APS 105 — Computer Fundamentals

Lab #4: Loops and Functions
Winter 2023

You must use `examify.ca` to electronically submit your solution by 11:59 pm on **Saturday, February 11, 2023**.

Objective

In this lab, you will be writing two C programs to demonstrate Kaprekar's Constant and graph functions. In a third exercise, you are asked to fix a given program. Input and output must be done using `scanf` and `printf`.

In the sample output examples that follow, the text that would be entered by the program user is displayed using a **bold** font. The text <**enter**> stands for the user pressing the `enter` key on the keyboard. On some keyboards, the `enter` key may be labeled `return`. When you execute your programs, the user's text will not be displayed in bold and <**enter**> will not be shown.

Once again, place your solution for each part in a separate directory, so that Visual Studio Code will not attempt to compile multiple files and link them together. Each of your directories should have one `.c` file corresponding to your solution for the individual part of the exercise. Please ensure you are selecting the folder with the `.c` you want to compile.

Part 1 — Kaprekar's Constant

6174, better known as "Kaprekar's Constant", is a mathematical phenomenon that applies to four-digit numbers. This process is as follows:

1. Take any four-digit number with at least two different digits.
2. Arrange the digits in descending and then in ascending order to get two new four-digit numbers (leading zeros are allowed).
3. Subtract the smaller number from the larger number.
4. Repeat the process from step 2 using the newly calculated number.

After performing this a few times, you will quickly realize something amazing: the process always converges to the number 6174 and remains there forever. An example of this process is as follows (starting with the number 8273):

Starting number	**Sorted digits (descending)**	**Sorted digits (ascending)**	**Difference**
8273	8732	2378	6354
6354	6543	3456	3087
3087	8730	0378	8352
8352	8532	2358	6174
6174	7641	1467	6174

Your task is to write a program which takes as input a four-digit number and prints the steps involved in converting that number to Kaprekar's Constant. We recommend you define a few functions. The prototypes for these functions are as follows:

```
    // This function prompts the user to enter a number
    // and returns the entered number
    int getNumber();

    // This function reverses the digits of the given number
    int reverseNumber(int);
```

To assist you, a helpful utility function has already been defined which you may use in your lab. This function takes as input a number and returns a number containing the same digits as the input number, though in ascending order. The prototype of this function is as follows:

```
    int getAscendingOrderedDigits(int);
```

Function Implementation (DO NOT SUBMIT THE FUNCTION IMPLEMENTATION ON EXAMIFY!!!)

```
int getAscendingOrderedDigits(int number) {
  int digits[10] = {0};

  while (number > 0) {
    digits[number % 10]++;
    number /= 10;
  }

  int i = 0;
  while (i < 10) {
    if (digits[i] > 0) {
      number *= 10;
      number += i;
      digits[i]--;
    } else {
      i++;
    }
  }

  return number;
}
```

Examples

Shown below is a sample run of the program. Data shown following the colons are supplied by the user; all other items are produced by the program. Your program should match the following patterns *exactly*, including all punctuation. Any variation from this will result in a loss of marks.

```
    Enter a number (-1 to stop): 8273
    Number 1: 8273
    Number 2: 6354
    Number 3: 3087
    Number 4: 8352
```

```
Number 5: 6174
Kaprekar's Constant was reached in 4 iteration(s).
Enter a number (-1 to stop): 6174
Number 1: 6174
Kaprekar's Constant was reached in 0 iteration(s).
Enter a number (-1 to stop): 327
Error: the number is too small.
Enter a number (-1 to stop): 12345
Error: the number is too large.
Enter a number (-1 to stop): -1
```

Checking for Bad Data

You should check for the following forms of bad data. The error messages that should be produced in these cases are shown in the examples.

- A number with fewer or more than four digits was entered

You may assume that

- The user will supply the correct number of values
- The user will not enter any same digit numbers, e.g. 1111, 2222, etc.
- The user will supply positive integers
- No result will ever be larger than the largest `int` value

Part 2 — Graphing

In this question, you will write a program to graph polynomial functions. Your program will graph functions up to order 3. These are:

- Order 0: $f(x) = C$
- Order 1: $f(x) = mx + b$
- Order 2: $f(x) = ax^2 + bx + c$
- Order 3: $f(x) = ax^3 + bx^2 + cx + d$

Your program must **repeatedly** prompt the user to enter the order of the function. This must be a value between $[0, 3]$. The program must then ask the user to input the coefficients of the polynomial function. Finally, the program must plot the function's points using the character " * ", drawing a Cartesian grid between x and y $[-10, 10]$. **If an x coordinate generates a y coordinate that is outside the range [-10, 10], the program should not print that point.** The x-axis is drawn using a " - " and the y-axis is drawn using a " | ". If no point or axis is to be printed three spaces are printed. **If the user enters an order of -1, the program should terminate.**

