Assignment 3: Line Editor/Encoder

This is a small project designed to get you accustomed to writing programs that use non-canonical input mode, to prepare you for the more interesting problems of writing interactive applications such as games, simulations, and other user-driven applications. It is a simple line-editing program that operates in non-canonical mode with echoing disabled, that lets the user enter and edit a line of text, and encodes parts of the text using the encryption method described below.

This program is specified with a descriptive summary rather than a man page.

Description

The program is named `encode`. The program must is started by typing its name. It can be used with standard output redirected

```
encode > output-file
```

When the program starts, the user's key-presses will be handled by your program, which has turned off canonical processing and echoing. Certain keys are just data, to be passed to the program's "processing engine". Other keys perform editing or control functions. The editing/function keys are:

Ctrl-C — If encoding is enabled, disable it so that the next sequence will not be encoded. If encoding is disabled, enable it, so that the next sequence is encoded.

Ctrl-U — Erase the entire line of characters.

Ctrl-W — Erase the right-most word. A word is a sequence of characters other than space and tab. If the rightmost characters at the cursor position are non-word characters, then they are removed and the line is scanned backwards until a word is found, and that word is removed. If no word is found, then all non-word characters are removed up to the start of the line.

`Backspace` — Erase the preceding character.

`Enter` — Append the entered line of text to the output file. The displayed line may contain a mixture of encoded and unencoded text.

Ctrl-\ — Terminate the program.

All other keys are to be processed by the program normally.

Notice that because Ctrl-C is used as a control character by the program, it cannot be used to terminate your program. You must modify the terminal attributes to prevent Ctrl-C from doing this. Since Ctrl-C cannot terminate your program, the program can only be terminated by Ctrl-\. Not even Ctrl-D can terminate it.

Input and Output

Whether or not encoding is enabled, all non-editing characters typed by the user are displayed on the screen exactly as the user enters them. (This way the user can verify that he or she did not make a typing mistake.) For example, if the user types `abcdefg` then `Backspace` then `123`, the current line will display `abcdef123`.

When the user types the `Enter` key, all of the text in the current line must be written to the standard output. If standard output is not redirected, then it should appear below the typed line on the screen. Of course if

standard output is redirected, then the output is instead appended to the file to which it is redirected. Your program does not have to do this – the shell handles that.

When the user enters a Ctrl-C and encoding is disabled, encoding becomes enabled. Whatever characters are typed from that point in time are to be encoded in the output when the Enter key is pressed. If the user types `abcde` then this will be encoded on output. When the user types Ctrl-C again, encoding is disabled.

To illustrate, suppose the user starts typing a new line and types `12345`, then types Ctrl-C, then types `6789`, then types Ctrl-C and then `Enter`. Then the output will be `12345` followed by the encoding of `6789`. Now suppose instead that the user typed Ctrl-C at the beginning, then `12345`, then Ctrl-C (disabling encoding), then backspaced twice and then typed `6789`, then `Enter`. In this case the output would be the encoding of `123` followed by the string `6789`.

If you think about how to implement this, you will realize that you need a buffer to store the characters, and a way to keep track of which parts of the buffer get encoded when the buffer is output, and which parts are clear text on output.

Although the program can have standard output redirected, it should fail if standard input is redirected. It should detect when its standard input is not the terminal and quit with an error message.

Encoding

The encoding that we will use is not a particular secure encoding. The purpose of this assignment is not to write a secure encryption program. It is designed to be easily decoded.

Recall that the C bitwise-exclusive-or operator is the caret "`^`". Suppose the string to be encoded is the character array `a[0..n-1]`. Then the encoded string `b[0..n-1]` will be created as follows. Let `key` be a character whose leading bit is 0 (otherwise you will get bad results.) All ASCII characters have this property.

```
b[0] = a[0] ^ key;
b[1] = b[0] ^ a[1];
b[2] = b[1] ^ a[2];
...
b[n-1] = b[n-2] ^ a[n-1];
```

The characters that are stored into the string `b` may not be printable. For example, if the key is the character '`A`' and the input character is also '`A`', then the output character will be the character with value 0, i.e., the `NULL` byte. This will display with one of those glyphs that look like dominoes. But that does not matter because the correct character code will be stored into the output string.

The key will be supplied to the encoding function as a separate argument. The function signature requires four arguments as follows:

```
encode_str(char *input_string, char key, char *output_string, int length);
```

where `input_string` and `output_string` are pointers to memory allocated by the caller, `key` is the key for encoding, and `length` is the number of characters in the input string.

Suppose `line[0..n-1]` is a line is to be printed to standard output. Suppose that `line[j..k]`, `0 <= j <= k <= n-1`, is a substring of `line` that has to be encoded. Then the key to be given to the `encode_str()` function should be the character `line[j-1]` if `j > 0`, and the character `'\0'` if `j == 0`. (If the key is 0, the character will be encoded as itself.)

The nice thing about this encoding method is that you can decode it easily with the following algorithm:

```
a[0] = b[0] ^ key;
a[1] = b[0] ^ b[1];
a[2] = b[1] ^ b[2];
...
a[n-1] = b[n-2] ^ b[n-1];
```

If you redirect standard output to a file, you can use a decoder like this to check whether you get back your original string.

Submitting the Assignment

You are to create a zip file containing all of your source code and put that zip file in the directory

```
/data/biocs/b/student.accounts/cs493.66/projects/project3
```

naming it `username_hwk3.zip`. Give it permissions 600 so that only you have access to it. Whether you have a single file or multiple files, you are to create a directory named `username_hwk3`, putting all files into it and using the command

```
zip -r username_hwk3.zip username_hwk3
```

to create the zip file. Make sure that you create a Makefile if you use multiple files, and include that Makefile in the directory. If you use a Makefile, make sure that it creates an executable named `encode` when it is built. If it is a single file program, name the source file `encode.X`, where `X` is the GNU extension for the language (.c for C, .cpp or .C for C++, etc.)

The program must be well-documented and must conform to my programming guidelines for full credit. It must be placed in the directory no later than midnight of the due date.