

## Project 2, Part 1, Program Design

Write a program that displays menus for converting length and calculates the result. The program should support the following conversions:

Miles	Kilometers	1.6093
Kilometers	Miles	0.6214
Inches	Centimeters	2.54
Centimeters	Inches	0.3937

1. Display the menu options as numbers.
  - 1 - Miles to Kilometers
  - 2 - Kilometers to Miles
  - 3 - Inches to Centimeters
  - 4 - Centimeters to Inches
2. Use a switch statement for option selection and calculation.
3. If the entered option is not in the range of 1 to 4, display an error message and abort the program.
4. The output should display two digits after the decimal point. For example, 3.23.

### Before you submit

1. Compile both programs with `-Wall`. `-Wall` shows the warnings by the compiler. Be sure it compiles on **circe** with no errors and no warnings.

```
gcc -Wall convert.c
```

2. Be sure your Unix source file is read & write protected. Change Unix file permission on Unix:

```
chmod 600 convert.c
```

3. Test the program with the shell script `try_convert` on Unix:

```
chmod +x try_convert
```

```
./try_convert
```

4. Submit `convert.c` on Canvas.

## Grading

Total points: 50

1. A program that does not compile will result in a zero for that program.
2. Runtime error and compilation warning 5%
3. Commenting and style 15%
4. Functionality 80%

## Programming Style Guidelines

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

1. Your program should begin with a comment that briefly summarizes what it does. This comment should also include your name.
2. In most cases, a function should have a brief comment above its definition describing what it does. Other than that, comments should be written only *needed* in order for a reader to understand what is happening.
3. Variable names and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.
4. Use consistent indentation to emphasize block structure.
5. Full line comments inside function bodies should conform to the indentation of the code where they appear.
6. Macro definitions (`#define`) should be used for defining symbolic names for numeric constants. For example: `#define PI 3.141592`
7. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
8. Use either underscores or capitalization for compound names for variable: `tot_vol`, `total_volumn`, or `totalVolumn`.