Examples

Shown below is a sample run of the program. Data shown following the colons are supplied by the user; all other items are produced by the program. Your program should match the following patterns *exactly*, including all punctuation. Any variation from this will result in a loss of marks.

```
Enter the order of the function: 3
Enter coefficient of x^0: 1
Enter coefficient of x^1: 2
Enter coefficient of x^2: 3
Enter coefficient of x^3: 1
                               |
                               |
                               |
                               |  *
                               |
                               |
                               |
                               |
                               |
                         *  *  *
 -  -  -  -  -  -  -  -  -  -  |  -  -  -  -  -  -  -  -  -  -
                               |
                               |
                               |
                               |
                      *        |
                               |
                               |
                               |
                               |
                               |
Enter the order of the function: 2
Enter coefficient of x^0: 0
Enter coefficient of x^1: 1
Enter coefficient of x^2: 2
                               |     *
                               |
                               |
                               |
                         *     |
                               |
                               |
                               |  *
                               |
                            *  |
 -  -  -  -  -  -  -  -  -  -  *  -  -  -  -  -  -  -  -  -  -
                               |
                               |
                               |
                               |
                               |
                               |
                               |
                               |
                               |
                               |
Enter the order of the function: 0
Enter coefficient of x^0: 5
                               |
                               |
                               |
                               |
                               |
 *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
                               |
                               |
                               |
                               |
 -  -  -  -  -  -  -  -  -  -  |  -  -  -  -  -  -  -  -  -  -
                               |
                               |
                               |
                               |
                               |
                               |
                               |
                               |
                               |
                               |
Enter the order of the function: -2
The order must be between [0, 3].
Enter the order of the function: 4
The order must be between [0, 3].
Enter the order of the function: -1
```

Checking for Bad Data

You should check if the user enters a function order less than 0 or greater than 3. If this is the case, you should print the following error message: "The order must be between [0, 3]."
You may assume that

- The user will supply the correct number of coefficients
- No y-coordinate will ever be larger than the largest `int` value

Part 3 — Debugging

Rea, Dami and Hira are working for a Telecommunications company. They developed a function that checks if a number is a shuffled version of an ordered number. For example, **917865432** is a shuffled number of the ordered **123456789** number. The function is supposed to return `true` if a number is shuffled, and `false` if the number is missing one or more numbers. It also returns `false` if a number is repeated more than once. The function is tested in the main function as shown below, and in **lab4par3-debugging-exercise.c** file on Quercus. However, the function is not working as expected. Your task is to debug the program and fix the errors.

Assume the user will not enter negative numbers or numbers with 0 digits.

Hint: As you may have observed the program is long, and it can be difficult to know the source of errors. As a hint, we are telling you that **some** of the bugs are in `getNumDigits` function. We suggest you isolate it and test it separately in a different `.c` file. The following is an example of a program you can use to test "`getNumDigits`" function. As you can see we are passing different values to `getNumDigits` to check if it works. Once you fix `getNumDigits` function, check `getSmallestDigit` and `lookForDigit` separately. Then, finally when you fix all the functions, check `isShuffled` function using the entire program. This is the benefit of using functions. You can break your code into pieces, and easily fix issues in each function regardless of the entire program.
Program to test `getNumDigits` function.

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

int getNumDigits(int);

int main(void) {
  int num = 123;
  printf("getNumDigits(%d) = %d\n", num, getNumDigits(num));
  num = 98765421;
  printf("getNumDigits(%d) = %d\n", num, getNumDigits(num));
  num = 76532;
  printf("getNumDigits(%d) = %d\n", num, getNumDigits(num));
  return 0;
}

int getNumDigits(int num) {
  int count = 1;
  while (num <= 0) {
    num = num / 10;
    count++;
  }
  return count;
}
```

The program with all functions that you are required to fix.

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

```
#include <stdio.h>

bool isShuffled(int);
int getNumDigits(int);
int getSmallestDigit(int);
bool lookForDigit(int, int);

int main(void) {
  int num;
  printf("Enter the number to check: ");
  scanf("%d", &num);
  if (isShuffled(num)) {
    printf("Shuffled!");
  } else {
    printf("Not shuffled!");
  }
  return 0;
}

bool isShuffled(int shuffledOrder) {
  bool isShuffled = true;
  int numOfDigits = getNumDigits(shuffledOrder);
  int smallestDigit = getSmallestDigit(shuffledOrder);

  for (int place = 1; place <= numOfDigits && isShuffled; place++) {
    if (!lookForDigit(shuffledOrder, smallestDigit + place)) {
      isShuffled = false;
    }
  }
  return shuffledOrder;
}

int getNumDigits(int num) {
  int count = 1;
  while (num <= 0) {
    num = num / 10;
    count++;
  }
  return count;
}

int getSmallestDigit(int num) {
  int smallestDigit = 0;
  int digit = 0;
  while (num != 0) {
    digit = num % 10;
    if (digit < smallestDigit) {
      smallestDigit = digit;
    }
    num /= 10;
    }
    return digit;
  }

  bool lookForDigit(int num, int searchDigit) {
    int digit = 0;
    bool foundDigit = false;
    while (num != 0 || foundDigit) {
      digit = num % 10;
      if (digit == searchDigit) {
        foundDigit = true;
      }
      num /= 10;
    }
    return digit;
  }
```

Example outputs of functioning code.
Example 1

```
Enter the number to check: 679812453
Shuffled!
```

Example 2

```
Enter the number to check: 876523451
Not shuffled!
```

Example 3

```
Enter the number to check: 123456789
Shuffled!
```

Example 4

```
Enter the number to check: 324
Shuffled!
```

Example 5

```
Enter the number to check: 8769
Shuffled!
```

Marking

This lab will be marked out of 10, 3 marks for Part 1, 4 marks for Part 2 and 3 marks for Part 3. The automarker may use multiple test cases in each part, and if this is the case, the marks for this part will be evenly split between the test cases. The test cases used for marking may or may not be the same as the test cases that are made available to you. The deadline is strictly enforced **(11:59 pm on Saturday, February 11, 2023)**, so avoid last minute submissions.