CS1010E Programming Methodology

Semester 1 2016/2017

Week of 26 – 30 September 2016

Tutorial 5 Suggested Answers

Value-Returning Functions

1. In tutorial #1, we performed a trace of a program containing only the `main` function. Extend your *mental model* to include traces of program execution involving more than one function using the program below.

```
#include <stdio.h>

int f(int x, int y);

int main(void) {
   int x = 3, y = 4;

   x = f(x,y);
   y = f(x, f(y,x));
   printf("x = %d; y = %d\n", x, y);
   return 0;
}

int f(int x, int y) {
   return x*10 + y;
}
```

   Keep in mind the following notions while you trace.

   - function call with evaluated arguments
   - function activation with parameter declaration
   - pass-by-value
   - lexical scoping
   - function termination upon return

   Refer to the slideshow ***`fModel.ppsx`****.*

2. In the following parts, you are required to define functions to find the maximum of a set of integer values. *Note that some of the values might be the same.* Devise your own `main` function to test the correctness of each of the functions.

   (a) Define a function `max2` that takes in two values a and b, and returns the maximum of the two values.

   ```
   int max2(int a, int b);
   ```

   (b) Define a function `max3` that takes in three values a, b and c, and returns the maximum of the three values. Use the `max2` function defined in part 2a above.

   ```
   int max3(int a, int b, int c);
   ```

```
#include <stdio.h>

int max2(int a, int b);
int max3(int a, int b, int c);

int main(void) {
   int a, b, c;

   printf("Enter three numbers: ");
   scanf("%d %d %d", &a, &b, &c);

   printf("Max of %d and %d is %d\n", a, b, max2(a,b));
   printf("Max of %d, %d and %d is %d\n", a, b, c, max3(a,b,c));

   return 0;
}

/*
   Function max2 returns the maximum of two arguments a and b.
   Pre-condition: none.
*/
int max2(int a, int b) {
   if (a > b) {
      return a;
   } else {
      return b;
   }
}

/*
   Function max3 returns the maximum of three arguments a, b and c.
   Pre-condition: none.
*/
int max3(int a, int b, int c) {
   return max2(max2(a,b),c);
}
```

3. The following are some guiding principles when defining functions.

   - A function should be reusable.
   - A function should perform a single well-defined task.
   - A function should rely on minimal input to do its work.

   Determine whether the following functions meet the above criteria.

   (a)
   ```
   double areaCircle(void) {
      double radius, area;

      printf("Please enter radius of circle: ");
      scanf("%lf", &radius );

      area = 3.14159 * radius * radius;

      return area;
   }
   ```
   In general, avoid performing input (`printf`) / output (`scanf`) together with computation in a single function. To make the function more reusable, "input" and "output" should be implemented via the function input parameters and the function return value respectively. This will also allow the function to be used in the following situations:

   - When the area computation is part of a more complex computation, `volume = areaCircle(radius) * height;`
   - When the radius is the result of a computation, `area = areaCircle(diameter/2);`

   (b)
   ```
   double areaCircle(double radius, double area) {
      area = 3.14159 * radius * radius;

      return area;
   }
   ```
   The area of a circle should be a function over radius only, since $f(r) = \pi r^2$. Input parameters should not hold irrelevant information.

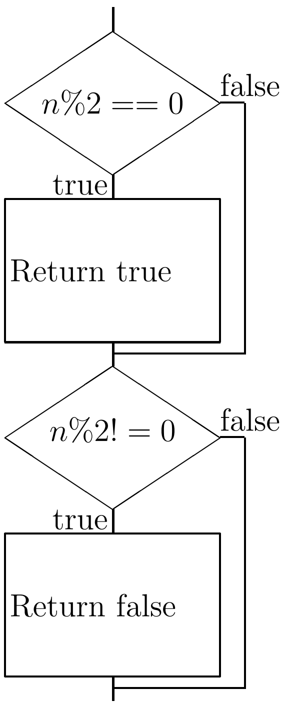
4. We would like to define the `isEven` function to determine if a given integer n is even. Two function implementations are given below.

   (a)
   ```
   bool isEven(int n) {
      if (n%2 == 0) {
         return true;
      } else {
         return false;
      }
   }
   ```

(b)
```
bool isEven(int n) {
    if (n%2 == 0) {
       return true;
    }
    if (n%2 != 0) {
       return false;
    }
}
```

By writing a suitable `main` function, test if the above functions work. Rewrite the `isEven` function such that it makes use of only one return statement.

