Part 3: 80 points

Create a new C source file called “expand.c” and add a new function expand() in this file.

The prototype of expand() must be:

```
int expand (char *orig, char *new, int newsize);
```

The parameter “orig” is the input. The parameter “new” should be a pointer to a fixed size array similar to buffer in main(). The parameter newsize is the number of characters in the new array. The parameter newsize is very important. This is an “input parameter”. It tells your expand() function the maximum numbers of characters that may safely be put into the parameter new. Your code must make sure that expand() does not overflow the new array by using the newsize parameter. The return int should be a value that means “expand was successful” or “expand had an error and processline should stop processing this line”. “expand()” should not change the original line. (Temporary changes are allowed as long as the array remains unchanged at the point of return from expand().)

This function processes the original line as explained below and produces a new line in the array pointed to by the “new” parameter.

–“builtin.h” that should have the prototype of expand in the file. Then both ush.c and expand.c should include the .h file so they both get the same prototype. Update your Makefile so it correctly compiles ush.c and expand.c. You should also have a depends line to show that ush.c and expand.c both depend on builtin.h because they both include it. Your expand.c file may have helper functions in it, but only the expand() function should have its prototype in builtin.h.

In the function processline(), you will add a call to “expand()”. This call to expand must be called before arg parse() because the new line is what must be passed to arg parse. Notice, as a result of this, the original line passed to processline() is not modified. Also, you should just declare an array in processline() for the new line. Do not malloc() space for this new line.

– expand() is to process the original line exactly once, starting with the first character (orig[0]) and looping to the last character. It should do this loop exactly once and do all the required processing during this one loop. When expand finds text that needs replacing, it replaces the text with the required new text and copies the new text to the new string. It then must continue processing the original text at the point after the replaced text. The basic processing done by expand() is to copy each character, one at a time, to the new string unless it detects that the current character is part of text that needs to be replaced. In that case, expand() adds the new text (expansion text) to the new string as specified by the replacement. Be careful to not write code that is order n squared. Also, remember that strlen(3) is an order n operation.

In expand(), replace ${NAME} in the original string with the value of the environment variable of the same name. (Yes, the braces are part of the syntax. $NAME is not processed as an environment variable.) This value of the environment variable is placed in the new string. If the name does not exist in the environment then no characters are copied to the new string. The ${NAME} characters are also NOT copied to the new string. This is done for all ${..}s found in the original string. In ALL cases the ${NAME} will not appear in the new string. If you find a ${ and do not find a closing }, print an error message and stop processing the line. Read about the library function getenv(3).