Lena Kemmelmeier

CS 326 - Programming Languages, Concepts and Implementation

Homework 4
Submitted: March 11th, 2024

CS326 – Programming Languages, Concepts and Implementation

Homework 4

(Due March 19)

1. (24 pts) Translate the following expression into (a) postfix and (b) prefix notation:

```
(b + sqrt(b×b - 4×a×c))/(2×a)
```

   a) **Postfix:** $bbb \times 4\,a \times c \times - sqrt + 2\,a \times /$

   b) **Prefix:** $/ + b\,sqrt - \times b\,b \times \times 4\,a\,c \times 2\,a$

2. (26 pts) 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 are conditions**

   A and B: if A then B else false
   - **We only look at B is A is true, otherwise we just return false**
   - **And requires both A and B to be true, if A isn't true, no point in continuing**

   A or B: if A then true else B
   - **If A is true, no need to look at B**
   - **Or requires only one condition to be true, if A is true, we can stop**
   - **If A is not true, we have to continue and check B**

3. (24 pts) 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);
}
```

Show how you might accomplish the same task in C using a (a) `while` and (b) `do` loop, if `break` instructions were not available.

(a)

```
bool continue = true;

while (continue){

        string line = read_line();

        if (all_blanks(line)){

                continue = false;

        }

        else{

                process_line(line);

        }

}
```

(b)

```
bool continue = true;

do {

        string line = read_line();
```

```
        if (all_blanks(line)){
                continue = false;
        }
        else {
                process_line(line);
        }
} while (continue);
```

Something to consider here is whether all_blanks has side effects (we don't know)

4. (26 pts) Write a *tail-recursive* function in Scheme to compute *n* factorial (n! = 1×2×…×*n*). You will probably want to define a "helper" function, as discussed in the textbook.

```
(define (fact n) ; n is the number we want to calculate the factorial of
  (letrec ((helper (lambda (n a)  ; a is the value we have accumulated (start at 1…)
                     (if (= n 0) ; if n is zero, then we are done!
                       s
                       (helper (- n 1) (* a n))))))  ; recursive call, use n - 1…
    (helper n 1)))
```

5. (Extra Credit - 10 pts) 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.

- **In-line subroutine faster:**

inline int cube(int a){ return a*a*a;}

- **Macro faster:**

#define EQUAL(x, y) ((x == y ? (1) : (0))