CSCI-GA.2110.001 Programming Languages October 11, 2015

Solutions to Problem 1 of Homework 1 ()

Name: GOWTHAM GOLI (N17656180) *Due: Sunday, October 11*

Provide regular expressions for defining the syntax of the following.

(a) Passwords consisting of letters and digits that contain at least two upper case letters and one digit. They can be of any length (obviously at least three characters).

Solution:

$$\begin{aligned}
&Letter &&\rightarrow UC|LC\\
&UC &&\rightarrow [A-Z]\\
&LC &&\rightarrow [a-z]\\
&Digit &&\rightarrow [0-9]\\
&Password &&\rightarrow (Letter \cup Digit)^{*}Digit.UC(Letter \cup Digit)^{*}UC(Letter \cup Digit)^{*}\\
& &&|(Letter \cup Digit)^{*}UC.Digit(Letter \cup Digit)^{*}UC(Letter \cup Digit)^{*}\\
& &&|(Letter \cup Digit)^{*}UC(Letter \cup Digit)^{*}UC.Digit(Letter \cup Digit)^{*}
\end{aligned}$$

□

(b) Floating point literals that specify an exponent, such as the following: $2.43876E13$ (representing 243.867×10^{13}).

Solution: Assuming that the (E) part can be optional, The left side of the exponent (E) part can take the following forms

- Left side of the decimal point is 0, Right side of the decimal point allows one or more digits from 0-9. Therefore, regular expression is $0.[0-9]+$
- Left side of decimal point is empty, Right side of the decimal point allows one or more digits from 0-9 . Therefore, regular expression is $.[0-9]+$
- There is no decimal point. Regular expression is $[1-9][0-9]^{*}$
- The left side of the decimal starts with 1-9 followed by zero or more digits from 0-9. The right side of the decimal allows one or more digits from 0-9. Therefore, regular expression is $[1-9][0-9]^{*}.[0-9]+$

(Note that ? makes the preceding token optional and + makes the token appear at least once) Therefore the final regular expression is

$$((0?.[0-9]+)|([1-9][0-9]^{*})|([1-9][0-9]^{*}.[0-9]+))\ (E[0-9]+)?$$

□

(c) Procedure names that: must start with a letter; may contain letters, digits, and (underscore); and must be no more than 10 characters.

Solution:

$$
\begin{aligned}
&Letter &&\rightarrow [A-Z] | | [a-z] \\
&Digit &&\rightarrow [0-9] \\
&Procedure &&\rightarrow Letter.\underbrace{(\epsilon \cup Letter \cup \_ \cup Digit)\ldots(\epsilon \cup Letter \cup \_ \cup Digit)}_{\text{9 times}}
\end{aligned}
$$

□

CSCI-GA.2110.001 Programming Languages October 11, 2015

Solutions to Problem 2 of Homework 1 ()

Name: GOWTHAM GOLI (N17656180) *Due: Sunday, October 11*

(a) Provide a simple context-free grammar for the language in which the following program is written. You can assume that the syntax of names and numbers are already defined using regular expressions (i.e. you dont have to define the syntax for names and numbers)

```
program one;
  var x;

  function f(var x, var y)
  var z;
  begin
  z := x+y-1;
  return z*2;
  end f;

  procedure g()
  var a;
  begin
  a := 3;
  x := a;
  end g;

begin
g();
print(f(x));
end one;
```

