weekday ){
 case SUN:      cout << setw(2) << date;       break;
 case MON:      cout << setw(6) << date;       break;
 case TUE:      cout << setw(10) << date;      break;
 case WED:      cout << setw(14) << date;      break;
 case THU:      cout << setw(18) << date;      break;
 case FRI:      cout << setw(22) << date;      break;
 case SAT:      cout << setw(26) << date;      break;
 default:       ;    /* 没有缺省情况需要处理 */
 }
 cout << endl << "------------------------\n";
 return 0;
}
```

```
Calendar 2006-12
----------------------------
Su   Mo   Tu   We   Th   Fr   Sa
----------------------------
                         1
----------------------------
```

break 语句

break 语句的目的

- 终止 switch 语句的执行
- 如果没有 break 语句，则程序会从指定的 case 分支开始，并在该分支结束后继续执行下去
- 除非 switch 语句结束，后面的其他 case 分支或 default 分支中的语句都会得到执行
- 原因：case 子句中的常数表达式仅起到语句标号的作用，不是分支之间的分隔标记

省去break 语句的场合

- 允许多个分支执行同样的代码

分支结构的嵌套

考虑表达某企业的工资晋级计划。该计划向在公司长期服务的老员工和虽然服务年限较短但年龄偏大的员工倾斜。计划规定，若员工服务年限未达 5 年，则若年龄不小于 28 岁长一级工资；若服务年限已达 5 年，长两级工资。那些服务年限短的小字辈不再此次调整工资之列。设 age 表示员工年龄，service_years 表示服务年限，salary_level 表示工资级别。

分支结构的嵌套

else 与哪个 if 配对？

else 与 if 配对规则

离它最近：距离最短

同层次：排除底层嵌套

```
if( service_years < 5 )
  if( age >= 28 )
    salary_level += 1;
  else
    salary_level += 2;
```

降低第二个 if 的层次，使 else 与第一个 if 配对

```
if( service_years < 5 )
{
  if( age >= 28 )
    salary_level += 1;
}
else
  salary_level += 2;
```

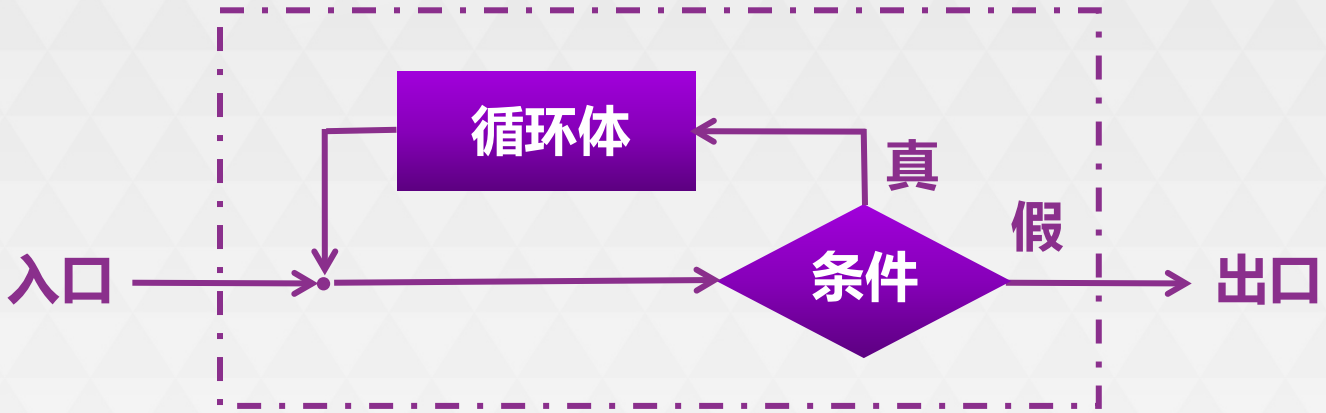
while 循环结构

while 循环格式：while（表达式）循环体

while 循环执行流程

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



while 循环示例

编写程序，接受用户输入的多个整数求和，用户输入0时程序结束

