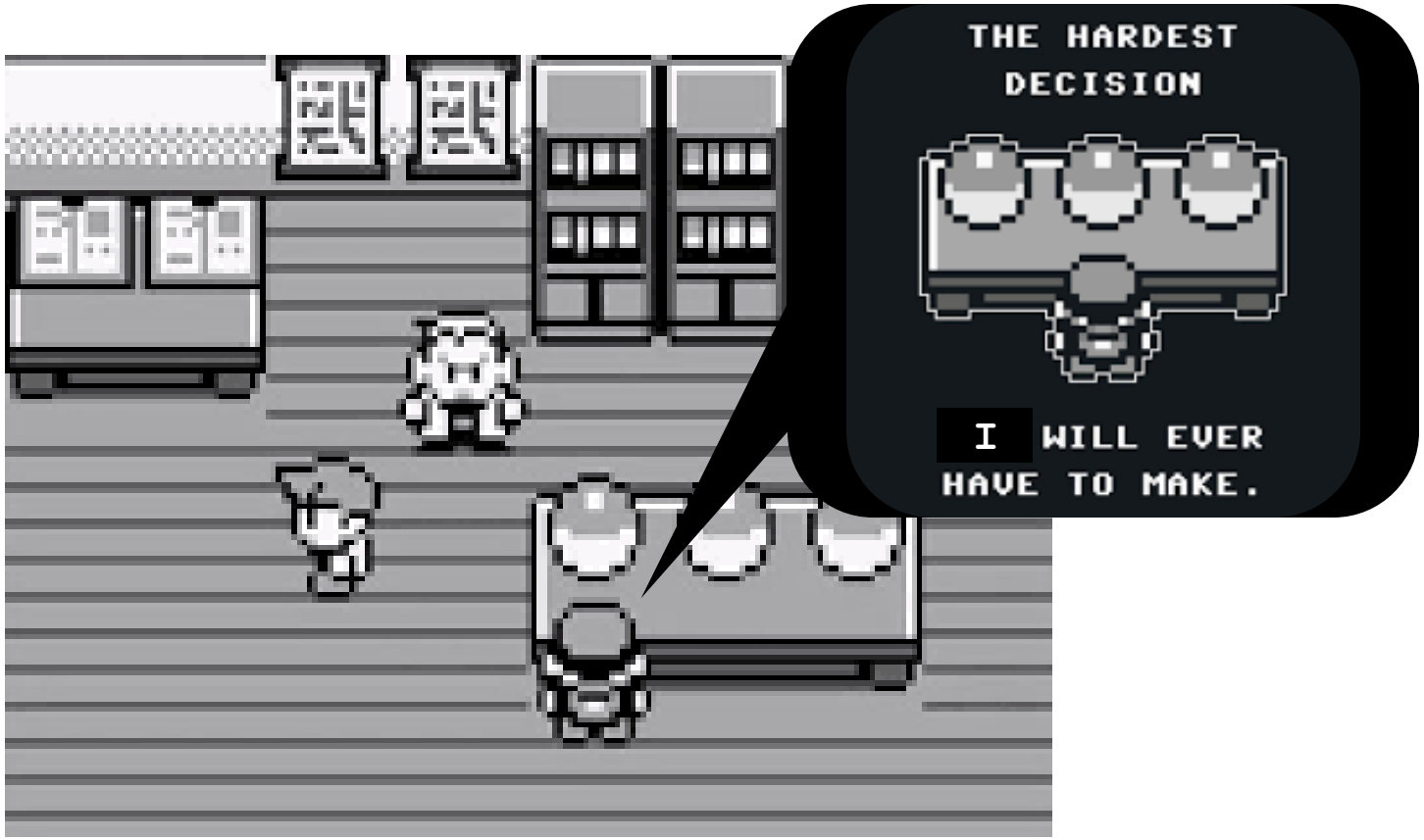
CS 1502H 程序设计思想与方法 (C++)
Principles and Methodology in Programming
Chapter 3. 分支程序设计

陈东尧
上海交通大学
2025年秋季

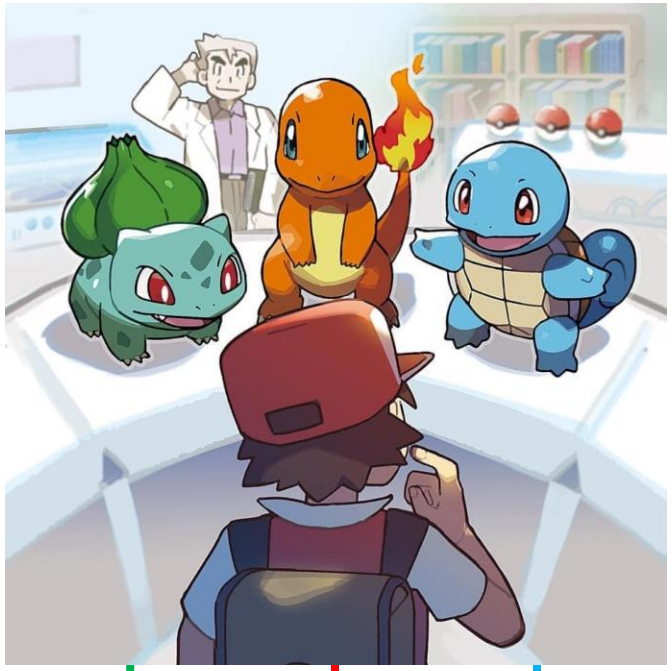
第二章回顾

- 用变量记录数据；
- 读取用户提供的数值数据；
- 在屏幕上显示文本和数据；
- 算数和赋值计算：
  - 变量赋值
  - 多重赋值与赋值嵌套
  - 自增自减运算符
  - 数据类型转换