Solution:

$$
\begin{array}{ll}
<Program> & \rightarrow program <prog\_name>;\ <prog\_decl> \\
 & \quad begin <body> end <prog\_name>; \\
<prog\_decl> & \rightarrow\ <var\_decl>\ <func\&proc\_decl> \\
<var\_decl> & \rightarrow \epsilon \mid var <identifier>;\ <var\_decl> \\
<func\&proc\_decl> & \rightarrow\ \epsilon\ \mid <func\_decl>\ <func\&proc\_decl> \\
 & \quad \mid\ <proc\_decl>\ <func\&proc\_decl> \\
<func\_decl> & \rightarrow\ function <func\_name> (<param>)\ <var\_decl> \\
 & \quad begin <body> end <func\_name>; \\
<proc\_decl> & \rightarrow\ procedure <proc\_name> (<param>)\ <var\_decl> \\
 & \quad begin <body> end <proc\_name>; \\
<body> & \rightarrow\ <stmt\_list> \\
<stmt\_list> & \rightarrow\ \epsilon\ \mid null;\ \mid <stmt>;\ <stmt\_list> \\
<stmt> & \rightarrow\ \epsilon\ \mid <identifier>:=<expr>\ \mid return <expr> \\
 & \quad \mid\ <func\_name> (<args>)\ \mid\ <proc\_name> (<args>) \\
<param> & \rightarrow\ \epsilon\ \mid var <identifier> \mid\ function <func\_name> \mid\ procedure <proc\_name> \\
 & \mid\ var <identifier>, <param>\ \mid\ function <func\_name>, <param> \\
 & \mid\ procedure <proc\_name>, <param> \\
<args> & \rightarrow\ \epsilon\ \mid\ <identifier>\ \mid\ <func\_name> (<args>) \mid <proc\_name> (<args>) \\
 & \mid\ <identifier>, <args>\ \mid <func\_name> (<args>), <args> \\
 & \mid\ <proc\_name> (<args>), <args> \\
<expr> & \rightarrow\ <expr> + <expr>\ \mid\ <expr> - <expr>\ \mid\ <expr> * <expr> \\
 & \quad \mid\ <expr> / <expr>\ \mid (<expr>) \mid number \mid identifier \\
<prog\_name> & \rightarrow identifier \\
<proc\_name> & \rightarrow identifier \\
<func\_name> & \rightarrow identifier
\end{array}
$$

□

(b) Draw the parse tree for the above program

Solution:

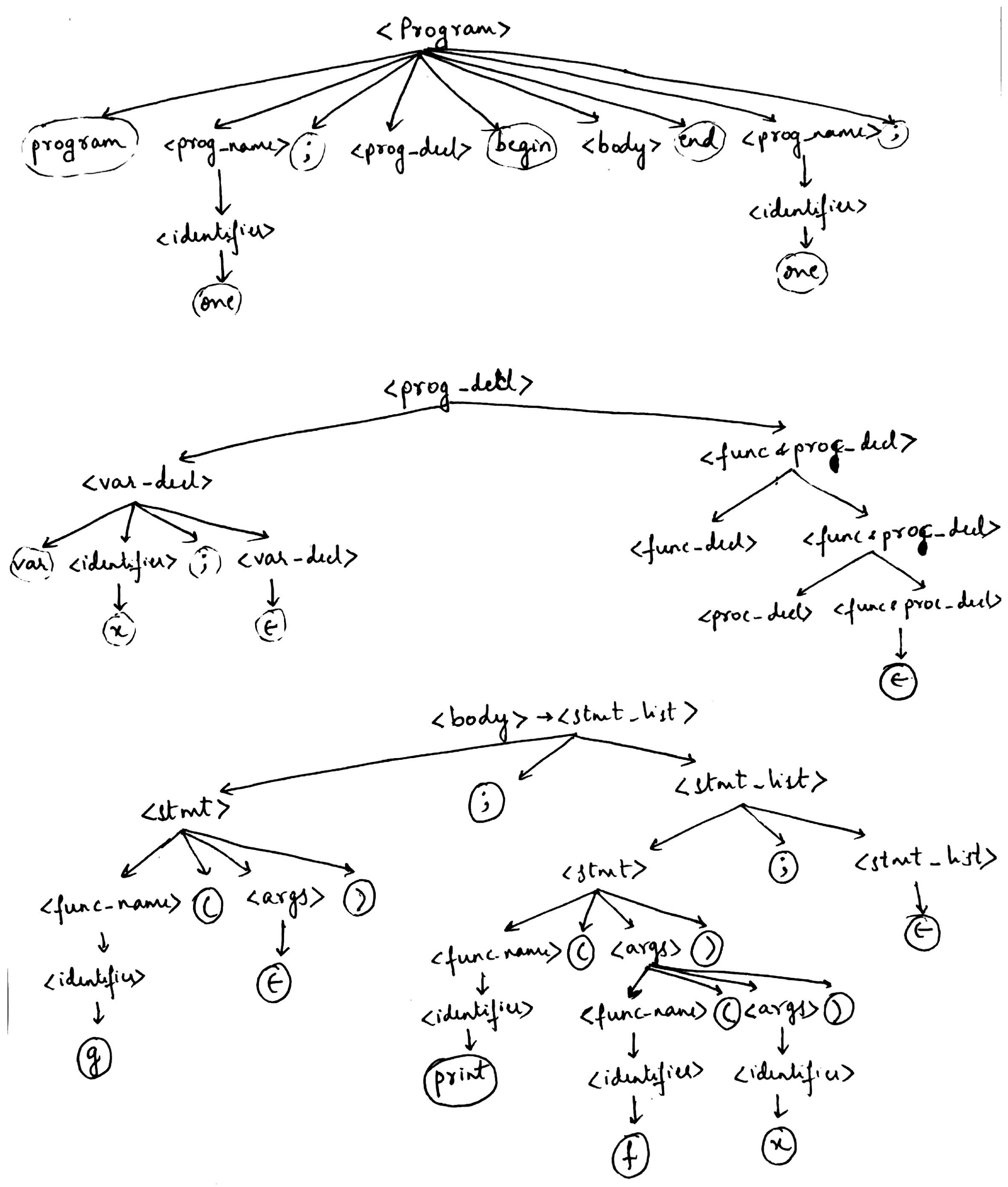

Figure 1: Parse tree of Program *one*

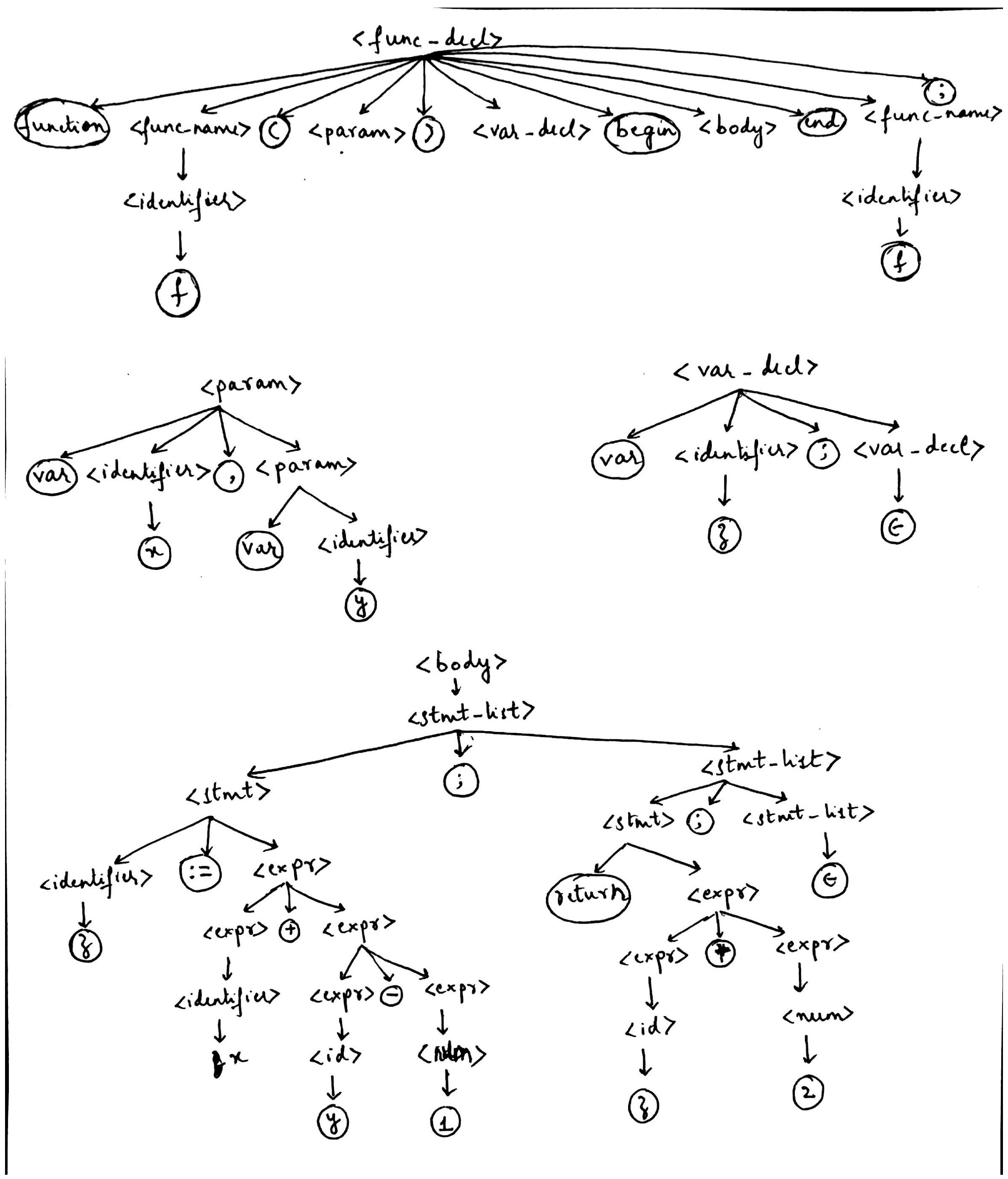

Figure 2: Parse tree of function f

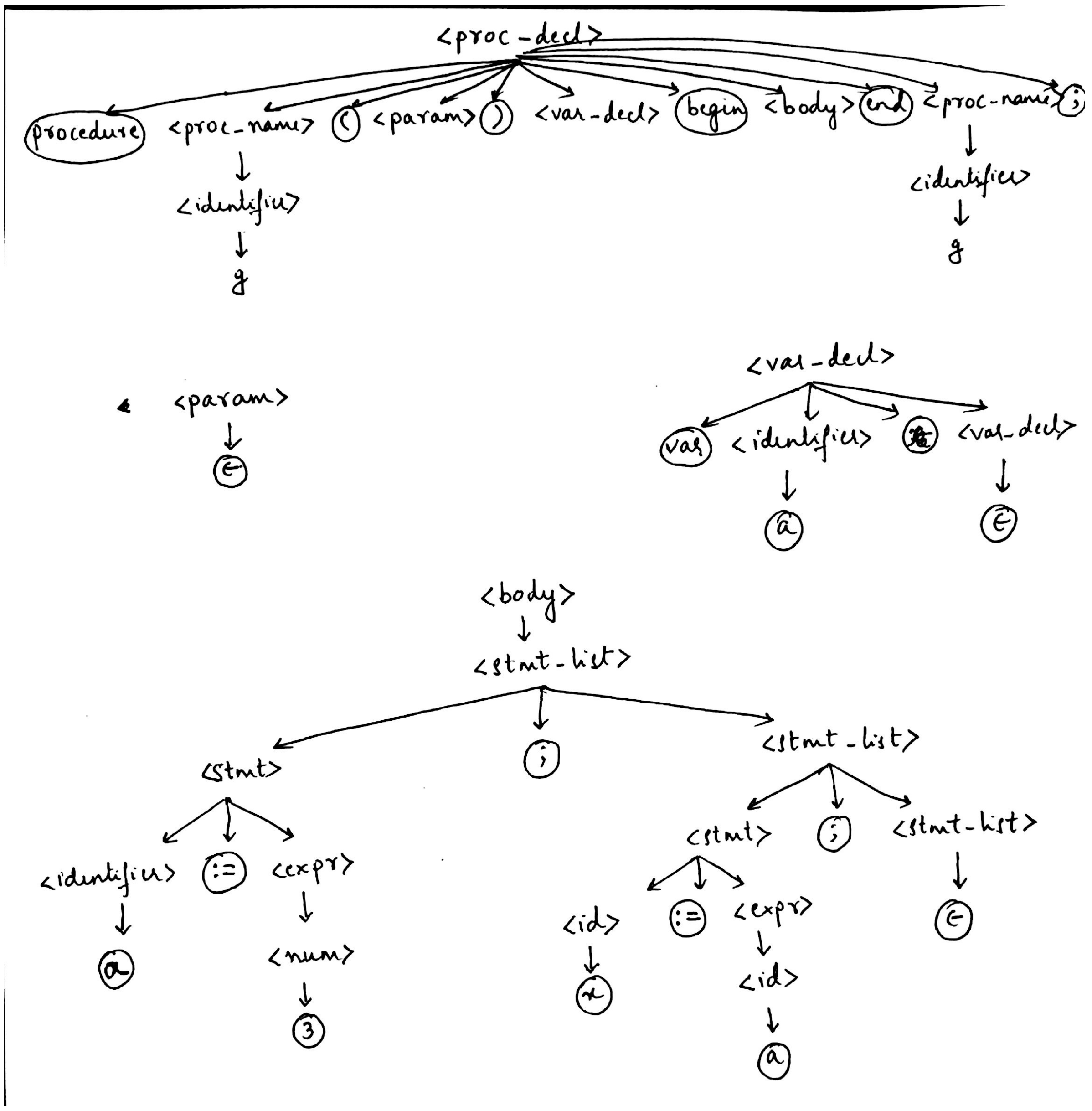

Figure 3: Parse tree of procedure g

□

CSCI-GA.2110.001 Programming Languages October 11, 2015

Solutions to Problem 3 of Homework 1 ()

Name: GOWTHAM GOLI (N17656180) *Due: Sunday, October 11*

(a) Define the terms static scoping and dynamic scoping.

Solution:

Static Scoping

In Static scoping, the structure of the program source code determines what variables you are referring to. Therefore the scope of the bindings can be determined at the compile time

Dynamic Scoping

In Dynamic scoping, the runtime state of the program stack determines what variable you are referring to. Therefore the scope of the bindings can be determined at the run time ☐

(b) Give a simple example, in any language you like (actual or imaginary), that would illustrate the difference between static and dynamic scoping. That is, write a short piece of code whose result would be different depending on whether static or dynamic scoping was used.

Solution:

```
program a() {
  x: integer;
  x = 1;

  procedure b() {
    x = 2;
  }

  procedure c() {
    x: integer;
    b();
  }

  c();
  print x;
}
```

With static scoping, we observe the lexical structure of the program source code to see which x we are referring to. There is no definition for x in the local scope for b. Hence we look for the definition of x in the statically enclosing scope of b where we can find the global definition of x. Therefore this is the x we are referring to in b. Therefore $x = 2$ writes 2 to the global value of x and 2 will be printed in the case of static scoping

With dynamic scoping, we look for the most active binding made at runtime. So when the program starts running, the global reference to x, let it be x_1 is pushed onto the stack and then the local definition of x made in c is pushed onto the stack, let it be x_2. When b is called from c it looks for the most recent binding of x which is x_2. Therefore $x = 2$ writes 2 to x_2. Now when c returns x_2 is popped from the stack and 1 is printed at the end of the program

□

(c) In a block structured, statically scoped language, what is the rule for resolving variable references (i.e. given the use of a variable, how does one find the declaration of that variable)?