```
#include <iostream>
using namespace std;
int main()
{
  int n, sum = 0;
  cout << "The program gets some integers, and output their sum.\n";
  cout << "To stop, please input 0.\n";
  cout << "Please input an integer: ";
  cin >> n;
  while( n != 0 )
  {
    sum += n;
    cout << "The next integer: ";
    cin >> n;
  }
  cout << "The sum is " << sum << "." << endl;
  return 0;
}
```

常见的程序结构

无限循环

发生无限循环的场合

循环体内没有改变循环变量值的语句
即使改变循环变量值，也不能否定循环条件
循环体内没有能够强制终止循环执行的语句或命令

无限循环的后果

程序永远不会结束，大多数时有害，编程时一般应避免
例外情况：循环体内有专门控制循环终止的语句或命令
命令格式：满足某种条件下使用 break 语句，终止循环执行
哨兵：使循环满足终止条件的循环变量值

使用 break 语句终止循环

```
cout << "The program gets some integers, and output their sum.\n";
cout << "To stop, please input 0.\n";
while( true )
{
  cout << "Please input an integer: ";
  cin >> n;
  if( n == 0 )
    break;
  sum += n;
}
```

使用无限循环和哨兵的优势

不再需要将首个数据处理过程提到循环体前单独处理

for 循环结构

- 递增递减表达式
- for 语句
- for 与 while 的比较
- 循环嵌套

递增递减表达式

递增递减表达式的优先级非常高

前缀递增递减

格式：++变量名称; --变量名称;

例一：设 a 为 1，++a 等价于 a = a + 1，a 结果为 2

例二：设 a 为 1，--a 等价于 a = a - 1，a 结果为 0

计算要诀：先递增递减，再参与运算

例三：设 a 为 1，b = ++a * 3 等价于 a = a + 1; b = a * 3，a 结果为 2， b 结果为 6

例四：设 a 为 1，b = --a * 3 等价于 a = a - 1; b = a * 3，a 结果为 0， b 结果为 0

递增递减表达式

后缀递增递减

格式：变量名称++; 变量名称--;

例一：设 a 为 1，a++ 等价于 a = a + 1，a 结果为 2

例二：设 a 为 1，a-- 等价于 a = a - 1，a 结果为 0

计算要诀：先参与运算，再递增递减

例三：设 a 为 1，b = a++ * 3 等价于 b = a * 3; a = a + 1，a 结果为 2， b 结果为 3

例四：设 a 为 1，b = a-- * 3 等价于 b = a * 3; a = a - 1，a 结果为 0， b 结果为 3

注意事项

操作数必须为变量，而不能为其他表达式

不要在复杂表达式中使用递增递减操作符

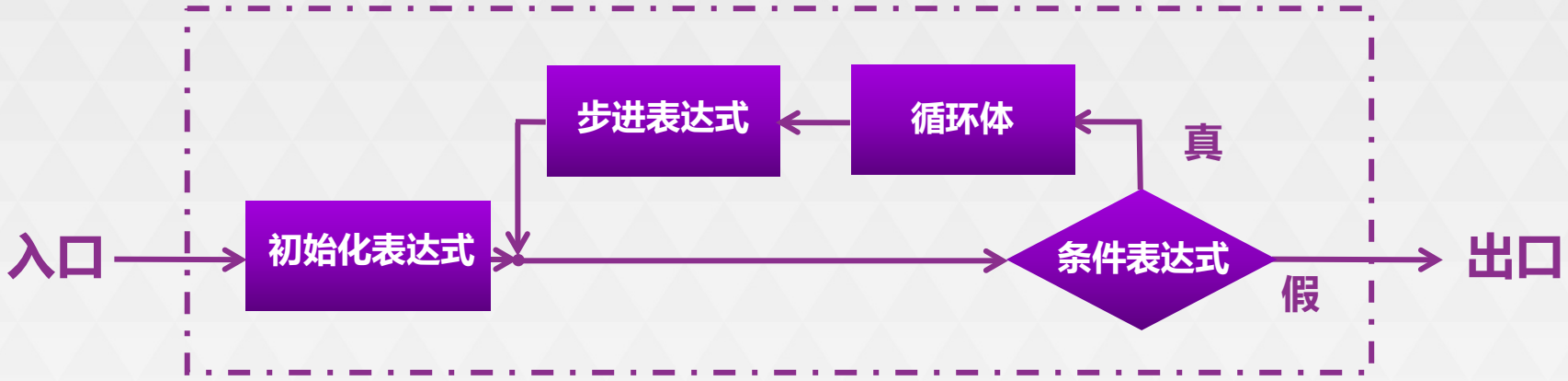
for 语句

for 循环格式

for(初始化表达式; 条件表达式; 步进表达式) 循环体

for 循环执行流程

先判断后执行：先执行初始化表达式，再计算条件表达式，并根据其结果决定是否执行一遍循环体（为真时执行），计算步进表达式的值（循环再次“初始化”），返回重新计算条件表达式的值以确定循环是否终止



for 循环示例

编写程序，接受用户输入的整数 n，求 1 ~ n 的平方和