什么是分支？



什么是分支？



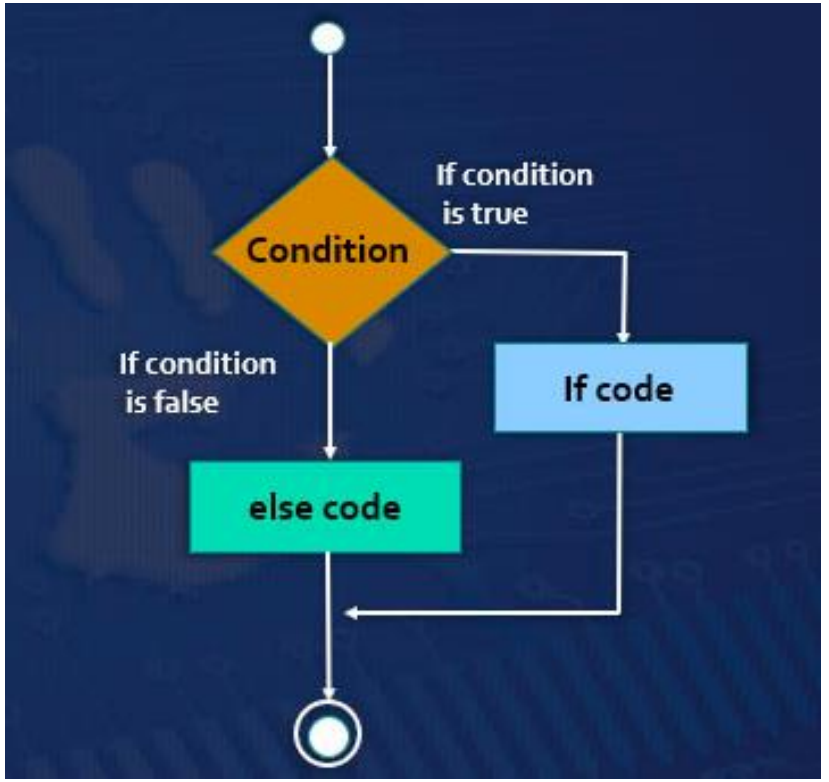
草系路线

火系路线

水系路线

什么是分支？

- 例如，解一元二次方程，结果可能正确、无穷大、异常终止。
  - 没有考虑特殊情况 $b^2-4ac<0$
- 如何将成绩百分制转换成等级制（A~D）？
  - 不同的分数段对应不同的等级



- 解决方案
  - 检测各种情况：关系表达式和逻辑表达式
  - 处理不同情况的机制：**if**和**switch**语句

章节内容

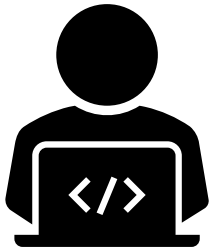
- 关系表达式
- 逻辑表达式
- if语句
- switch语句

关系表达式：简单介绍

- 关系表达式用来实现比较两个值的大小。
- 6个关系运算符 ：

<, <=, >, >=, 高优先级

==, != 低优先级



代码演示

chapter3-trivia.cpp

- 用关系运算符将二个表达式连接起来称为关系表达式。格式如下：

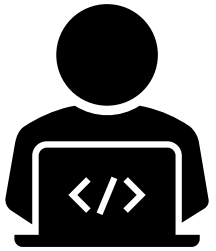
表达式 --- 关系运算符 ---- 表达式

例：　(a + b) > (c – 3)　(a = b) < 5　(a > b) == (c < d)　-2 < -1 < 0

关系表达式：简单介绍

- 关系表达式用来实现比较两个值的大小。
- 6个关系运算符 ：

<, <=, >, >=, 高优先级　　**==, !=** 低优先级



代码演示

chapter3-trivia.cpp

- 用关系运算符将二个表达式连接起来称为关系表达式。格式如下：

表达式 --- 关系运算符 ---- 表达式

例：　(a + b) > (c – 3)　(a = b) < 5　(a > b) == (c < d)　-2 < -1 < 0

- 关系表达式的结果是布尔型： true 或 false
- C++中关系表达式结果的自动类型转换： true (1), false (0)

问题：　(4 > 3) == (1 + 3) ➔ ?

关系表达式：优先级（Precedence）

- **优先级：高于赋值运算符，低于算术运算符；**

```
c > a + b       等效于  c > (a + b)
a > b == c      等效于  (a > b) == c
a == b < c      等效于   a == (b < c)
a = b > c       等效于   a = (b > c)
```

<, <=, >, >=,　　**==, !=**

高优先级　　**低优先级**

关系表达式：优先级（Precedence）

- **优先级：高于赋值运算符，低于算术运算符；**

  c > a + b　　等效于　c > (a + b)

  a > b == c　　等效于　(a > b) == c

  a == b < c　　等效于　a == (b < c)

  a = b > c　　等效于　a = (b > c)

  <, <=, >, >=,　　**==, !=**

  高优先级　　**低优先级**

- 下面这些哪些可以去掉括号而不改变逻辑？

(a + b) > (c – 3)　(a = b) < 5　(a > b) == (c < d)　(-2 < -1) < 0

关系表达式：优先级（Precedence）

(a + b) > (c – 3)　(a = b) < 5　(a > b) == (c < d)　(-2 < -1) < 0

```
int main(){
    int a = 1, b=2, c=3, d = 4;
    cout<<((a + b)>(c - 3)) <<'\n';
    cout <<((a = b) < 5) <<'\n';
    cout <<((a > b) == (c < d)) <<'\n';
    cout <<((-2 < -1) < 0) <<'\n';

    //逻辑运算符优先级：高于赋值运算符，低于算术运算符;
    cout << "After removing parentheses:"<<'\n';
    cout<<(a + b > c - 3) <<'\n';
    cout <<(a = b < 5) <<'\n';
    cout <<(a > b == c < d) <<'\n';
    cout <<(-2 < -1 < 0) <<'\n';
}
```

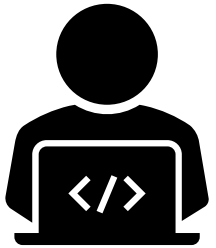
$(-2 < -1) < 0$

↓

$1 < 0$

↓

False



代码演示

chapter3-precedence.cpp

关系表达式：小练习

- 当int a = 2, b = 4, c= 1, d = 0时，　int x = a + b > c == ++d的值为（　　）

关系表达式：小练习

- 当int a = 2, b = 4, c= 1, d = 0时，　int x = a + b > c == ++d的值为

（　　）

2+4 > 1 == 1

6 > 1 == 1

1 == 1

扩展阅读：C++中的优先级

- https://en.cppreference.com/w/cpp/language/operator_precedence

C++ Operator Precedence

The following table lists the precedence and associativity of C++ operators. Operators are listed top to bottom descending precedence.

