Zach David B. Maregmen
B.S. Computer Science
CS1A

I. Write assignment statements for the following:

1. Assign a value of 0 to **between** if n is less than − k or greater than +k; otherwise, assign 1

```
if (n < -k || n > k) {
between = 0;
} else {
between = 1;
}
```

2. Assign a value of 1 to **divisor** if digit is a divisor of num; otherwise, assign a value of 0

```
If (!(num % digit)) {
divisor = 1;
} else {
divisor = 0;
}
```

3. Assign a value of 1 to lowercase if ch is a lowercase letter, otherwise, assign a value of 0.

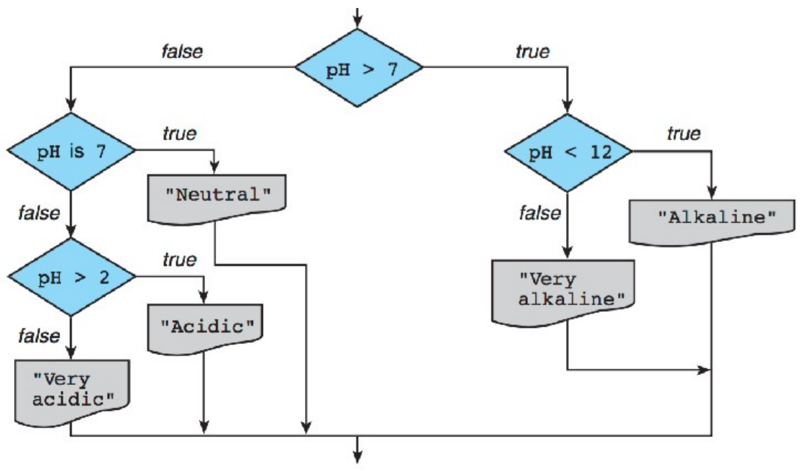
```
If (islower(ch)) {
lowercase = 1;
} else {
lowercase = 0;
}
```

II. implement the following decision table that categorizes a systolic blood pressure reading. Assume that the systolic blood pressure has been input as an integer.

Systolic Blood Pressure	**Category**
140 and higher	Hypertension
120-139	Prehypertensio n
Under 120	Normal

```
if (systolic_blood_pressure > 139) {
    printf(“Hypertension”);
} else if (systolic_blood_pressure > 119 && systolic_blood_pressure < 140) {
    printf(“Pre-hypertension”);
} else {
    printf(“Normal”);
}
```

III. Write a nested if statement for the decision diagrammed in the accompanying flow chart. Use a multiple alternative if for intermediate decisions where possible.



```
if (pH > 7) {
     if (pH < 12) {
          printf(“Alkaline”);
     } else {
          printf(“Very Alkaline”);
     }
} else {
     if (pH == 7) {
          printf(“Neutral);
     } else if (pH > 2) {
          printf(“Acidic”);
     } else {
          printf(“Very Acidic”);
     }
}
```

IV. What will printed by this carelessly constructed switch statement if the value of color is ‘R’

```
switch (color) {
      case ‘R’:
            printf(“Red\n”);
      case ‘B’:
            printf(“Blue\n”);
      case ‘Y’:
            printf(“Yellow\n”); }
}
```

- it will not print the statement because there is an excess ‘}’ identifier, therefore it will result in a compilation error. If we disregard the ‘}’ identifier, it will print

```
Red
Blue
Yellow
```

- it will continue to execute in every case because it didn’t tell the statement to break once the case returns True.

V. Given the following functions what will be the value returned after calling,

fun2(fun1(1,2,3), fun1(4,5,6));

```
int fun1(int a, int b, int c)
{
  if ((a >= b) && (a >= c)){
    return a;
  }
  else if ((b >= a) && (b >= c)){
    return b;
  }
  else {
    return c;
  }
}

int fun2(int a, int b)
{
  if (a < b) {
    return a;
  }
  else {
    return b;
  }
}

 /*
for fun1(1,2,3):
- it will return 3 since the the first and second condition is not true.
for fun1(4,5,6):
- it will return 6 since the first and second condition is not true.

Hence, fun2(fun1(1,2,3), fun1(4,5,6)) is equal to
  fun2(3,6).

for fun2(3,6):
the first condition returns true since 3 is less than 6.
```

Therefore, the returned value of fun2(fun1(1,2,3), fun1(4,5,6)) is 3.

VI. Write C Statements to carry out the following steps.

1. if item is nonzero, then multiply **product** by **item** and save the result in **product**; otherwise, skip the multiplication. In either case, print the value of the product.

```
if (!item) {
      product = product * item;
      printf("Product: %i", product);
} else {
      printf("Product: %i", product);
}
```

2. Store the absolute difference of **x** and **y** in **y**, where the absolute difference is **(x-y)** or **(y-x)**, whichever is positive. Do not use abs of fabs function in your solution.

```
if (0<=(x-y)) {
      y = x-y;
} else if (0<=(y-x)) {
      y = y-x;
}
```

3. If **x** is 0, add 1 to **zero_count**. If **x** Is negative, add **x** to **minus_sum**. If **x** is greater than 0, add **x** to **plus_sum**.

```
if (!x) {
      zero_count = zero_count + 1;
} else if (x < 0) {
      minus_sum = minus_sum + x;
} else {
      plus_sum = plus_sum + x;
}
```