**Amarilda Celhaka**

**Assignment #6**

**CS 4250**

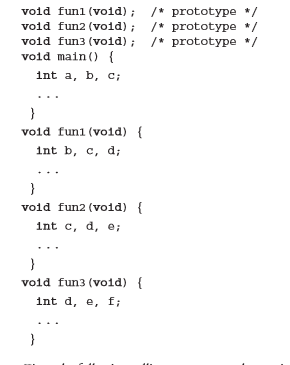
**4/25/2019**

1. [4 points] Describe the primary tasks of a lexical analyzer.

* A lexical analyzer serves as the front end of a syntax analyzer. A lexical analyzer performs syntax analysis at the lowest level of program structure.
* The lexical analyzer collects characters into logical groupings and assigns internal codes to the groupings according to their structure.
* A lexical analyzer extract lexemes from an input of strings and produces the corresponding tokens. Its processes include skipping comments and white spaces outside the lexemes as they are not relevant to the meaning of the program.
* It inserts lexemes for user-defined names into the symbol table.
* Detects syntactic errors in tokens and reports such errors to the user.

1. [4 points] Some programming languages are typeless. Indicate advantages and disadvantages of having no types in a language.

Some programming languages like SmallTalk are typeless. They have no data types specified, all the code is effectively generic. The main advantage is that it provides flexibility, does not require to know type rules but static type checking is sacrificed. It requires to find some errors by run time debugging that in a typed language are found from a compiler.

1. [6 point] Problem #11 (b), p.231 – 232.

a. main calls fun1; fun1 calls fun2; fun2 calls fun3.

d, e, f in fun3,

c in fun2

b in fun1

a in main

b. main calls fun1; fun1 calls fun3

d, e, f in fun3,

b, c, in fun1,

a in main

c. main calls fun2; fun2 calls fun3; fun3 calls fun1.

b, c, d in fun1,

e, f in fun3

a in main

d. main calls fun3; fun3 calls fun1.

b, c, d in fun1

e, f in fun3

a in main

e. main calls fun1; fun1 calls fun3; fun3 calls fun2.

C, d, e in fun2

f in fun3

b in fun1

a in main

f. main calls fun3; fun3 calls fun2; fun2 calls fun1.

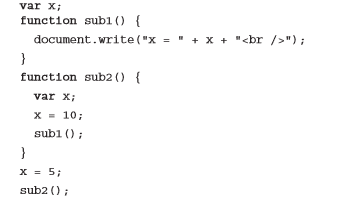
b, c, d in fun1

e in fun2

f in fun3

a in main

1. [6 points] Problem #7, p.229.



Static-scoping: value under sub1: X = 5

Dynamic-scoping: value under sub1: X = 10

1. [5 points] Describe a situation when the ‘add’ operator in a programming language would not be commutative.

Add operator (+) is not commutative in the situation when we concatenate two strings. Example: “FirstName” + “ LastName” = “FirstName LastName” which is clearly not commutative.

1. [5 points] Describe a situation when the ‘add’ operator in a programming language would not be associative.

If we are working with a 16 bit integer it can store numbers from the range −32,768 to 32,767. If we try to add:

(32,767 + 1) – 1 = ERROR. An overflow exception will be caused, therefore the add operator is not associative.

32,767 + (1 – 1) = 32,767

1. [5 points] Explain why it is difficult to eliminate functional side effects in C.

Functional side effects include functions changing the value of the parameter or it changes the global variable.

Example: x= 5; //global variable

Y = x + ex(x); if we assume that function ex(x)

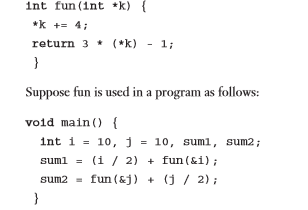
changes the value of x to 10 and returns 2

If associativity is left to right: Y= 7

Right to left Y = 12;

Here we can see that it is difficult to eliminate the side effects of functions because they offer some kind of flexibility to the programmer.

1. [5 points] Problem # 13, p.327.



What are the values of sum1 and sum2

a. if the operands in the expressions are evaluated left to right?

Sum1 = 5 + (3 \* 14 – 1) = 5 + 41 = 46

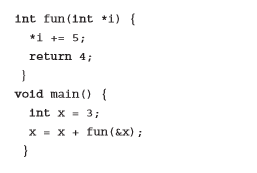
Sum2 = (3 \* 14 – 1 ) + (14 / 2) = 41 + 7 = 48

b. if the operands in the expressions are evaluated right to left?

Sum1 = 14 / 2 + 41 = 48

Sum2 = 41 + 5 = 46

1. [5 points] Problem # 19, pp. 327 – 328.



What is the value of x after the assignment statement in main, assuming

a. operands are evaluated left to right.

X = 3 + 4 = 7

b. operands are evaluated right to left.

X = 8 + 4 = 12