PROGRAM 1

Working with Functions

"Be still my beating heart! Well, not too still, that would be bad."

OBJECTIVES

Upon successful completion of this lab, the student will be able to:

- use functions as part of a complete C++ program

PROGRAM: Heart Rate Zone Calculator

According to the Mayo Clinic (www.mayoclinic.com), one way to gauge the intensity of a physical workout is by measuring heart rate. If you know your maximum heart rate, you can classify how hard you are exercising as follows:

Light intensity	40 to 50 percent of maximum heart rate (MHR)
Moderate intensity	50 to 70 percent of MHR
Vigorous intensity	70 to 85 percent of MHR

Further, the Mayo Clinic offers the following formula to calculate MHR:

Maximum heart rate = 220 – age

Using the formulas above, it is a simple matter of multiplying MHR by the appropriate percentage to determine the heart rate zones for different exercise intensities. For example, if your age is 20, your MHR is 200 beats per minute (bpm). The zones are thus 80 to 100 bpm for light intensity, 100 to 140 bpm for moderate, and 140 to 170 bpm for vigorous.

However, there are alternative methods of determining exercise heart rate zones. One well-known method is the Karvonen method, which uses your resting heart rate (that is, your heart rate taken when you are still) in the calculation. According to this method, your target heart at a particular percentage of intensity is calculated by:

Target rate = ((maximum heart rate – resting heart rate) * % intensity level) + resting heart rate

So, for example, if your maximum heart rate were 200 bpm and your resting heart rate were 60 bpm, you would calculate your target rate at 70% intensity as:

Target rate = ((200 – 60) * 70%) + 60 = (140 * 70%) + 60 = 98 + 60 = 158 bpm

Your goal in this lab is to write a program, using functions, to write out two tables showing heart rates for different levels of intensity, from 40% to 100% by 5% steps. One table should use the Mayo Clinic formula, and the other table should use the Karvonen formula. The user will provide their age and their resting heart rate as input.

Sample run (user input shown highlighted):

```
Welcome to the target heart rate calculator!

Excuse my nosiness, but how old are you? 20

What is your resting heart rate? 60

Mayo Clinic Formula Heart Rate Table

Percent      Heart rate          Percent      Heart rate
40           80                  75           150
45           90                  80           160
50           100                 85           170
55           110                 90           180
60           120                 95           190
65           130                 100          200
70           140
```

(cont.)

PROGRAM 1

```
Karvonen Formula Heart Rate Table

Percent      Heart rate          Percent      Heart rate
40           116                 75           165
45           123                 80           172
50           130                 85           179
55           137                 90           186
60           144                 95           193
65           151                 100          200
70           158
```

To write this program, you must use functions. Do the following:

- Write an **int**-valued function called **getAge** with no parameters. The function should prompt the user for their age, and verify that it is a positive number. If it is not positive, the function should ask the user to enter their age again, until a valid age is entered. The function should then return the age value.
- Write an **int**-valued function called **getRHR** with no parameters. The function should prompt the user for their resting heart rate, and verify that it is between 25 and 350. If it is not in this range, the function should ask the user to enter their resting heart rate again, until a valid value is entered. The function should then return the value.
- Write an **int**-valued function called **calcMHR** with one parameter representing an age. The function should compute a maximum heart rate based on the age and return this value.
- Write an **int**-valued function called **calcMayo** with two parameters, representing a maximum heart rate and an exercise intensity percentage. Using the Mayo Clinic formula, it should calculate the target heart rate based on the MHR and percentage information received, and return this value.
- Write an **int**-valued function called **calcKarvonen** that takes three parameters, representing a maximum heart rate, a resting heart rate, and an exercise intensity percentage. Using the Karvonen formula, it should calculate the target heart rate based on the provided information, and return this value.
- Write the main program that declares the necessary variables and calls the above functions as appropriate to generate the desired tables. You will likely need two loops: one to produce the Mayo clinic table, and a second to produce the Karvonen table.

I suggest that you proceed a little bit at a time. Write one function, then complete enough of the main program to test that function to see if it works properly. Then add the next function, and test it as well. Proceed to build the program one function at a time, modifying the main program as you go to verify correct functionality.

