# *ENGG1410: Introductory Programming for Engineers*
# Lab 5: "Introduction to C Programming": Functions

School of Engineering, University of Guelph
Fall 2024

**Start Date: Week #7 2024**
**Duration: 1 Week**

## Objectives

The objective of this lab is to enhance your programming skills by developing a C program that performs arithmetic operations on numbers written in different bases. You will learn to implement modular code using functions, validate inputs, convert between numerical bases, and handle various operations (addition, subtraction, multiplication, and division).

## Instructions

In this lab, you are required to implement a single C program that:

1. Uses functions to organize the code into logical components for each task:
   - A function for input validation to ensure correct operator, base, and operand formats.
   - Functions for converting numbers between bases (e.g., from any base to base 10 and vice versa).
   - Functions for each arithmetic operation (addition, subtraction, multiplication, and division).
2. Repeatedly prompts the user to input values in the following format:
   - A single character for the operation (+, -, *, /, or $).
   - An input base (from 2 to 10, inclusive).
   - Two operands, each represented in the specified input base.
   - An output base (from 2 to 10, inclusive).
3. Validates the inputs:
   - Ensure the operator is one of the allowed characters (+, -, *, /, or $).
   - Check that the input and output bases are within the specified range (2 to 10).
   - Verify that the operands contain valid digits for the specified base.
4. Converts the operands to base 10, performs the specified operation, converts the result to the output base, and prints the results.
5. Continues processing until the user inputs the character '$' to terminate the program.

**Additional Requirements:**

- Your program should include clear and descriptive comments for each function, explaining its purpose, input parameters, and output.
- Use appropriate error messages to guide the user if invalid inputs are detected.
- Ensure the program output is clear, well-formatted, and follows the structure provided in the examples.

## 1 Programming — Numerical Bases

Write a program that performs the four basic arithmetic operations (addition, subtraction, multiplication, and division) on pairs of positive integers written in one base and produces answers as numbers written in a (possibly) different base. The program should repeatedly prompt the user to supply input in the following order:

1. A single character representing an operation: one of the characters +, -, *, /, or \$. The \$ signals the end of the data; reading it should cause the program to terminate.
2. An input base: 2 to 10 (inclusive).
3. A first operand: a positive integer written as a numeral in the input base.
4. A second operand: a positive integer written as a numeral in the input base.
5. An output base: 2 to 10 (inclusive).

### Example 1

Suppose the following inputs are provided:

```
Give input ($ to stop): + 4 201 2312 5
201 + 2312 (base 4) = 33 + 182 = 1330 (base 5)
Give input ($ to stop): * 2 1101 101 8
1101 * 101 (base 2) = 13 * 5 = 101 (base 8)
Give input ($ to stop): / 10 743 18 4
743 / 18 (base 10) = 743 / 18 = 221 (base 4)
Give input ($ to stop): # 8 27 15 6
Invalid operator
Give input ($ to stop): - 4 23 11 15
Invalid base
Give input ($ to stop): * 3 27 12 8
Invalid digits in operand
Give input ($ to stop): + 10 44 67 10
44 + 67 (base 10) = 44 + 67 = 111 (base 10)
Give input ($ to stop): $
```

### Checking for Bad Data

The program should check for the following forms of invalid input:

- An invalid operator, e.g., # or @.
- A base outside the range 2 to 10.
- An operand having digits too large for the given input base, e.g., 45 in base 3 (where the largest digit should be 2).

If more than one invalid input is entered, the program should prioritize error detection in the following order:

1. Operator
2. Input or output base
3. Operands

### Example 2

```
Give input ($ to stop): * 3 4 2 15
Invalid base
Give input ($ to stop): * 3 4 2 8
Invalid digits in operand
Give input ($ to stop): * 3 2 2 8
2 * 2 (base 3) = 2 * 2 = 4 (base 8)
Give input ($ to stop): $
```

### Example 3

```
Give input ($ to stop): - 6 205 43 7
205 - 43 (base 6) = 77 - 27 = 101 (base 7)
Give input ($ to stop): & 9 74 57 10
Invalid operator
Give input ($ to stop): / 9 74 57 10
74 / 57 (base 9) = 67 / 52 = 1 (base 10)
Give input ($ to stop): % 2 10 01 8
Invalid operator
Give input ($ to stop): + 2 10 01 8
10 + 1 (base 2) = 2 + 1 = 3 (base 8)
Give input ($ to stop): $
```

### Assumptions

- The user will supply the correct number of values.
- The user will supply positive integers.
- No result will ever be larger than the largest `int` value.
- All operations will produce valid, non-negative results.

## Demo and Report Submission

Demonstrate the correct operation of at least two functions to your TA and be readiy to answer the TA quesitons about your code. Submit a report of the lab in DropBox for evaluation. Include your `.c` files containing your source code for the diffrent programming parts. Your code should include appropriate comments explaining the implementation of the algorithms used. The file should have the group number, names of group members, date, and any additional notes at the top as comments.

### General Demo and Report Evaluation Criteria

Herefollows a detailed criteria for assessing the students demonstrations, reports and projects. Each criterion focuses on a key aspect of software development, aiming to ensure that students not only complete the tasks but also adhere to best practices in coding and project documentation.
**Note:***Depending on the lab specifics some of the below points might be excluded.*

- **Correctness of Algorithm Implementation:**

  - The program must correctly implement the algorithm as described in the project specifications.
  - All functionalities outlined must be fully operational without errors during the demo.
  - Special cases and edge conditions should be correctly handled by the algorithm.

- **Proper Handling of Input Validation:**
  - The program should include robust input validation to reject invalid inputs without crashing or producing incorrect results.
  - Error messages should be informative and guide the user towards providing correct input.
  - All inputs in the demo must be shown to be properly validated to prevent data type errors, out-of-bound errors, or any other type of input-related issues.

- **Clarity of the Output Provided to the User:**
  - Output must be clear, precise, and correctly formatted according to project specifications.
  - The program should provide outputs that are easy to understand and logically presented, with labels or explanations if necessary.
  - Any numerical results should be displayed with appropriate units and an adequate number of significant figures.

- **Code Readability and Documentation:**
  - The source code must be neatly formatted with consistent indentation and clear, logical organization.
  - Comments should be used to explain the purpose of functions, the logic behind critical sections of code, and any complex algorithms.
  - The documentation provided (either as comments in the code or in separate documentation files) should clearly explain the program's structure, provide a usage guide, and list all dependencies.