Keeyon Ebrahimi
kme322
N14193968
Each problem on seperate page

1 Problem 1

Provide regular expressions for defining the syntax of the following

(*a*)
```
([a-z0-9]*[A-Z]+[a-zA-Z]*[0-9]+[a-zA-Z0-9]*)|
([a-zA-Z]*[0-9]+[a-z0-9]*[A-Z]+[a-zA-Z0-9]*)
```

(*b*)
```
 [0-9]+\.[0-9]+E[0-9]+
```

(*c*)
```
^[a-zA-Z][a-zA-Z_0-9]{0,19}
```

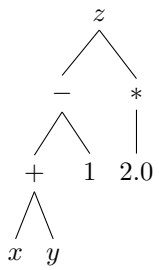
2 Problem 2

(a)

- <Agenda> ::= <SubRouteDef>
  { <varDecl> ; \n}
  begin
  { <varOperate>; \n}
  end <ID>;

- <SubRouteDef> ::= <func> | <prog> | <procedure>
- <func> ::= function <ID> '(' [<varDecl>+, <varDecl>] ')'
- <prog> ::= program <ID>;
- <procedure> ::= procedure <ID> '(' [<varDecl>+, <varDecl>] ')'
- <varDecl> ::= var <ID>
- <varOperate> ::= <varAssign> | <varCalc>
- <varAssign> ::= <ID> := <varOption> ;
- <varCalc> ::= <ID> := <ID> | (<varOption> <MathOperation>)*<varOption>;
- <varOption> ::= <ID> | <number>

(b)



3 Problem 3

(*a*) Static scoping means we bind the value of a variable based on the innermost enclosing block. Dynamic scoping means we bind the value of a variable based on the most recent declaration of the variable.

(*b*)
```
foo():
    x = 12;
Main():
    x = 2;
    foo();
    print x;
```

Static scoping will print 2
Dynamic Scoping will print 12

(*c*) In a block structured, statically scoped language, we resolve variables by the innermost enclosing block.

(*d*) In a block structured but dynamically scoped language, we resolve variables by the most recent declaration of the variable.

4 Problem 4

Please ignore the spaces between the different frames. I put the labels there so you know which function each group of boxes is representing. The spaces between the boxes between frames is not intended.

A | Return Address |

F | Dynamic Link to A |
| Static Link to A |
| Return Address |

B | Dynamic Link to F |
| Static Link to A |
| Parameter $ProcessC$ | $\rightarrow$ This contains G's information |
| Return Address |

G | Dynamic Link to B |
| Static Link to F |
| Parameter $ProcessH$ | $\rightarrow$ This contains D's information |
| Return Address |

D | Dynamic Link to G |
| Static Link to B |
| Parameter 6 |
| Return Address |

Part B

Sometimes functions will access variables that are outside of its immediate scope. If we do not store the closer on the stack, when the closer reaches outside of its immediate scope, its frame will get popped, causing memory issues and failed execution.

```
foo(x):
    y = 12;
    bar(z):
        print x + y + z;
Main():
    var baz = foo(4);
    baz(10);
    print x;
```

5 Problem 5

(a) Pass by value $\rightarrow$ 1 2 3 4 5

(b) Pass by reference $\rightarrow$ 4 2 3 4 5

(c) Pass by value-reference $\rightarrow$ 2 2 3 4 5

(d) Pass by name $\rightarrow$ 1 2 4 4 5

6 Problem 6

Part A

```
with Ada.Text_IO;

procedure Problem6 is
use Ada.Text_IO;

    protected PrintOne is
        entry TwoWasPrinted;
    end PrintOne;

    protected PrintTwo is
        entry OneWasPrinted;
    end PrintTwo;

    p1 : PrintOne;
    p2 : PrintTwo;

    protected body PrintOne is
    S : String;
    begin
        loop
            Put_Line('one');
            p2.OneWasPrinted;
            accept TwoWasPrinted do
            end TwoWasPrinted;
        end loop;
    end PrintOne;
```

```
protected body PrintTwo is
    S : String;
    begin
        loop
           accept OneWasPrinted do
            end OneWasPrinted;
            Put_Line('two');
            p1.TwoWasPrinted;
        end loop;
    end PrintOne;

begin
    PrintOne;
    PrintTwo;
end Problem6;
```

Part B

These two are running concurrently because we have two threads that are both making progress. They both do not need to be running at the same instant like with parallelism. Because we have multiple tasks making progress, we have concurrency.