Part III Project description

Write a C program with reusable functions. The program first prompts user to enter an arithmetic expression, evaluates the expression and displays its result. And then, it prompts user to enter a radix *r*, converts the result to an equivalent radix *r* expansion and displays the expansion. The following shows sample run.

```
Enter an expression: 9*2 - 5/3 -9
Value = -5
Enter radix: 2
Answer = -101
```

For this project, you can assume that an arithmetic expression will be evaluated from left to right with no concepts of precedence. For example, to evaluate 6 + 4 ∗ 3, first evaluate 6 + 4, and then multiply the result 10 with 3. The result should be 30 instead of 18 if the usual rules of precedence were considered. 6 + 4 ∗ 3/7 – 9 will result in -5. The numbers and results are all integers. Here are more examples:

Expression	**Value**
6	6
9 + 5 * 0 - 7	-7
0+ 7 + 4 * 5 - 9	46
9+9-7+4-2/2+4-6	4
7*8-9*4-5*6-6/3+4	368

In addition, you can assume that

- The program asks for one expression per execution.
- An arithmetic expression consists of only tokens that can be any digits from [0-9], any operators from {+, −, ∗, /} and spaces for tokens.
- An arithmetic expression has no parentheses.
- Operands are all single-digit tokens.
- There can be zero or more spaces between two adjacent tokens.
    - For example, 1+2 can be also entered with spaces inserted.
        - 1 + 2, 1+ 2, 1 +2, or 1 + 2.
- A user-entered expression is terminated by the newline (’\n’) character.
- An arithmetic expression may not have more than 80 symbols.
- Only valid arithmetic expressions will be entered. No error-checking for invalid inputs is required.

After the result is displayed, the program will prompt user to enter a radix (e.g., r) and determine an equivalent radix r expansion. For this part of the project the result must be of type signed int. If you used different type, force it to the required type regardless of possible data loss. A radix entered by user can be 2, 3, 4, . . ., 15, or 16. A negative value for all number systems must have a minus sign (-) as prefix. For example,

- If -138 is to be converted using radix 16, the equivalent hexadecimal expansion should be -8A.
- If 284 is to be converted using radix 13, the equivalent base 13 expansion should be 18B. In base 13, the digits used are 0, 1, 2, . . ., 9, A, B, and C, where A, B, and C represent 10, 11, and 12, respectively.

The following table shows more examples.

Number	**Radix**	**Output**
138	16	8A
-2279	12	-139B

37373	10	37373
741	2	1011100101
0	11	0
284	13	18B

Tips:

Divide and Conquer:

You should design a function that evaluates an expression and returns the result as an integer. You should design another function that takes a result, a decimal integer, and a radix and returns an equivalent expansion. For each of the two functions, you may divide the tasks into smaller tasks by writing more functions.

The divide method:

For any radix r ≥ 2, the digits that can be used are 0, 1, . . ., r − 1. Use the letters A, B, C, D, E, and F to represent 10, 11, 12, 13, 14, and 15, respectively, as in the hexadecimal system. Thus, expansions in radix 11 can use the digits 0, 1, . . ., 9, A; expansions in radix 12 can use 0, 1, . . ., 9, A, B, and so on. Use the divide method for base conversion. All digits obtained can be stored in a char array that can be printed out at the end.

The program reads from keyboard (`stdin`) and writes on screen(`stdout`). After each call of `printf`, include `fflush(stdout);` to flush out the contents of the output stream as the output data may be buffered before being written to an output stream like `stdout` .

```
printf("Value = %d\n", result); fflush(stdout);
printf("Enter radix: "); fflush(stdout);
```