Question 1

a) Given the following function:

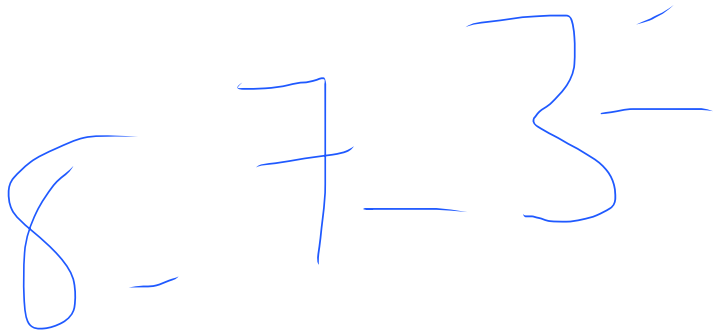
```
int func (int &a, int b) {

  int c;

  a = 7; b = 9; c = 10;
  return c * 5;
}
```

What would be output if the following code were executed?

```
int a = 12, b = 4, c = 6;

a = func (b, c);
cout << a << " " << b << " " << c << endl;
```

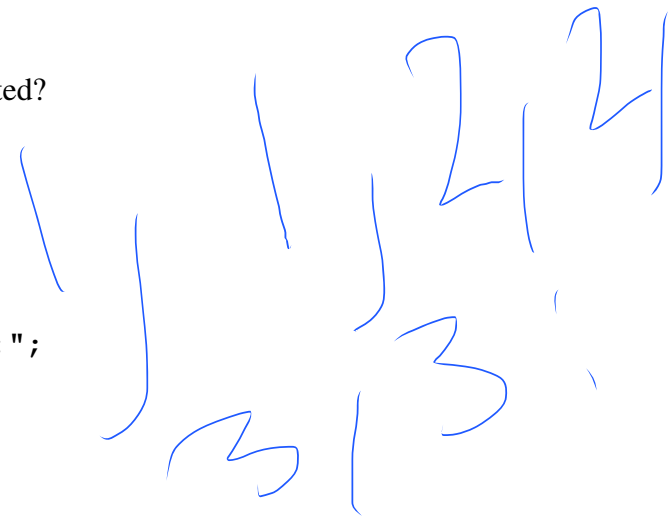
b) What would be output if the following code were executed?

```
int a = 8, b = 7, c = 10, d = 15;

if ((a < 6) || (c != 9)) {
  b++;
}
if ((d == 0) && (a != 8)) {
  b += 2;
}

cout << b << " " << d / 2 << " " << d % 4 << endl;
```



c) What would be output if the following code were executed?

```
int i, j;

for (i = 1; i <= 3; i++) {
  for (j = 1; j <= 3; j++) {
     if (j <= i) {
        cout << i << "," << j << ":";
     }
  }
}
cout << endl;
```



d) Show exactly what would appear if the following code were executed. Use # to represent blanks.

```
int a = 5; double b = 2.439;

cout << fixed << showpoint
     << setprecision(2) << setw(4) << a << setw(8)
     << b << endl;
```

Question 2

a) Rewrite the following code with the (many) errors corrected.

```
double i, s = 10;
double ar[s];
cout >> "Enter " >> s >> " values: ";
for (i==0, i<=s, i+=) {
cin >> ar[i];
}
```

b) Write code that adds up all of the values stored in array *data* and stores the result in variable *sum*. Be sure to declare any additional variables that you might require.

```
double data[20], sum;
```

c) Write an **expression** that evaluates to true if the value stored in variable z is evenly divisible by 5 and to false otherwise.

```
int z;
```

Question 3

Write a function that accepts an array of integer values and the number of values in this array. It should return true if any of the values in the array is the square of one of the **other** values, and false otherwise

example1:

the array contains { 4, 5, 9, -3, 25, 17 }
the function should return true (25 = 5 squared, 9 = -3 squared)

example2:

the array contains { 4, 5, 12, -3, 24, 17, 1 }
the function should return false (none of the values are the square of one of the **other** values)

Note the "**other** values". Having a "1" somewhere in the array does NOT guarantee a true result.

