Homework 4

Keizou Wang

March 18, 2025

1. Translate the following expression into (a) postfix and (b) prefix notation:

```
(b + sqrt(b*b - 4*a*c))/(2*a)
```

Prefix: `/ + b sqrt - *bb **4ac *2a`
Postfix: `b bb* 4a*c* - sqrt + 2a* /`

2. Some languages (e.g., Algol 68) do not employ short-circuit evaluation for Boolean expressions. However, in such languages an if. . . then. . . else construct (which only evaluates the arm that is needed) can be used as an expression that returns a value. Show how to use if. . . then. . . else to achieve the effect of short-circuit evaluation for A and B and for A or B.

```
A and B: if A then B else False
A or B: if A then True else B
```

3. Consider a midtest loop, here written in C, that processes all lines in the input until a blank line is found:

```
for ( ; ; ) {
    line = read_line();
    if (all_blanks(line)) break;
    process_line(line);
}
```

(*a*) Show how you might accomplish the same task in C using a while loop.

```
line = read_line();
while (!all_blanks(line)) {
    process_line(line);
    line = read_line();
}
```

(*b*) Show how you might accomplish the same task in C using a do loop.

```
do{
    line = read_line();
    if(!all_blanks(line)) process_line(line);
}while (!all_blanks(line))
```

4. Write a *tail-recursive* function in Scheme to compute n factorial ($n! = 1 \times 2 \times \cdots \times n$). You will probably want to define a "helper" function, as discussed in the textbook.

```
(define (factorial n)
    (define (helper a n)
        (if (equal? n 1)
            a
            (helper (* a n) (- n 1))
        )
    )
    (helper 1 n)
)
```

5. Give an example in C in which an in-line subroutine may be significantly faster than a functionally equivalent macro. Give another example in which the macro is likely to be faster. Hint: think about applicative versus normal-order evaluation of arguments.

```
#include <stdio.h>
#define macroSum(x, y) (x + y)

int functionSum(int x, int y){
    return x + y;
}

int factorial(int x){
    return x == 1 ? x : x * factorial(x - 1);
}

int main(){
    // Function faster than macro
    printf("%i\n", functionSum(factorial(10), factorial(10)));
    printf("%i", macroSum(factorial(10), factorial(10)));

    // Macro faster than function
    printf("%i\n", functionSum(10, 10));
    printf("%i", macroSum(10, 10));
}
```