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right
2	`a++` `a--`	Suffix/postfix increment and decrement	
	type`()` *type*`{}`	Functional cast	
	`a()`	Function call	
	`a[]`	Subscript	
	`.` `->`	Member access	
3	`++a` `--a`	Prefix increment and decrement	Right-to-left
	`+a` `-a`	Unary plus and minus	
	`!` `~`	Logical NOT and bitwise NOT	
	(*type*)	C-style cast	
	`*a`	Indirection (dereference)	
	`&a`	Address-of	
	`sizeof`	Size-of[note 1]	
	`co_await`	await-expression (C++20)	
	`new` `new[]`	Dynamic memory allocation	
	`delete` `delete[]`	Dynamic memory deallocation	
4	`.*` `->*`	Pointer-to-member	Left-to-right
5	`a*b` `a/b` `a%b`	Multiplication, division, and remainder	
6	`a+b` `a-b`	Addition and subtraction	
7	`<<` `>>`	Bitwise left shift and right shift	
8	`<=>`	Three-way comparison operator (since C++20)	
9	`<` `<=` `>` `>=`	For relational operators < and ≤ and > and ≥ respectively	
10	`==` `!=`	For equality operators = and ≠ respectively	
11	`&`	Bitwise AND	
12	`^`	Bitwise XOR (exclusive or)	
13	`|`	Bitwise OR (inclusive or)	
14	`&&`	Logical AND	
15	`||`	Logical OR	
16	`a?b:c`	Ternary conditional[note 2]	Right-to-left
	`throw`	throw operator	
	`co_yield`	yield-expression (C++20)	
	`=`	Direct assignment (provided by default for C++ classes)	
	`+=` `-=`	Compound assignment by sum and difference	
	`*=` `/=` `%=`	Compound assignment by product, quotient, and remainder	
	`<<=` `>>=`	Compound assignment by bitwise left shift and right shift	
	`&=` `^=` `|=`	Compound assignment by bitwise AND, XOR, and OR	
17	`,`	Comma	Left-to-right

章节内容

- 关系表达式
- 逻辑表达式
- if语句
- switch语句

逻辑表达式-概念

- 逻辑表达式是用于实现更复杂的情况。
- 3个逻辑运算符:
  - 一元 unary operator：！(not)
  - 二元 binary operator： && (and)， || (or)

一元就是只需要一个操作数，如a--、a++、 !a、 ~a等
二元就是需要两个操作数才能完成运算 如典型的a+b、 a-b、 a*b、 a/b等

优先级：！ 大于 关系运算符 大于 && 大于 ||

逻辑表达式-概念

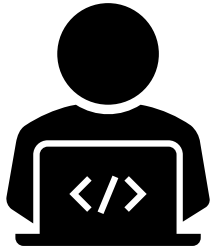
- **优先级：！ 大于 关系运算符 大于 && 大于 ||**

例： 0 < x < 10可以写成： 0 < x && x < 10，或者等价于： **!(x <= 0 || x >= 10)**

- 逻辑表达式的结果是布尔型： true 或 false
- 表达式可以为任意类型的数据，会自动类型转换： true (1)， false (0)

例： 5 % 2 && 4

5 > 3 && 2 || 8 < 4 - !0



代码演示

chapter3-trivia-2.cpp

C++中的符号结合律（Associativity）

运算符的结合性是指相同优先级的运算符在同一个表达式中，且没有括号的时候，运算符和操作数的结合方式，通常有从左到右结合和从右到左结合两种方式

- 假设~是一个运算符，又有表达式a~b~c
- 左结合（left-associative）
  - 被解析为(a~b)~c
- 右结合（right-associative）
  - 被解析为a~(b~c)

优先级表

优先级	运算符	结合性
1	[ ] ( ) . –> 后置 ++ 后置 ––	左→右
2	前置 ++ 前置 –– sizeof & * +（正号） –（负号） ~ ！	右→左
3	（强制转换类型）	右→左
4	.* ->*	左→右
5	* / %	左→右
6	+ –	左→右
7	<< >>	左→右
8	< > <= >=	左→右
9	== !=	左→右
10	&	左→右
11	^	左→右
12	|	左→右
13	&&	左→右
14	||	左→右
15	? :	右→左
16	= *= /= %= += –= <<= >>=&= ^= |=	右→左
17	,	左→右

C++ Operator Precedence

The following table lists the precedence and associativity of C++ operators. Operators are listed top to bottom descending precedence.

