Andrew Vo
Professor Chinua Umoja
Test 2
November 16, 2020

Coded Answers

Question 1)

```
import os
import sys
import re
# Parses text file into an empty list r
f = open("question1.txt", "r")
r = f.read()
# converts the list of lists into a lists of strings as list_String
list_String = []
list_String = r.split( )

# iterates through the list of strings and assigns a token to every valid string
for i in list_String:
   # Uses regex to find any string that starts with @, $, or %
   if(re.fullmatch("^(@|\$|%)[a-zA-Z0-9]+_?[a-zA-Z0-9]+", i)):
       print(i + ": " + "Pearl")
   # Uses regex to find any string that is within quotations
   elif(re.fullmatch("^(\").*(\"$)", i)):
      print(i + ": " + "String")
   # Uses regex to find just numbers with a minimum of 1 digit char
   elif(re.fullmatch("[1-9][0-9]*", i)):
       print(i + ": " + "Decimal")
   # Uses regex to find numbers starting with 0 and digit chars consisting of 1-7
   elif(re.fullmatch("0[0-7]*", i)):
       print(i + ": " + "Octal")
   # Uses regex to find a number starting with 0x and ending with an optional suffix
   elif(re.fullmatch("0[xX][0-9a-fA-F]*", i)):
       print(i + ": " + "C-Style integer literal hex")
   # Finds any floating point with optional prefix, minimum of 1 digit and followed by
a exponent
   elif(re.fullmatch("[-+]?([0-9]*[.])?[0-9]+([eE][-+]?\d+)?", i)):
       print(i + ": " + "C-Style float literal")
   # Finds any string starting with a alpha, followed by optional underscore.
   elif(re.fullmatch("[a-zA-Z_][a-zA-Z_0-9]*", i)):
       print(i + ": " + "C-Style character literal")
   # Finds any non-alphanumeric char that was listed in the question
```

```
elif(re.fullmatch("([+]|[=]|[\==]|[-]|[\]|[\\]|[\/]|[*]|[++]|[--]|[%]|[&&]|[||]|[!=]|[
(]|[)]|[{]|[}])*", i)):
        print(i + ": " + "Non-Alphanumeric")
    else:
        print(i + ": " + "Invalid")
```

Question 2)

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

#define size 10

/* FUNCTION DECLARATIONS */
void heap(void);
void stack(void);
void static_s(void);
/* FUNCTIONS */
void heap(void){ int*z = (int*) malloc(100000*sizeof(int)); }
void stack(void){ int y[size]; }
void static_s(void) { static int x[size]; }

/* DRIVER METHOD */
int main() {
   struct timespec t_1, t_2;
   int i;

   /* HEAP */
   clock_gettime(CLOCK_REALTIME, &t_1);
   for( i=0; i<100000; i++){
       heap();
   }
   clock_gettime(CLOCK_REALTIME, &t_2);
   printf("Heap: %ld \n", (t_2.tv_nsec - t_1.tv_nsec));

   /* STACK */
   clock_gettime(CLOCK_REALTIME, &t_1);
```

```
    for( i=0; i<100000; i++ ){
        stack();
    }
    clock_gettime(CLOCK_REALTIME, &t_2);
    printf("Stack: %ld \n", (t_2.tv_nsec - t_1.tv_nsec));

    /* STATIC */
    clock_gettime(CLOCK_REALTIME, &t_1);
    for ( i=0; i< 100000; i++ ) {
        static_s();
    }
    clock_gettime(CLOCK_REALTIME, &t_2);
    printf("Static: %ld \n", (t_2.tv_nsec - t_1.tv_nsec));

    return 0;
}
```

```
(base) andrews-mbp-2:PLCExam andrew$ ./Test2q2
Heap: 11924000
Stack: 225000
Static: 177000
(base) andrews-mbp-2:PLCExam andrew$
```

Question 6)

```
from re import fullmatch
import sys
import os
import re

f = open("question6.txt", "r")
r = f.read()
# converts the list of lists into a lists of strings as list_String
```

```
list_String = []
list_String = r.split( )

java_While = False
java_If = False
# Java while statement

nextToken = None
IF_code = "if"
WHILE_code = "while"
LEFT_PAREN = "("
RIGHT_PAREN = ")"
ELSE_CODE = "else"


for i in list_String:

if(re.fullmatch("([\w]|[+]|[=]|[\==]|[-]|[\]|[\\]|[\/]|[*]|[++]|[--]|[%]|[&&]|[||]|[!=
]|[(]|[)]|[{]|[}])*", i)):
       print(i + ": " + "Non-Alphanumeric")
   else:
       nextToken+=1
# RDA for Mathematical Assignment Statement
def math_expr():
# checks nextToken for a variable
   if((re.fullmatch("[a-zA-Z0-9]", nextToken))):
       return var()
   else:
       return error()
   # uses Regex to see if the nextToken is a assignment operator

if((re.fullmatch("([\w]|[+]|[=]|[\==]|[-]|[\]|[\\]|[\/]|[*]|[++]|[--]|[%]|[&&]|[||]|[!
=]|[(]|[)]|[{]|[}])*", nextToken))):
       return error()
   else:
       return
   if((re.fullmatch("[a-zA-Z0-9]", nextToken))):
       return error()
   else:
       return var()

# RDA for Mathematical Expression Statement
```

```python
def math_assign():
   # checks to see if the variable is a variable
   if(re.fullmatch("[a-zA-Z0-9]", nextToken)):
       return var()
   else:
       return error()
   # checks the nextToken if it is an assignment operator
   if(re.fullmatch("[==]", nextToken)):
       return expr()
   else:
       return error()
   # checks next token to see if it is a variable
   if(re.fullmatch("[a-zA-Z0-9]", nextToken)):
       return lex()
       return statement()
   else:
       return error()


# RDA for Java While statement
def while_stmt():
   # checks to see if first token is 'while'
   if (nextToken != WHILE_code):
       return error()
   else:
       return lex()
   # checks for first left parenthesis
   if (nextToken != LEFT_PAREN):
       return error()
   else:
   # parses the Boolean expression
       boolexpr()
   if (nextToken != RIGHT_PAREN):
       return error()
   else:
       return statement()

# RDA  for Java If statement
def if_stmt():
   # checks to see if the first token is 'if'
   if (nextToken != IF_code):
       return error()
```

```
    else:
        return lex()
    # check for the first left parenthesis
    if (nextToken != LEFT_PAREN):
        return error()
    else:
    # parses the Boolean expression
        boolexpr()
    # checks for the first right parenthesis
    if (nextToken != RIGHT_PAREN):
        return error()
    else:
        return statement()
    # if an else is net, parse the else clause
    if (nextToken == ELSE_CODE):
        return lex()
        return statement()
```

Question 8)

```
$x = 10;
sub dynamic_Scoping
{
  return $x;
}
sub return_D
{
  local $x = 20;
  return dynamic_Scoping();
}
print "Dynamic Scoping: ", return_D()."\n";
$y = 15;
sub static_Scoping
{
   return $y;
}
sub return_S
{
   my $y = 25;
   return static_Scoping();
```

```
}
print "Static Scoping: ", return_S()."\n";
```