Your function must be consistent with the sample call below:

```
int values[20];
.....
if (checkForSquares (values, 20)) {
    .... // one of the values is the square of one of the other values
}
```

Question 4

File "data.txt" is supposed to contain between 10 and 100 integer values (inclusive of these limits) and all of the values are supposed to be different (i.e. there should not be any duplicate values). Write a program that reads this file and verifies that it is valid. Your program should terminate after having output exactly one of the following messages:

- `"File cannot be opened"`
- `"File contains bad data"`
- `"File contains too many values"`
- `"File contains too few values"`
- `"File contains at least one duplicate value"`
- `"File is OK"`

Question 5

In many applications it is convenient to represent the days of a year using "day numbers". January 1st is day 1, January 2nd is day 2, and so on through December 31st which is either day 365 (if the year isn't a leap year) or day 366 (if it is).

Write a function that, given year and a day number, "returns" the corresponding month (January = 1, etc.) and day of the month (1 = first day of month, etc).

Example: If your function is given 2007 as the year and 32 as the day number it should "return" 2 as the month and 1 as the day of month (the day number corresponds to February 1st).

Assume that somebody else has provided you with a function that returns the number of days in a month. The prototype for this function is

```
int daysInMonth (int month, int year); // month = 1 means January, etc.
```

You may use this function in your function. You **DO NOT** have to write it.

Your function must be consistent with the following sample call:

```
cout << "Enter year and day number: ";
cin >> year >> dayNumber;
getMonthAndDay  (year, dayNumber, month, dayOfMonth);
cout << "That is day " << dayOfMonth << " of month " << month);
```

Question 6

a) Write the implementation of a function which accepts a number of weeks, days, and hours (all three are integers) and returns the corresponding number of hours (also an integer). For example, 1 week, 2 days, and 2 hours is 218 hours. Read the notes below.

b) Write the implementation of a function which accepts a number of hours (an integer) and "returns" the corresponding number of weeks, days, and hours (all integers). For example, 218 hours corresponds to 1 week, 2 days, and 2 hours. Read the notes below.

Notes:

1. Comments and proper indentation are required for full marks.
2. Are there any invalid integer inputs to deal with?
    - You may choose how you deal with these, if applicable.
3. You may choose the name for each function, as well as the return type, and the number and type of parameters.
    - Make sure that you demonstrate good programming practice in all of your choices.
4. Give a brief explanation of your choices (as per notes 2 and 3) below each function.

Question 7

a) Write a function called *numOccurrences* which accepts an array of integers, its size, and a value, and returns the number of times that value appears in the array. Assume that all input values are valid (i.e. no error checking is needed on the array or its size). Comments and proper indentation are required for full marks.

For example, if an array of size 10 contained { 1, 2, 4, 2, 1, 2, 4, 3, 5, 4 }, then the value 2 occurs 3 times. (Also, the value 1 occurs 2 times, the value 5 occurs 1 time, and the value 7 occurs 0 times.)

b) Write a void function called *mostCommon* which accepts an array of integers and its size, and "returns" the element that occurs most often, and the number of times that it appears in the array. If there is a tie for the most common element, any of the most common elements may be returned. Your solution must make use of (i.e. call) the function in part (a) for full marks. Again, you may assume that all input values are valid (i.e. no error checking is needed on the array or its size). Comments and proper indentation are required for full marks.

For example, for the array above, the most frequently occurring values are 2 and 4, and each occurs 3 times. Thus *mostCommon* would "return" either 2 or 4 as the most common element, and 3 as the number of times it appears.