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right
2	`a++` `a--`	Suffix/postfix increment and decrement	
	type`()` *type*`{}`	Functional cast	
	`a()`	Function call	
	`a[]`	Subscript	
	`.` `->`	Member access	
3	`++a` `--a`	Prefix increment and decrement	Right-to-left
	`+a` `-a`	Unary plus and minus	
	`!` `~`	Logical NOT and bitwise NOT	
	(*type*)	C-style cast	
	`*a`	Indirection (dereference)	
	`&a`	Address-of	
	`sizeof`	Size-of[note 1]	
	`co_await`	await-expression (C++20)	
	`new` `new[]`	Dynamic memory allocation	
	`delete` `delete[]`	Dynamic memory deallocation	
4	`.*` `->*`	Pointer-to-member	Left-to-right
5	`a*b` `a/b` `a%b`	Multiplication, division, and remainder	

6	`a+b  a-b`	Addition and subtraction	
7	`<<  >>`	Bitwise left shift and right shift	
8	`<=>`	Three-way comparison operator (since C++20)	
9	`<  <=  >  >=`	For relational operators < and ≤ and > and ≥ respectively	
10	`==  !=`	For equality operators = and ≠ respectively	
11	`&`	Bitwise AND	
12	`^`	Bitwise XOR (exclusive or)	
13	`|`	Bitwise OR (inclusive or)	
14	`&&`	Logical AND	
15	`||`	Logical OR	
16	`a?b:c`	Ternary conditional[note 2]	Right-to-left
	`throw`	throw operator	
	`co_yield`	yield-expression (C++20)	
	`=`	Direct assignment (provided by default for C++ classes)	
	`+=  -=`	Compound assignment by sum and difference	
	`*=  /=  %=`	Compound assignment by product, quotient, and remainder	
	`<<=  >>=`	Compound assignment by bitwise left shift and right shift	
	`&=  ^=  |=`	Compound assignment by bitwise AND, XOR, and OR	
17	`,`	Comma	Left-to-right

逻辑表达式：小练习

- 检查字符变量a的内容是否为字母。

```
a >= 'a' && a <= 'z'    || a >= 'A' && a <= 'Z'
```

  - 注意：不能写成 'a' <= a <= 'z' || 'A' <= a <= 'Z'，为什么？
- 整型变量m的内容是否为偶数

```
m%2 == 0
```

- 年份year是否为闰年

```
待续
```

逻辑表达式：设计技巧

- 短路求值：逻辑表达式在执行时，先处理左边。如左边已能决定此逻辑表达式的结果，则右边不执行。通过巧妙设计，可以减少程序执行时间。
  - && 表达式：把false可能性较大的条件放在左边
  - ||表达式：把true可能性较大的条件放在左边

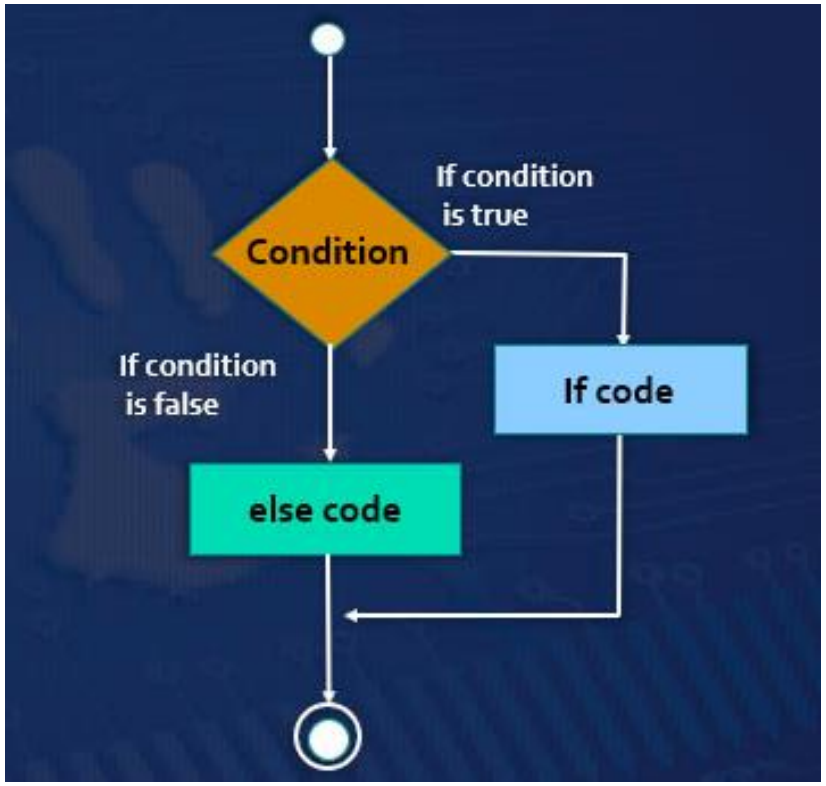
- 避免在一个逻辑中完成过多任务
- 在翻译自然语言的时候要注意不能直译

x不等于2或3：　x != 2 || x != 3　　x != 2 && x != 3　　!(x ==2 || x == 3)

