Miguel Muniz
CS 326
Hw 4
3/9/2023

1)
Postfix: b b b * 4 a * c * - sqrt + 2 a * /

Prefix: / + b sqrt - * b b * * 4 a c * 2 a

2)
For A and b:

```
if (A){
    if (B){ // b can't be evaluated without a
        //do something
    }
}
```

For A or B:

```
    if (A){
        //do something
    } else { //evaluations are seperate
        if (B){
        //do something
        }
    }
```

3)

For while loop:

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

}
```

For do while loop:

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

4)

```
(define (factorial n) ;main function
```

```
  (factorial-helper n))

(define (factorial-helper n) ;helper function
  (if (> n 1)
      (* n (factorial (- n 1))) ;recursive call
      1)) ;return 1(else)
```

5)

An example where an inline-subroutine is faster then a MACRO would be for finding the sum of an equation.

```
#define SUM(n) ((n) * ((n) + 1) / 2) //macro

inline int sum(int n) {  //inline subroutine
  return n * (n + 1) / 2;
}
```

the inline subroutine is faster because it is called to acutual code where as the macro must be expanded before it is called.

An example where a MACRO is faster then inline would be for calculating pi.

```
#define PI 3.14159
#define AREA(r) (PI * r * r)
```

This is thanks to the Constant of pi being run at compile time rather then run time.