Question 8

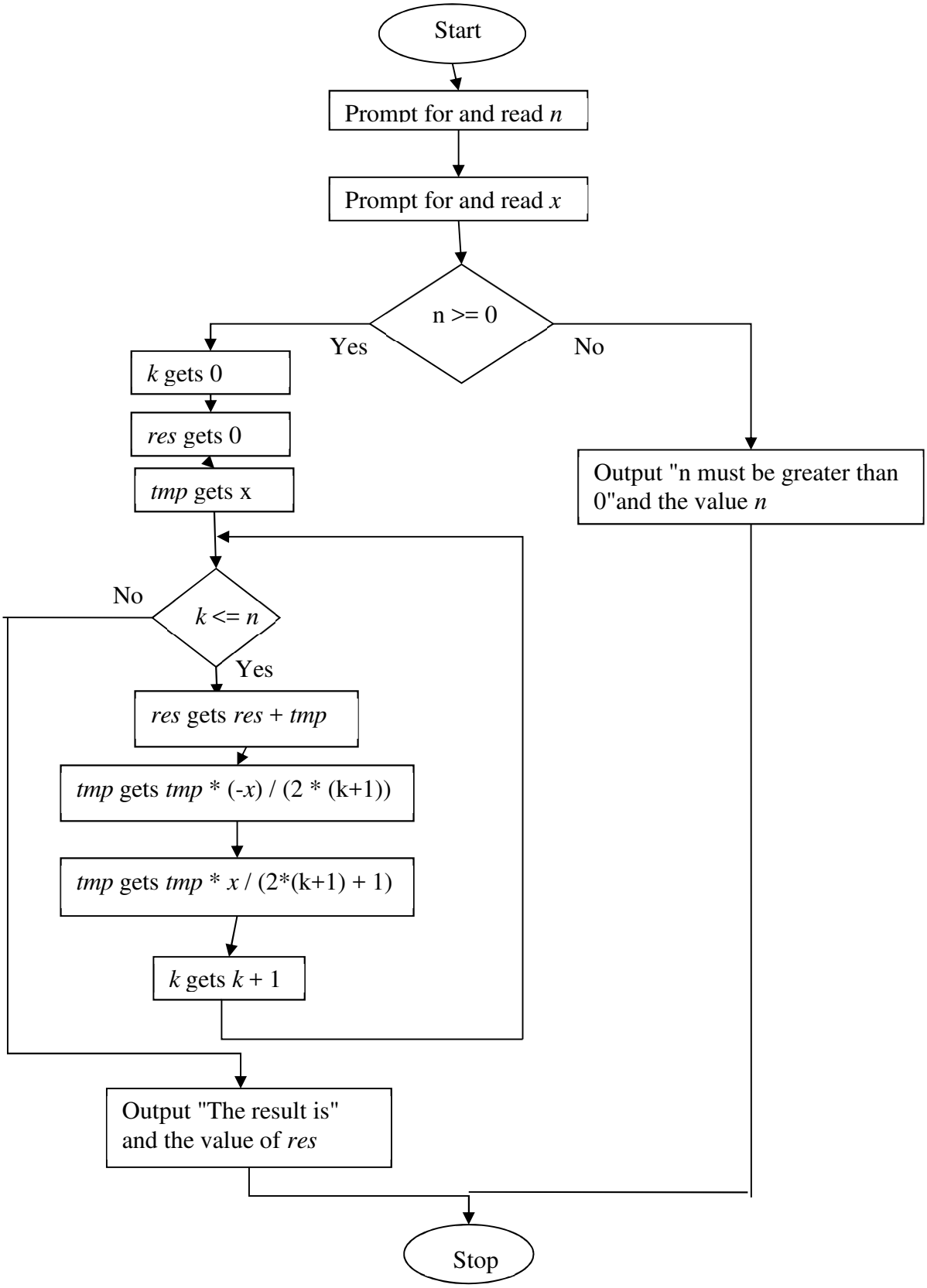
What will be output by the following program?

```
int alpha (int x, int &y) {
  cout << "PEAR " << x + y << endl;
  x = 17; y = 3;
  return x / y; // be careful
}

void beta (int x[], int n) {
  x[n] = x[0];
}

// continued on next page
```

```
int main () {

  int a = 17, b = 7, c = 10, d = 0,
      i, j, data[5] = { 0, 5, 4, 3, 8 }, t = 2;
  bool flag = true;

  for (i = 1; i < 5; i+= 2) {
    t *= data[i];
  }

  b = alpha (a, c);
  cout << "APPLE " << a % 5 << " " << b << " " << c << " "
       << t << endl;

  i = 1;
  do {
    d += data[i];
    if (d > 15 || i == 3) {
      flag = false;
    }
    i = i + 1;
  } while (flag);

  beta(data, 3);

  cout << "ORANGE " << d << " " << i << " " << data[3] << endl;

  for (i = 12; i > 10; i--) {
    for (j = 1; j < 3; j++) {
      cout << "MELON " << i << " " << j << endl;
    }
  }

  system("PAUSE"); return 0;

}
```