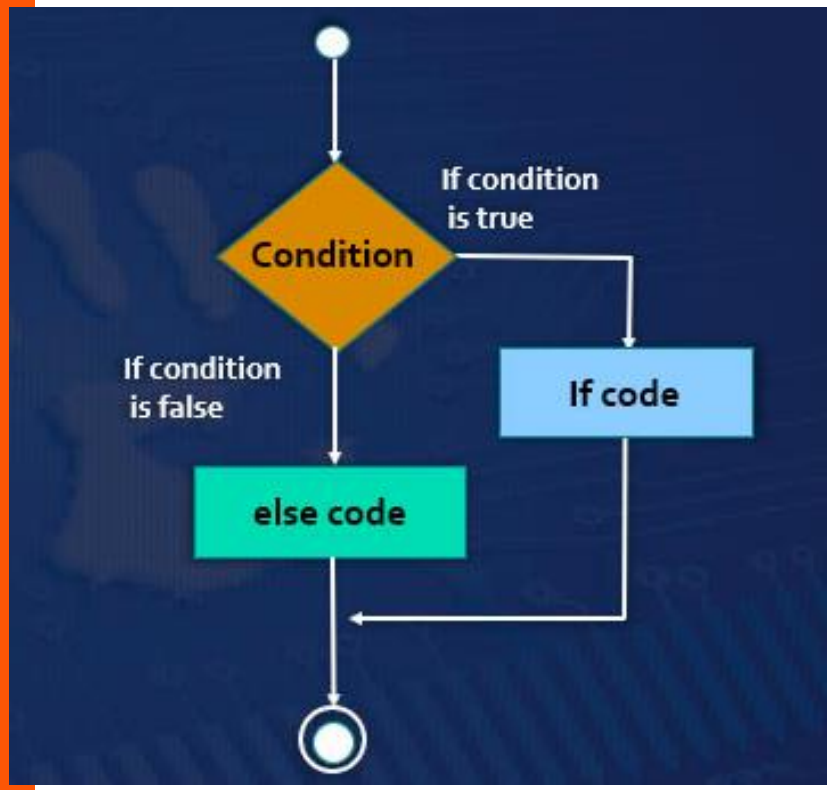


章节内容

- 关系表达式
- 逻辑表达式
- if语句
- switch语句



If语句语法



- if语句的格式

if （条件） 语句 // 条件为true时，执行语句

if （条件） 语句1 else 语句2 //条件为true时，执行语句1，否则执行语句2

  - 条件为关系表达式或逻辑表达式。

例：
```
if (grade >= 60)
    cout << “passed”;
```

```
if (grade >= 60)
    cout << “passed”;
else
    cout << “failed”;
```

条件语句使用时注意

- 条件的结果值应该是 true 或 false，它们是C++中bool类型的值
- 条件可为任意表达式，不一定是关系表达式。0 为false，非 0 为true。
- 注意合理的缩排，使程序结构更加清晰

例子：求解一元二次方程

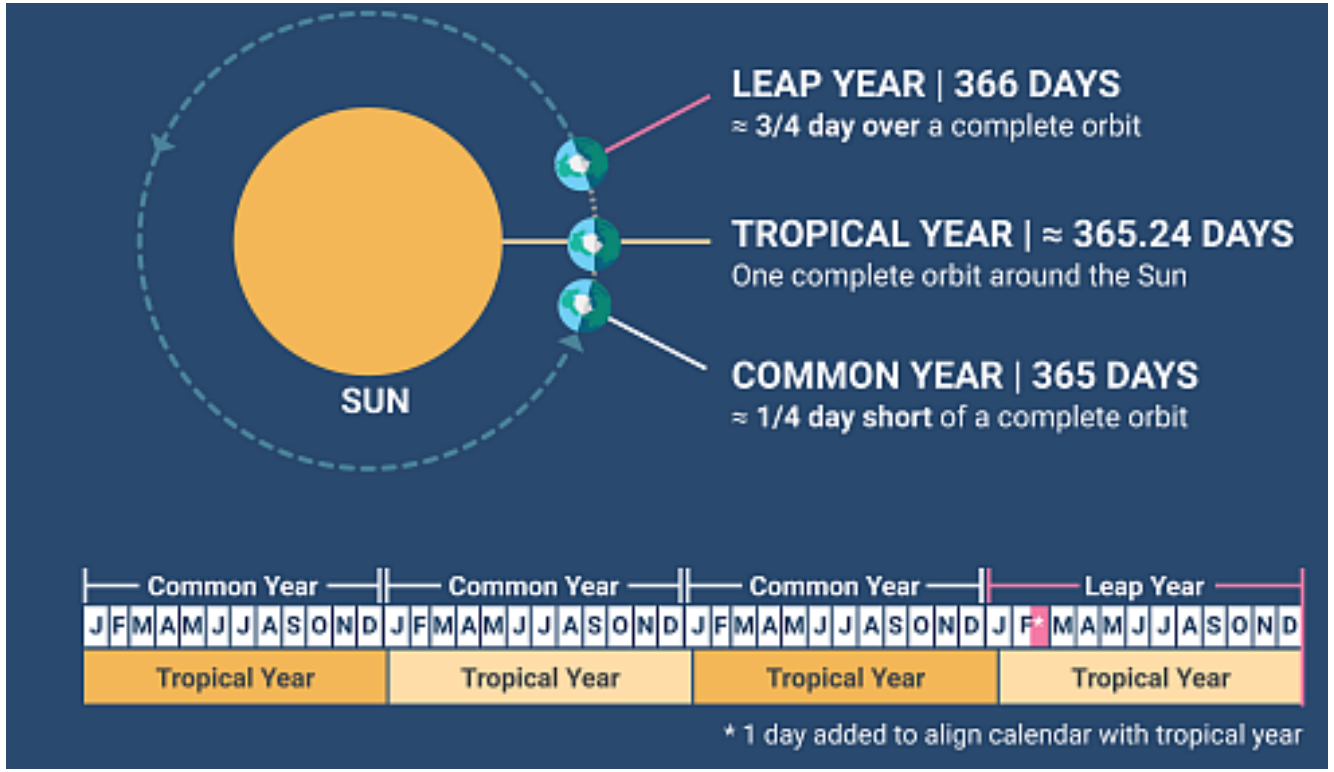
注意合理的缩排

注意合理的缩排

```
int main()
{
    double a, b, c, x1, x2, dlt;
        cout << "请输入3个参数：  " << endl;
        cin >> a >> b >> c;
        if (a == 0) cout << "不是一元二次方程" << endl;
        else {
            dlt = b * b - 4 * a * c;
            if (dlt >= 0) {
                x1 = (-b + sqrt(dlt)) / 2 / a;
                x2 = (-b - sqrt(dlt)) / 2 / a;
                cout << "x1=" << x1 << " x2=" << x2 << endl;
            }
            else cout << "无根" << endl;
        }
        return 0;
}
```