The function in part 4b compiles with a warning when using `gcc -Wall` *as there is no complete path coverage, i.e. all possible control flow paths must terminate with a return statement. Notice from the function control flow below that if both condition checks are false, then the path does not lead to any valid return statement.*



```
#include <stdio.h>
#include <stdbool.h>

bool isEven(int n);

int main(void) {
   int n;

   printf("Enter a number: ");
   scanf("%d", &n);

   if (isEven(n)) {
      printf("%d is even\n", n);
   } else {
      printf("%d is odd\n", n);
   }

   return 0;
}

/*
   isEven function returns true when n is even, and false otherwise.

   Precondition: none
*/
bool isEven(int n) {
   return (n%2 == 0);
}
```

5. Credit card numbers are typically 16-digits long and must conform to the Luhn Algorithm which is also known as the modulus-10 algorithm. We illustrate the algorithm using the credit card number: 8765 4321 1357 2468

8	7	6	5	4	3	2	1	1	3	5	7	2	4	6	8
Double the values of every alternating digit starting from the second digit from the right.															
16	7	12	5	8	3	4	1	2	3	10	7	4	4	12	8
Add together the individual single digits of all the numbers obtained above.															
7	7	3	5	8	3	4	1	2	3	1	7	4	4	3	8
Sum up all the numbers:															
$7+7+3+5+8+3+4+1+$ $2+3+1+7+4+4+3+8=70$															

If the last digit of the Luhn sum is zero, then the credit card number is a valid one. Your task is to write a program that reads in a 16-digit credit card number as input from the user and outputs the last digit of the corresponding Luhn sum. The credit card number is to be read as four 4-digit numbers. *Why not just read a single 16-digit number?.*

(a) Notice that for each 4-digit number the method to compute the partial sum is the same. Write a function `partialSum` that takes in a 4-digit number `num` as argument and returns the partial sum of that 4-digit number.

```
int partialSum(int num);
```

(b) Write a function `luhnSum` that takes in a credit card number as four 4-digit numbers and computes the corresponding Luhn sum.

```
int luhnSum(int num1, int num2, int num3, int num4);
```

(c) Write a `main` function to read in the 16-digit credit card number and outputs the last digit of the corresponding Luhn sum. Sample runs are given below. User input is underlined.

- Sample run #1:
```
Enter credit card number: 8765 4321 1357 2468
The last digit of the Luhn sum is 0.
```

- Sample run #2:
```
Enter credit card number: 1234 5678 8765 4321
The last digit of the Luhn sum is 2.
```

```c
#include <stdio.h>

int luhnSum(int, int, int, int);
int partialSum(int);

int main(void) {
   int num1, num2, num3, num4, lastDigit;

   printf("Enter credit card number: ");
   scanf("%d %d %d %d", &num1, &num2, &num3, &num4);

   lastDigit = luhnSum(num1, num2, num3, num4) % 10;

   printf("The last digit of the Luhn sum is %d.\n", lastDigit);

   return 0;
}

/*
   Function luhnSum computes the sum using the Luhn Algorithm for
   16-digit credit card number represented as four 4-digit numbers:

   num1 num2 num3 num4
*/
int luhnSum(int num1, int num2, int num3, int num4) {
   return partialSum(num1) + partialSum(num2) +
          partialSum(num3) + partialSum(num4);
}

/*
   Function partialSum returns the Luhn sum of 4 digits d1d2d3d4.
 */
int partialSum(int num) {
   int d1, d2, d3, d4;

   d1 = num / 1000 * 2;
   d2 = (num / 100) % 10;
   d3 = (num / 10) % 10 * 2;
   d4 = num % 10;

   d1 = d1 / 10 + d1 % 10;
   d3 = d3 / 10 + d3 % 10;

   return d1 + d2 + d3 + d4;
}
```

6. (a) Write a function `getDistance` to compute and return the euclidean distance between two locations given by their Cartesian coordinates (x_1, y_1) and (x_2, y_2). Use the distance formula

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Declare suitable parameters to accept the two locations. Note that each location is given by a pair of real numbers.

(b) Everyday, Mr. Get A. Life will drive from home to office in the morning. After work, he makes a trip to the market for groceries before returning home. Suppose all three locations (home, office and market) are represented as Cartesian coordinates. Using the function developed in 6a, write a program to calculate the total distance traveled by Mr. Get A. Life.

A sample run is given as follows. User input is underlined.

```
Enter x and y coordinates for home: 1.5 2.0
Enter x and y coordinates for office: 4.25 5.75
Enter x and y coordinates for market: 7.0 10.20

The total distance is 19.755128
```

```
#include <stdio.h>
#include <math.h>

double getDistance(double xStart, double yStart,
                   double xEnd, double yEnd );

int main(void)
{
   double xHome, yHome;
   double xOffice, yOffice;
   double xMarket, yMarket;
   double dist1, dist2, dist3, total;

   printf("Enter x y coordinates for home: ");
   scanf("%lf %lf", &xHome, &yHome);

   printf("Enter x y coordinates for office: ");
   scanf("%lf %lf", &xOffice, &yOffice);

   printf("Enter x y coordinates for market: ");
   scanf("%lf %lf", &xMarket, &yMarket );

   dist1 = getDistance(xHome, yHome, xOffice, yOffice);
   dist2 = getDistance(xOffice, yOffice, xMarket, yMarket);
   dist3 = getDistance(xMarket, yMarket, xHome, yHome);
   total = dist1 + dist2 + dist3;

   printf("\nThe total distance is %f\n", total);

   return 0;
}

/*
   Function getDistance computes the euclidean distance between
   (xStart,yStart) and (xEnd,yEnd).

   Precondition: none.
*/
double getDistance(double xStart, double yStart,
                   double xEnd, double yEnd )
{
   double dist, xDiff, yDiff;

   xDiff = xStart - xEnd;
   yDiff = yStart - yEnd;
   dist = sqrt(xDiff*xDiff + yDiff*yDiff);

   return dist;
}
```