```
#include <iostream>
using namespace std;
int main()
{
  int n, i, sum;
  cout << "The program gets a positive integer,\n";
  cout << "and prints the squared sum from 1 to the number.\n";
  cout << "The number: ";
  cin >> n;
  sum = 0;
  for( i = 1; i <= n; ++i )
    sum += i * i;
  cout << "The sum is " << sum << "." << endl;
  return 0;
}
```

for 与 while 的比较

两种循环结构可以互换使用

while 循环常用于不需要或很少需要初始化的场合

for 循环常用于需要简单初始化和通过递增递减运算控制循环体执行的场合

for 循环将所有循环控制因素都放在循环头部，循环结构最清晰

通过省略循环头部的一个或多个表达式，for 循环也可能非常复杂

循环嵌套

编写程序，按下述格式打印九九乘法表

```
   Nine-by-nine Multiplication Table
-------------------------------------
     1   2   3   4   5   6   7   8   9
-------------------------------------
 1   1   2   3   4   5   6   7   8   9
 2       4   6   8  10  12  14  16  18
 3           9  12  15  18  21  24  27
 4              16  20  24  28  32  36
 5                  25  30  35  40  45
 6                      36  42  48  54
 7                          49  56  63
 8                              64  72
 9                                  81
-------------------------------------
```

程序代码

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  int i, j;
  /*  打印表头  */
  cout << "    Nine-by-nine Multiplication Table\n";
  cout << "-----------------------------------\n";
  cout << "   ";    /*  两个空格，跳过最左列的竖排标志  */
  for( i = 1; i <= 9; ++i )    /*  横排 1～9 标志  */
    cout << setw(4) << i;
  cout << "\n";
  cout << "-----------------------------------\n";
```

程序代码

```
/*  逐行打印  */
  for( i = 1; i <= 9; ++i )    /*  共打印 9 行，i 为行号  */
  {
    cout << setw(2) << i;    /*  最左列的竖排标志  */
    /*  共打印 9 列，j 为列号，只打印上三角，下三角使用空格填充  */
    for( j = 1; j <= 9; ++j )
    {
      if( j < i )
        cout << "    ";
      else
        cout << setw(4) <<  i * j;
    }
    cout << endl;    /*  结束一行打印，换行  */
  }
  cout << "-----------------------------------\n";
  return 0;
}
```

编程实践

2.1 使用循环结构打印下述图形，打印行数n由用户输入。图中每行事实上包括两部分，中间间隔空格字符数m也由用户输入。

```
    *     *********
   ***     *******
  *****     *****
 *******     ***
*********     *
```

2.2 按照下述格式打印2016年1月日历：

```
Calendar 2016-01
---------------------------
Su  Mo  Tu  We  Th  Fr  Sa
---------------------------
                    1   2
 3   4   5   6   7   8   9
10  11  12  13  14  15  16
17  18  19  20  21  22  23
24  25  26  27  28  29  30
31
---------------------------
```