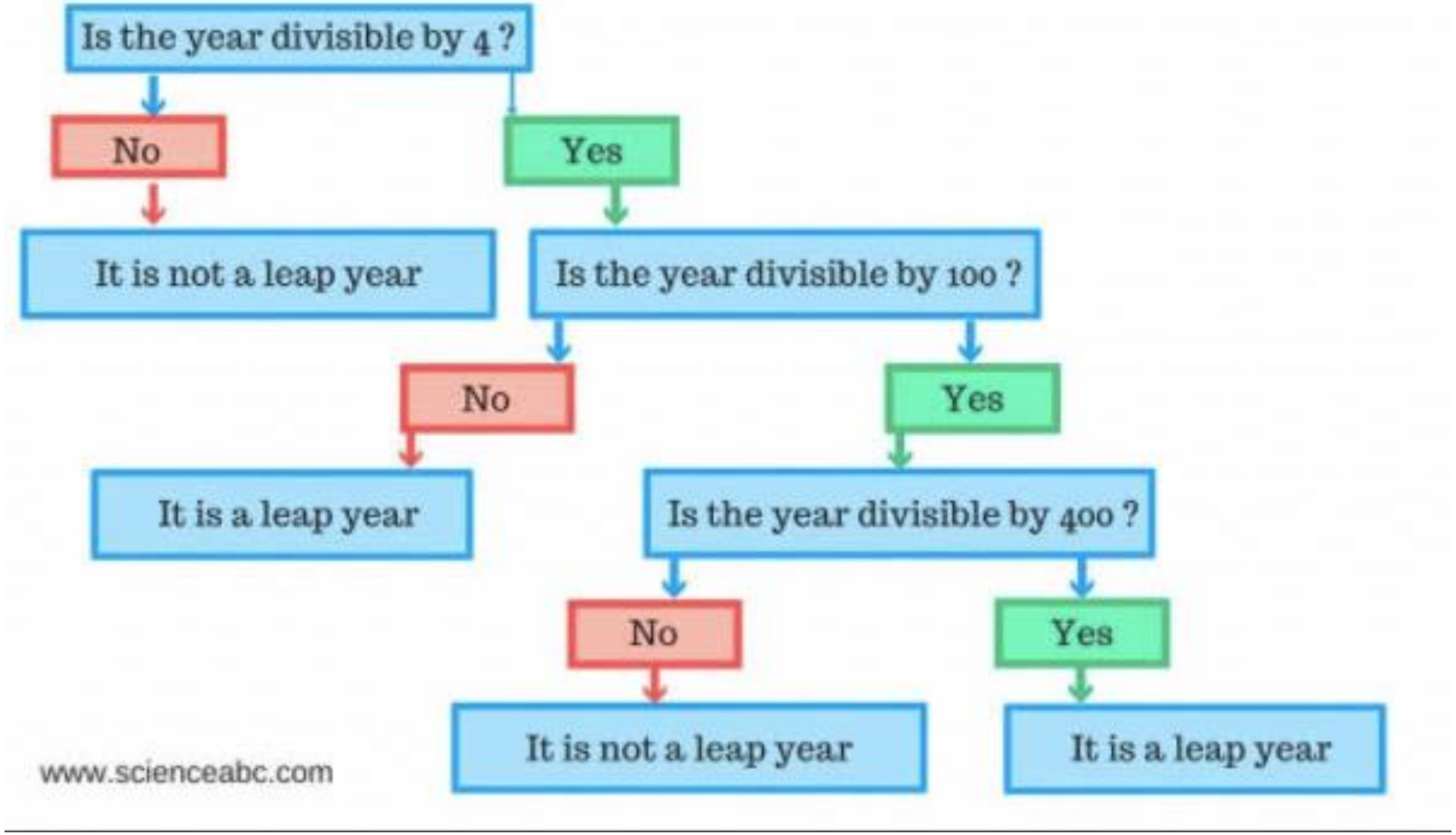
if 语句练习：判断闰年（Leap year）

闰年判断

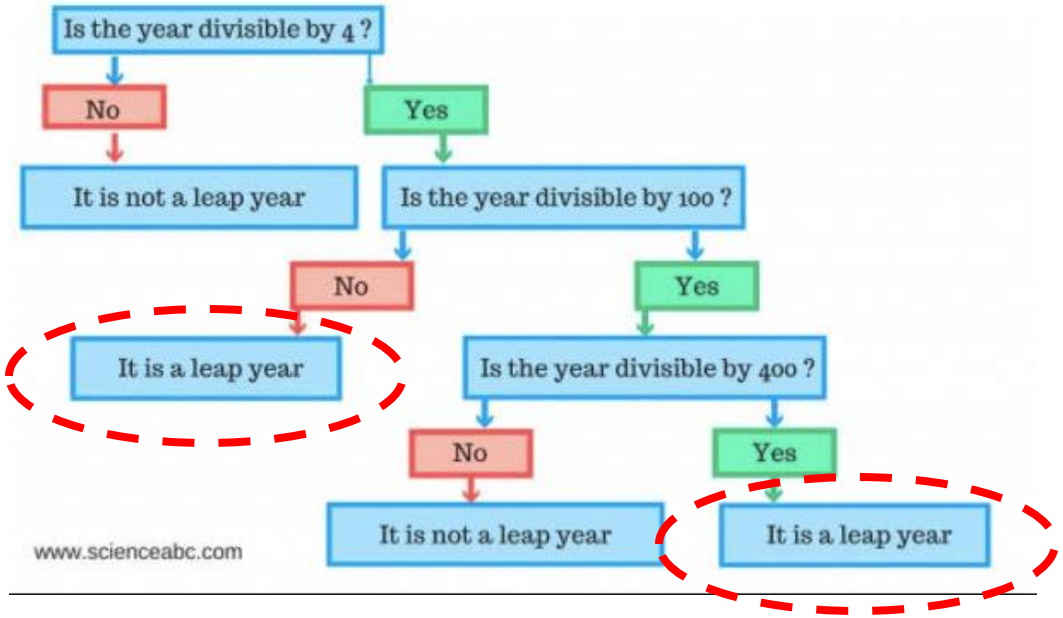


- 闰年(Leap Year)是为了弥补因人为历法规定造成的年度天数与地球实际公转周期的时间差而设立的。
- 闰年的计算方法：
  - 公元纪年的年数可以被四整除，即可能为闰年；
    - 例外是：世纪数被100整除为平年，被400整除的才为闰年。比如说1600年是闰年，而1900年不是。

