Project 5, Program Design

1. (30 points) Note: This program will be graded based on whether the required functionality were implemented correctly instead of whether it produces the correct output, for the functionality part (80% of the grade).

Modify `barcode.c` (attached, Project 4, #2), the edge detection function using pointer arithmetic. The function prototype should be the following. Name your program **`barcode2.c`**.

```
void edge(int n, int *a1, int *a2);
```

The function should use pointer arithmetic – not subscripting – to visit array elements. In other words, eliminate the loop index variables and all use of the [] operator in the function.

2. (70 points) Write a C program that asks the user to enter a positive integer (the integer could be of any number of digits in the range of the integer type) and replace each digit by *the sum of that digit plus 6 modulus 10*. The program then should swap the first digit with the last digit before it displays the output. A sample input/output:

```
Enter the number of digits of the number: 5
Enter the number: 92828
Output: 48485
```

1) Name your program **`replace2.c`**.
2) The user will enter the total number of digits before entering the number.
3) You can use format specifier "%1d" in scanf to read in a single digit into a variable (or an array element). For example, for input 101011, scanf("%1d", &num) will read in 1 to num.
4) As part of the solution, write and call the function `replace()` with the following prototype. The `replace()` function assumes that the digits are stored in the array `a` and computes the replaced digits and store them in the array `b`. c represents the size of the arrays.

   ```
   void replace(int *a, int *b, int n);
   ```

   The `replace()` function should use pointer arithmetic – not subscripting – to visit array elements. In other words, eliminate the loop index variables and all use of the [] operator in the function.

5) As part of the solution, write and call the function `swap()` with the following prototype.

   ```
   void swap(int *p, int *q);
   ```

   When passed the addresses of two variables, the `swap()` function should exchange the values of the variables:

   swap(&i, &j); /* exchange values of i and j */

Before you submit:

1. Compile with –Wall. Be sure it compiles on ***circe*** with no errors and no warnings.

gcc –Wall barcode2.c
gcc –Wall replace2.c

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

chmod 600 barcode2.c
chmod 600 replace2.c

3. Test your program with the shell scripts on Unix:

chmod +x try_barcode
./try_barcode

chmod +x try_replace2
./try_replace2

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%

Problem #1: The function should use pointer arithmetic – not subscripting – to visit array elements. (80%)
Problem #2: The replace() function should use pointer arithmetic – not subscripting – to visit array elements. (40%)
The swap function implemented as required. (20%)

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. Information to include in the comment for a function: name of the function, purpose of the function, meaning of each parameter, description of return value (if any), description of side effects (if any, such as modifying external variables)
4. 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.
5. Use consistent indentation to emphasize block structure.
6. Full line comments inside function bodies should conform to the indentation of the code where they appear.
7. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: **`#define PI 3.141592`**
8. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
9. Use underscores to make compound names easier to read: **`tot_vol`** or **`total_volumn`** is clearer than totalvolumn.