Solution: To resolve a reference to any variable, we examine the local scope and statically enclosing scopes until a binding to that variable is found

□

(d) In a block structured but dynamically scoped language, what would the rule for resolving variable references be?

Solution: To resolve a reference to any variable, we use the most recent, active binding made to that variable at run time

□

CSCI-GA.2110.001 Programming Languages October 11, 2015

Solutions to Problem 4 of Homework 1 ()

Name: GOWTHAM GOLI (N17656180) *Due: Sunday, October 11*

(a) Draw the state of the stack, including all relevant values (e.g. variables, return address, dynamic link, static link), during the execution of procedure S in the following program.

```
procedure P;
  procedure Q(procedure R)
    procedure S(x:integer);
    begin
      writeln(x);
    end;
  begin (* Q *)
    R(S);
  end;
  procedure T;
    procedure U(procedure V);
    begin
      V(6);
    end;
  begin (* T *)
    Q(U);
  end;
begin (* P *)
  T;
end;
```

Solution:

In the below figure, the return address of each stack frame contains the address of the instruction where the function has to return after it's execution

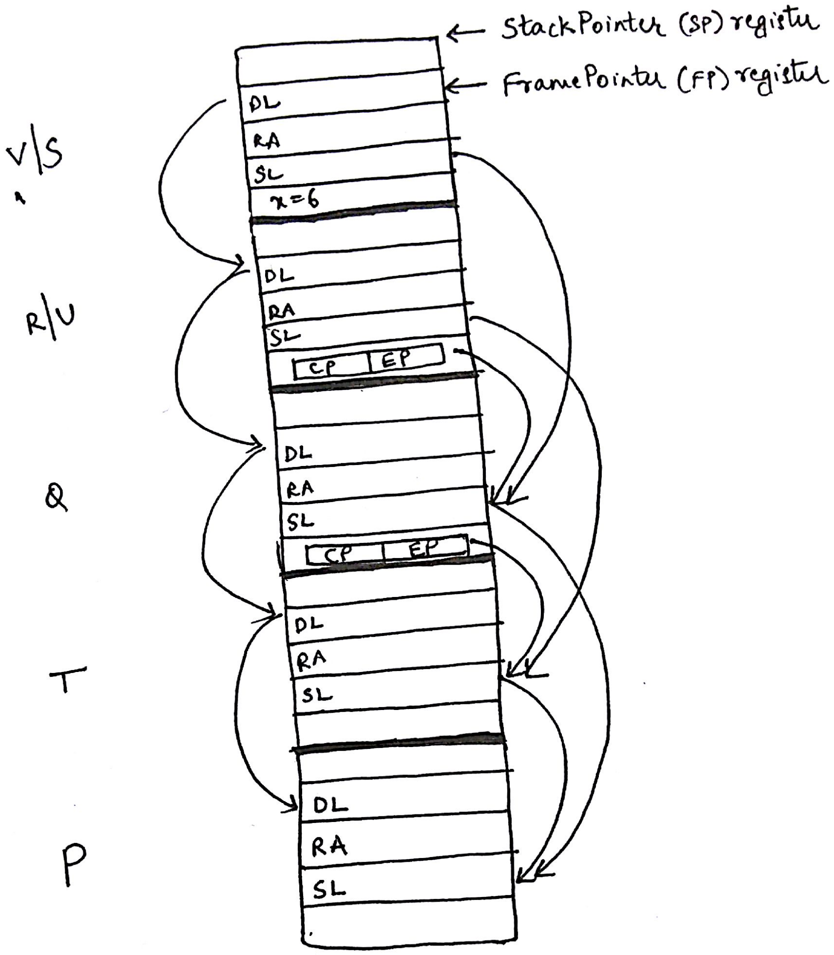

Figure 4: Call Stack g

□

(b) Explain why closures on the heap are needed in some languages, and give an example of a program (in any syntax you like) in which a closure would need to be allocated on the heap.