闰年判断


Is the year divisible by 4 ?
No
Yes
It is not a leap year
Is the year divisible by 100 ?
No
Yes
It is a leap year
Is the year divisible by 400 ?
No
Yes
It is not a leap year
It is a leap year
www.scienceabc.com

闰年判断：分析

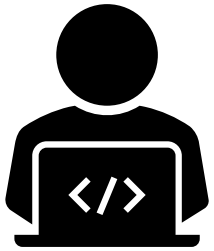


- 能被4整除同时不能被100整除则是闰年
- 能被400整除就一定是闰年

【扩展阅读，**为什么将能被4整除但不能被100整除和能被400整除的年份设为闰年**】
https://blog.csdn.net/lihui126/article/details/46236981

if 语句练习：判断闰年（Leap year）

```
#include<iostream>
using namespace std;
int main() {
	int year = 2016;
	if (((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0))
		cout<<year<<" is a leap year";
	else
		cout<<year<<" is not a leap year";
	return 0;
}
```



代码演示

Chapter3-leap-year.cpp

if 语句：小练习

- 改错题。

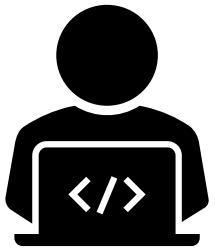
```
1    int main()
2    {
3        int x;
4        cout<<"Please input x:";
5        cin>>x;
6        if (x=8)
7          cout<<"x equals 8!"<<endl;
8        else
9          cout<<"x does not equal 8!"<<endl;
10       return 0;
11   }
```

1. x = 2时输出为（　　），
   x = 8时输出为（　　）。
2. 请修改第(　　)行代码为：
   (　　)。

请注意：赋值语句作为条件判据时，会按照所赋的值来决定true or false。

例如：x赋值为0则会按照false的分支执行；

赋值为非0则会按照true的分支执行。



代码演示

chapter3-class-debug.cpp

If 语句嵌套语法

- if语句的格式
  - if （条件）语句可包含if语句
  - 配对原则：**每个else子句是和在它之前最近的一个没有else子句的if语句配对。**

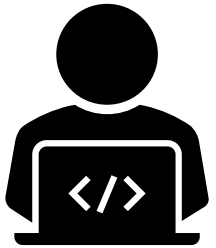
```
if (x < 100)
   if (x > 90)
         语句1
   else
         if (x<80) 语句2
         else 语句3
else 语句4;
```

用**缩进对齐**表示层次

用 { } 来确定层次关系

```
if (x < 100)
{
   if (x > 90)
         语句1
   else
   {
         if (x<80) 语句2
         else 语句3
   }
}
else
   语句4;
```

再看求解一元二次方程的代码



代码演示

chapter3-solve-equation.cpp

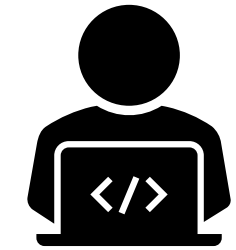
注意合理的缩排

注意合理的缩排

```
int main()
{
    double a, b, c, x1, x2, dlt;
        cout << "请输入3个参数：" << endl;
        cin >> a >> b >> c;
        if (a == 0) cout << "不是一元二次方程" << endl;
        else {
            dlt = b * b - 4 * a * c;
            if (dlt >= 0) {
                x1 = (-b + sqrt(dlt)) / 2 / a;
                x2 = (-b - sqrt(dlt)) / 2 / a;
                cout << "x1=" << x1 << " x2=" << x2 << endl;
            }
            else cout << "无根" << endl;
        }
        return 0;
}
```

用嵌套来重写闰年判断

```
int main() {
    int year;
    cout << "Enter a year: ";
    cin >> year;
    if (year % 4 == 0) {
        if (year % 100 == 0) {
            if (year % 400 == 0){
                cout << year << " is a leap year.";
            } else {
                cout << year << " is not a leap year.";
            }
        }
        else
          cout << year << " is a leap year.";
    }
    else
        cout << year << " is not a leap year.";
    return 0;
}
```



代码演示

chapter3-leap-year-nesting.cpp

条件表达式

- ？：运算符 ：问号冒号运算符
- 作用：更加简练地用来表达条件执行的方式
- 形式 ：

  (条件)？表达式1：表达式2
- 执行过程：首先计算条件值。如果条件结果为true，则计算表达式1的值，并将它作为整个表达式的值。如果条件结果为false，则整个表达式的值为表达式2的值。

条件表达式：实例

- 例如将x和y中值较大的一个赋值给max，可以用下列语句：

```
max = (x > y) ? x : y;
```

条件表达式：实例

- 例如将x和y中值较大的一个赋值给max，可以用下列语句：

```
max = (x > y) ? x : y;
```

- ？：运算符用于输出。例如，想输出一个布尔变量flag的值，如果直接用

```
cout << flag;
```

当flag为“真”时，输出为1；当flag为“假”时，输出为0。

如果我们想让flag为“真”时输出true，为“假”时输出false，可以用if 语句

```
if（flag）  cout << “true”;
  else  cout << “false”;
```

也可以做此修改：

```
cout << (flag ? "true" : "false" ) << endl;
```

章节内容

- 关系表达式
- 逻辑表达式
- if语句
- switch语句

Switch语句

- switch 语句适用于多分支情况

Switch语句语法

- switch 语句适用于多分支情况。格式为：

```
switch（表达式）
  {
     case 常量表达式1：语句1  //当表达式为常量表达式1时，执行语句1到语句n+1;

     case 常量表达式2：语句2 //当表达式为常量表达式2时，执行语句2到语句n+1;
          .
          .
     case 常量表达式n：语句n //当表达式为常量表达式n时，执行语句n到语句n+1;

     default：语句n+1          //否则执行语句n+1。default可以省略。
  }
```

