

Project 1, Program Design

Write a C program that asks the user to enter a three-digit integer and then replace each digit by (the sum of that digit plus 6) modulus 10. If the integer entered is less than 100 or greater than 999, output an error message and abort the program. A sample input/output:

```
Enter a three-digit number: 928
Output: 584
```

Note: you may need an if/else statement, the syntax for an if/else statement in C is the following:

```
if (expression ) statement1
    else statement2
```

Before you submit:

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

```
gcc -Wall replace.c
```

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

```
chmod 600 replace.c
```

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

```
chmod +x try_replace
./try_replace
```

4. Submit *replace.c* on Canvas>Assignments.

Grading

Total points: 100

1. A program that does not compile will result in a zero.
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 underscores to make compound names easier to read: **`tot_vol`** or **`total_volumn`** is clearer than `totalvolumn`.