Solution:

Consider the following code, Q is the sub-procedure of P which uses the local variable, a of P and P returns Q.

Therefore, P has to be on top of the stack before calling Q but what if that's not the case. As seen in the code, T calls P and it returns Q into S. Now P is popped from the stack and now S/Q is being called. What happens to the a that is being used in S/Q as P is not on the stack anymore

In such a case, we would require to have the closure in the heap and it contains a copy of all the local variables of P that are being used in Q

```
procedure P
  a : Integer;
  procedure Q;
  begin
     .....
     a = 5;
  end Q;
begin
  .....
  return Q;
end P;

procedure T
  var S;
begin
  .....
  .....
  S = P();
end T;
```

□

CSCI-GA.2110.001 Programming Languages October 11, 2015

Solutions to Problem 5 of Homework 1 ()

Name: GOWTHAM GOLI (N17656180) *Due: Sunday, October 11*

For each of these parameter passing mechanisms, state what the following program (in some Pascal-like language) would print if that parameter passing mechanism was used:

```
program foo;
  var i,j: integer;
  a: array[1..6] of integer;
  procedure f(x,y:integer)
    begin
      x := x * 3;
      i := i + 1;
      y := a[i] + 2;
    end
begin
  for j := 1 to 6 do a[j] = j;
  i := 1;
  f(i,a[i]);
  for j := 1 to 6 do print(a[j]);
end.
```

(a) pass by value

Solution:

1 2 3 4 5 6 □

(b) pass by reference

Solution:

6 2 3 4 5 6 □

(c) pass by value-result

Solution:

4 2 3 4 5 6 □

(d) pass by name

Solution:

1 2 3 6 5 6 □

CSCI-GA.2110.001 Programming Languages October 11, 2015

Solutions to Problem 6 of Homework 1 ()

Name: GOWTHAM GOLI (N17656180) *Due: Sunday, October 11*

(a) In Ada, define a procedure containing two tasks, each of which contains a single loop. The loop in the first task prints the numbers from 1 to 500, the loop in the second task prints the numbers from 501 to 1000. The execution of the procedure should cause the tasks to alternate printing fifty numbers at a time, so that the user would be guaranteed to see: $1, 2 \ldots 50, 501, 502, \ldots, 550, 51, 52, \ldots, 100, 551, 552, \ldots, 600, \ldots$ Be sure there is only one loop in each task.

Solution:

```
procedure print_1to1000 is
  task print_1to500 is
    entry print;
  end print_1to500;
  task print_501to1000 is
    entry print;
  end print_501to1000;

  task body print_1to500 is
  begin
    for i in 1 .. 500 loop
      Put(i);
      if i%50 == 0 then
        print_501to1000.print;
        accept print do
          null;
         end print;
      end if;
    end loop;
    print501_1000.print;
  end print_1to500;

  task body print_501to1000 is
  begin
    accept print do
      null;
    end print;
    for i in 501 .. 1000 loop
      Put(i);
      if i%50 == 0 then
```

```
        print_1to500.print;
        accept print do
          null;
        end print;
      end if;
    end loop;
  end print_501to1000;

begin
  null;
end print_1to1000;
```

□

(b) Looking at the code you wrote for part (a), are the printing of any of the numbers occurring concurrently? Justify your answer by describing what concurrency is and why these events do or do not occur concurrently.

Solution:

Concurrency means that we cannot make any assumptions about the relative ordering of execution of any two or more parts of the code.

But we can clearly see that we are restricting the above code in such a way that the tasks print 1 . . . 50 51 . . . 100 . . . 451 . . . 500 951 . . . 1000 by synchronizing the two tasks using the entry calls. Hence there is relative ordering of the execution of the two tasks. Therefore the printing of the numbers is not occurring concurrently □