### 1．

**（1）**

Boolean型

### 2．

**（1）**

A.

|  |
| --- |
| B：(int)5.8+1.0;  (int)5.8=5  5+1.0=6.0（double型）） |
| C：‘苹’+‘果’;  char型变量只能存储一个字符，所以‘苹‘+’果’实际上是一个字符串（String型） |
| D：(short)10+’a’;  short类型的变量占用1字节，而char型变量占用2字节，char型变量的精度大于short型，根据变量精度运算规则，输出的结果应为精度高的类型即char型。 |

**（2）**

C. m=0是一个赋值语句，并不会提供一个boolean型结果用以判断，所以会报错。

|  |
| --- |
| A：m-->0  根据运算符的优先级，先进行算术运算，即m--，再进行逻辑运算，最后结果仍然为boolean型，符合应有的参数类型，可以编译。 |
| B：m++>0  原因同上。 |
| D：m>100&&true  大小关系运算优先级为6，逻辑与运算优先级为11，先进行m>100（结果为false）再进行&&运算，最终结果为false。虽然为false，但是并没有语法错误，可以编译。 |

**（2）**

C.（只有将一个精度比较高的变量赋值给一个精度比较低的变量的时候才会出现这种情况。C的题目中char m=‘a’+x; x为int型变量，精度大于char型，故会提示。）

|  |
| --- |
| A：short t=12+’a’;  本题中，12和‘a’都是常量，常量之间做运算并不会提示精度问题。 |
| B：char c=‘a’+1;  虽然这条语句实质上和C.的语句等价，但是由于这里是加了一个常量，故也不会提示精度问题。 |
| D：byte n=‘a’+1;  同样是常量之间进行运算，不会提示精度问题。 |

### 3.

**（2）**

|  |
| --- |
| **public class** E {   **public static void** main(String[] args) {  **char** x = **'你'**, y = **'e'**, z = **'吃'**;  **if** (x > **'A'**) {  y = **'苹'**;  z = **'果'**;  }   **else** y = **'酸'**;  z = **'甜'**;  System.***out***.println(x + **","** + y + **","** + z);  } } |
| 结果：  你,苹,甜  分析：  首先对这三个变量初始化，然后x>’A’结果为真，执行y = '苹'; z = '果';两个语句。此时x=’你’, y=’苹’, z=’果’。然后跳过else的语句，再进行赋值z=’甜’。所以最后输出的时候就是这样。 |

**（2）**

|  |
| --- |
| **public class** E {  **public static void** main(String[] args) {  **char** c = **'\0'**;   **for** (**int** i = 1; i <= 4; i++) {   **switch** (i) {  **case** 1:  c = **'J'**;  System.***out***.print(c);  **case** 2:  c = **'e'**;  System.***out***.print(c);  **break**;  **case** 3:  c = **'p'**;  System.***out***.print(c);  **default**:  System.***out***.print(**"好"**);  }  }  } } |
| 结果：  Jeep好好  分析：  第一次i=1，执行case 1：的赋值并输出‘J’，由于之后没有break，就也执行了case2： 的赋值并输出了一个‘e’，随后跳出switch。  第二次i=2，执行case 2：的赋值并输出‘e’，之后break，立刻跳出switch。  第三次i=3，执行case 3：的赋值并输出‘p’，由于之后没有break，就也执行了default： 的输出：一个‘好’，随后跳出switch。  第四次i=4，执行default：的输出：一个‘好’，随后跳出switch，跳出for，结束整个 方法。 |

**（3）**

|  |
| --- |
| **public class** E {  **public static void** main(String[] args) {  **int** x = 1, y = 6;  **while** (y-- > 0) {   x--;  }  System.***out***.print(**"x="** + x + **", y="** + y);  } } |
| 结果：  x=-5, y=-1  分析：  y-->0先进行判断再进行自减运算。之前的每次循环的结果都是x-1同时y-1。但是当经过最后一次循环y=0时，还会再进入while进行一次判断，这时候y=0已经不符合条件了，故不执行x--。但是在最后一次y-->0的过程中已经进行过一次了，所以y此时等于-1，共减7，而--执行比y--少一次，故减6，为-5。 |

### 4.

**（3）**

**do-while写法**

|  |
| --- |
| **int** i=1; **double** sum = 0,item=1; **do** {  sum=sum+item;  i++;  item=item\*(1.0/i); } **while**(i<=20); System.***out***.println(sum); } |

**for循环写法**

|  |
| --- |
| **package** com.company;  **import** java.util.Scanner;  **public class** Main {  **public static void** main(String[] args) {  **double** sum = 0;  **double** item = 1;  **for** (**int** j = 1; j <= 20; j++) {  item = item \* (1.0 / j);  sum = sum + item;  }  System.***out***.println(sum);  } |