As you write each function, make sure you write comments at the start of the function that describe the function's purpose, the meaning of each parameter (if any), and the meaning of the returned value (if any). Use the following example as a template for your commenting style.

```
double calcCircleArea(double radius)
// Purpose:    Computes area of a circle, given its radius.
// Parameters: radius - radius of the circle
// Returns:    Area of the circle.
```

Perform a sufficient number of test runs to show that the program works.

PROGRAM 2

PROGRAM 2

OBJECTIVES

Upon successful completion of this assignment, the student will be able to:

- write a program using loops, selection and functions

PROGRAM

You are to write a program that will play a dice game called "Sprock!" with two players. Here is the basic order of play:

1. The first player's turn starts by rolling five six-sided dice.
2. If the average of the dice is less than or equal to 3, the player gets 0 points for this turn and play passes to the next player. This is called "Sprocking out".
3. If the average of the dice is greater than 3 (there's approximately a 70% chance of this with five dice), the player is safe, adds the total of the dice to their turn score, and chooses one of the following options:
   a. They can roll again, but this time with one fewer die. For example, if the player gets a total of 22 with five dice, they can keep 22 for their turn score and roll again with four dice. However, they run the risk of Sprocking out again and ending their turn with 0 points. If they do not Sprock out, however, their turn score increases with the total of that roll. They can continue to do this all the way to rolling just one die. After that die has been rolled, play will automatically pass to the next player.
   b. They can declare their turn over, add their turn score to their total game score, and pass the remaining dice to the next player. Note that the total game score is not affected by Sprocking out. Only the turn score can be reset to zero.
4. The next player can start their turn either with 0 points and five dice, or the current number of points from the other player and however many dice are left at that point.

When the program starts, the program should ask the user how many points to play to. The game must always end with both players having had an equal number of turns; if the first player gets a winning total score, the second player must get one last turn. If the second player gets a winning total score, the first player will not get another turn.

SAMPLE SESSION

Here is what your program *may* look like when you run it (user input is highlighted):

```
Let's Play Sprock!

Enter player 1's name: Elon
Enter player 2's name: Jeff

How many points are we playing to? 50

Elon's turn 1

Enter R to roll 5 dice: R

Your dice are: 6 2 4 1 4
The total is 17, and the average is 3.4

Elon is safe! Elon's turn score is now 17.
```

Elon starts with 5 dice

Elon is safe, and gets to choose whether to roll again or to pass 4 dice to the next player.

```
Enter R to roll again, or P to pass: R
```
He chooses to roll again.

```
Your dice are 3 4 5 2
The total is 14, and the average is 3.5
```
Safe again!

```
Elon is safe! Elon's turn score is now 31.
Enter R to roll again, or P to pass: P
```
This time Elon chooses to save his points, end his turn, and pass his remaining 3 dice to Jeff.

```
Jeff's turn 1

Do you want to start from zero (Z), or with Elon's points (P) and 3 dice? P

Your dice are 3 2 1
The total is 6, and the average is 2.0
```
Jeff chooses to continue where Elon left off, but is unlucky and ends with 0 points this turn.

```
Sprock! You get 0 points.

At the end of round 1, the score is Elon: 28 and Jeff: 0

Elon's turn 2

Enter R to roll 5 dice: R

Your dice are: 6 3 3 1 1
The total is 14, and the average is 2.8
```

…play should continue until one of the players reaches 50 points.

HINTS

It would be highly advisable to write this program a bit at a time, making sure that each part works as you build it. Here are some individual pieces that you can focus on:

- write functions to show the game intro, to get the user information, and to get the point target for the game
- write a function that uses random number generation to simulate the rolling of one die
- write a function that takes a number as input, and simulates the rolling of that number of dice, showing their values and returning the total
- write a function that plays one round for a player; the round will start with a certain number of dice and a certain number of points; the function will contain a loop so a player can keep rolling dice while they choose to do so, or until they lose their turn due to Sprocking out
- add code to handle a second player, so they can have their turn after the first player
- write the loop that determines when the game is over

As you write the functions, you should write code to test each function to make sure it is working properly.