Question 9

This question involves converting a flowchart into the corresponding C++ code.
You are NOT required to produce the program framework or to declare any variables. Simply translate the logic represented by the flowcharts into C++ code.


Start
Prompt for and read n
Prompt for and read x
n >= 0
Yes
No
k gets 0
res gets 0
tmp gets x
Output "n must be greater than 0"and the value n
No
k <= n
Yes
res gets res + tmp
tmp gets tmp * (-x) / (2 * (k+1))
tmp gets tmp * x / (2*(k+1) + 1)
k gets k + 1
Output "The result is" and the value of res
Stop

Question 10

Function *zeta* is defined as follows:

$$zeta(n) = \frac{1}{1-(1/k_1)^2} * \frac{1}{1-(1/k_2)^2} * \frac{1}{1-(1/k_3)^2} .... * \frac{1}{1-(1/k_m)^2}$$

where $k_1, k_2, k_3, ...k_m$ are the prime numbers that are >= 2 and <= n

examples: $zeta(2) = \frac{1}{1-(1/2)^2}$

$$zeta(7) = \frac{1}{1-(1/2)^2} * \frac{1}{1-(1/3)^2} * \frac{1}{1-(1/5)^2} * \frac{1}{1-(1/7)^2}$$

Write a function that, given some integer *n* (guaranteed to be >= 2) computes and returns *zeta*(*n*). You may assume that you have been given and can make use of a function that determines whether or not a number is prime. You do **not** have to write this function. This function is called "isPrime". It takes an integer as a parameter, and returns true if the integer is prime, and false otherwise, i.e. its prototype is:

```
bool isPrime (int num);  // returns true if and only if "num" is prime
```

Question 11

Environment Canada has a program that processes daily temperatures. It includes the variable declarations given below.

```
const int MAXDAYS = 100;
double temperatures[MAXDAYS], minTemperature;
int actualDays, longestColdSnapDuration, longestColdSnapStart;
```

Part I

Part of the program involves finding the lowest temperature. The code is as follows:

```
minTemperature = findMinimumTemperature (temperatures, actualDays);
cout << "The lowest temperature is " << minTemperature << endl;
```

Unfortunately the code for function `findMinimumTemperature` has been lost. Save the day by writing this function. You should be able to work out everything you need to know.

Part II

The program also deals with cold snaps (one or more consecutive days all having a temperature below -10 degrees). The relevant code is given below:

```
findLongestColdSnap (temperatures, actualDays,
                                        longestColdSnapDuration,
longestColdSnapStart);
if (longestColdSnapDuration == 0) {
      cout << "There were no cold snaps." << endl;
} else {
      cout << "The longest cold snap started on day " <<
longestColdSnapStart
              << " and lasted for " << longestColdSnapDuration << "
days." << endl;
}
```

Function `findLongestColdSnap` is missing and you must write it as well. If there a number of equally long cold snaps, any one of them may be chosen as the longest cold snap. As the output is intended for humans, the first day for which there is temperature data is day one (and not day zero). You should be able to work out everything else you need to know from the declarations and code supplied.