  - 表达式的值应为整型（包括字符型和枚举型）。
  - 常量表达式不包含变量。

Switch语句语法

- default子句可以省略
- default子句省略时，当表达式的值不等于表达式1到表达式n的值时，直接跳出switch语句，执行switch语句的下一语句

Switch语句 --- Break语句

- 可以用break语句跳出当前switch：

```
switch（表达式）
{
  case 常量表达式1：语句1; break; //当表达式为常量表达式1时, 执行语句1，跳出switch;

  case 常量表达式2：语句2; break; //当表达式为常量表达式2时, 执行语句2，跳出switch;
                  .
                  .
   case 常量表达式n：语句n; break; //当表达式为常量表达式n时, 执行语句n，跳出switch;

   default：语句n+1                    //否则执行语句n+1。default可以省略。
}
```

  - 表达式的值应为整型（包括字符型和枚举型）。
  - 常量表达式不包含变量。
  - 每个case后面可以跟符合语句，不需要{ }

用Switch语句重写百分制转换

score>=90	**A**
90>score>=80	**B**
80>score>=70	**C**
70>score>=60	**D**
score<60	**E**

```
switch(score)
{
        case score >= 90: cout << "A"; break;
        case score >= 80: cout << "B"; break;
        case score >= 70: cout << "C"; break;
        case score >= 60: cout << "D"; break;
        default: cout << "E";
}
```

需要是常量表达式

用Switch语句重写百分制转换

score>=90	**A**
90>score>=80	**B**
80>score>=70	**C**
70>score>=60	**D**
score<60	**E**

```
switch( score/10 )
 {
        case 10:
        case  9: cout << "A"; break;
        case  8: cout << "B"; break;
        case  7: cout << "C"; break;
        case  6: cout << "D"; break;
        default: cout << "E";
 }
```

switch 语句 — 自动出四则运算计算题

- 算法设计：要求自动出0 - 9之间的四则运算题， 并批改结果。
- 关键在生成题目， 其算法伪代码为：

```
Switch(题目类型)
{ case 加法： 显示题目，输入和的值，判断正确与否
  case 减法： 显示题目，输入差的值，判断正确与否
  case 乘法： 显示题目，输入积的值，判断正确与否
  case 除法： 显示题目，输入商和余数的值，判断正确与否
}
```

- 如何让程序每次执行的时候都出不同的题目？包括不同的题目类型和不同的计算数据。
  - 解决方案：随机数生成器rand()

switch 语句 — 自动出四则运算计算题— rand()

- 随机数生成器rand()：能随机生成0到RAND_MAX之间的整型数
- 将生成的随机数映射到0 - 9之间：
  - rand() % 10
- 运算符的生成：用编码0 - 3表示四个运算符。因此题目的生成就是生成0 - 3之间的随机数。

【拓展阅读，C++ rand】https://www.cplusplus.com/reference/cstdlib/rand/

switch 语句 — 自动出四则运算计算题— rand()

- 随机数生成器rand()：能随机生成0到RAND_MAX之间的整型数
- 将生成的随机数映射到0 - 9之间：
  - rand() % 10

  注： to get a random integer in the range [0, x]：rand() % (x + 1)

- 运算符的生成：用编码0 - 3表示四个运算符。因此题目的生成就是生成0 - 3之间的随机数

【拓展阅读，C++ rand】https://www.cplusplus.com/reference/cstdlib/rand/

switch 语句 — 自动出四则运算计算题— rand()

- 设置种子的函数srand ： srand （种子）
- 如何让程序每次执行时选择的种子都不一样呢?
  - 选择系统时间为种子：time(NULL) 取当前的系统时间。

【拓展阅读，C++ rand】https://www.cplusplus.com/reference/cstdlib/rand/

switch 语句 — 自动出四则运算计算题例

```
#include <cstdlib>   //包含伪随机数生成函数
#include <ctime>     //包含取系统时间的函数
#include <iostream>
using namespace std;

int main()
{
  int num1, num2, op, result1, result2;
  //num1,num2:操作数，  op:运算符，  result1,result2: 结果

  srand(time(NULL));                    //随机数种子初始化

  num1=rand() % 10;// 生成运算数
  num2= rand() % 10 ;//生成运算数
  op= rand() % 4 ; // 生成运算符 0--+，  1-- -，  2--*，  3-- /
```

```
switch (op)
   {case 0: cout << num1 << "+" << num2 << "= ?" ; cin >> result1;
              if (num1 + num2 == result1)  cout << "you are right\n";
                    else  cout << "you are wrong\n";
                    break;
    case 1: cout << num1 << "-" << num2 << "= ?" ; cin >> result1;
              if (num1 - num2 == result1) cout << "you are right\n";
                    else  cout << "you are wrong\n";
              break;
    case 2: cout << num1 << "*" << num2 << "= ?" ; cin >> result1;
              if (num1 * num2 == result1) cout << "you are right\n";
             else  cout << "you are wrong\n";
                   break;
    case 3: cout << num1 << "/" << num2 << "= ?" ;  cin >> result1;
                    cout << “余数为 = ?";  cin >> result2;
              if  ((num1 / num2 == result1) && (num1 % num2 == result2))
                         cout << "you are right\n";
                     else  cout << "you are wrong\n";
               break; }
  return 0;
}
```

程序的不足

- 每次执行只能出一道题
- 减法可能出现负值
- 除法可能出现除0
- 结果太单调

小结

- 本章主要介绍了计算机实现逻辑思维的机制。主要包括两个方面：
  - 如何表示一个逻辑判断
  - 如何根据逻辑判断的结果执行不同的处理
- 逻辑判断
  - 关系表达式实现
  - 逻辑表达式
- 根据逻辑判断执行不同的处理
  - if语句
  - switch语句