Problem 10.1

1. Show the stack with all activation record instances, including static and dynamic chains, when execution reaches position 1 in the following skeletal program. Assume `Bigsub` is at level 1.

```
procedure Bigsub is
  procedure A is
   procedure B is
     begin  -- of B
     . . . <------------------ 1
     end; -- of B
   procedure C is
     begin  -- of C
     . . .
     B;
     . . .
     end; -- of C
   begin  -- of A
   . . .
   C;
   . . .
   end; -- of A
  begin  -- of Bigsub
  . . .
  A;
  . . .
  end;  -- of Bigsub
```

Problem 10.1 (cont'd)

Subprogram	Static Parent
Bigsub	Main
A	Bigsub
B	A
C	A

Call Sequence:

Main calls Bigsub (assumed)
Bigsub calls A
A calls C
C calls B

NOTE: The example code doesn't show any variable declarations. We will show where the locals would go, if there are any, by showing a single local storage of one word for each subprogram. This is not necessary.

ARI	Address	Value	Content	Description
B	16		Local	
	15	9	Dynamic Link	Link to caller's ARI: C
	14	5	Static Link	Link to parent's ARI: A
	13		Return (to C)	
C	12		Local	
	11	5	Dynamic Link	Link to caller's ARI: A
	10	5	Static Link	Link to parent's ARI: A
	9		Return (to A)	
A	8		Local	
	7	1	Dynamic Link	Link to caller's ARI: Bigsub
	6	1	Static Link	Link to parent's ARI: Bigsub
	5		Return (to Bigsub)	
Bigsub	4		locals	
	3	0	Dynamic Link	Link to caller's ARI: Main
	2	0	Static Link	Link to parent's ARI: Main
	1		Return (to Main)	
Main	0		Locals	(place holder)

Problem 10.2

2. Show the stack with all activation record instances, including static and dynamic chains, when execution reaches position 1 in the following skeletal program. Assume `Bigsub` is at level 1.

```
procedure Bigsub is
 MySum : Float;
 procedure A is
   X : Integer;
 procedure B(Sum : Float) is
   Y, Z : Float;
   begin -- of B
   . . .
   C(Z)
   . . .
   end; -- of B
 begin -- of A
 . . .
 B(X);
 . . .
 end; -- of A
procedure C(Plums : Float) is
 begin -- of C
 . . .<------------------ 1
 end; -- of C
L : Float;
begin -- of Bigsub
. . .
A;
. . .
end;  -- of Bigsub
```

Problem 10.2 (cont'd)

Subprogram	Params	Locals	Static Parent
Bigsub	--	MySum, L	
A	--	X	Bigsub
B	Sum	Y, Z	A
C	Plums	--	Bigsub

Call Sequence:

Bigsub calls A
A calls B(X)
B calls C(Z)

NOTE: The example code doesn't show any variable declarations. We will show where the locals would go, if there are any, by showing a single local storage of one word for each subprogram.

ARI	Address	Value	Content	Description
C	19		Param	Plums
	18	10	Dynamic Link	Link to caller's ARI: B
	17	1	Static Link	Link to parent's ARI: Bigsub
	16		Return (to C)	
B	15		Local	Z
	14		Local	Y
	13		Param	Sum
	12	6	Dynamic Link	Link to caller's ARI: A
	11	6	Static Link	Link to parent's ARI: A
	10		Return (to A)	
A	9		Local	X
	8	1	Dynamic Link	Link to caller's ARI: Bigsub
	7	1	Static Link	Link to parent's ARI: Bigsub
	6		Return (to Bigsub)	
Bigsub	5		Local	L
	4		Local	MySum
	3	0	Dynamic Link	Link to caller's ARI: Main
	2	0	Static Link	Link to parent's ARI: Main
	1		Return (to Main)	
Main	0		Local	(place holder)

Problem 10.3

3. Show the stack with all activation record instances, including static and dynamic chains, when execution reaches position 1 in the following skeletal program. Assume `Bigsub` is at level 1.

```
procedure Bigsub is
 procedure A(Flag : Boolean) is
  procedure B is
   ...
   A(false);
   end; -- of B
  begin -- of A
  if flag
   then B;
   else C;
  ...
  end; -- of A
 procedure C is
  procedure D is
   ... <------------------ 1
   end; -- of D
  ...
  D;
  end; -- of C
 begin -- of Bigsub
 ...
 A(true);
 ...
  end;  -- of Bigsub
```

2015/02/11 128.198.202.237

The calling sequence for this program for execution to reach `D` is

`Bigsub` calls `A`
`A` calls `B`
`B` calls `A`
`A` calls `C`
`C` calls `D`

Problem 10.3 (cont'd)

Subprogram	Params	Locals	Static Parent
Bigsub	--	--	
A	Flag	--	Bigsub
B	--	--	A
C	--	--	B
D	--		C

NOTE: The example code doesn't show any variable declarations. We will show where the locals would go, if there are any, by showing a single local storage of one word for each subprogram.

ARI	Address	Value	Content	Description
D	20	15	Dynamic Link	Link to caller's ARI: C
	19	15	Static Link	Link to parent's ARI: C
	18		Return (to C)	
C	17	11	Dynamic Link	Link to caller's ARI: A
	16	8	Static Link	Link to parent's ARI: B
	15		Return (to A)	
A	14	false	Param	Flag
	13	8	Dynamic Link	Link to caller's ARI: B
	12	1	Static Link	Link to parent's ARI: Bigsub
	11		Return (to B)	
B	10	4	Dynamic Link	Link to caller's ARI: A
	9	4	Static Link	Link to parent's ARI: A
	8		Return (to A)	
A	7	true	Param	Flag
	6	1	Dynamic Link	Link to caller's ARI: Bigsub
	5	1	Static Link	Link to parent's ARI: Bigsub
	4		Return (to Bigsub)	
Bigsub	3	0	Dynamic Link	Link to caller's ARI: Main
	2	0	Static Link	Link to parent's ARI: Main
	1		Return (to Main)	
Main	0		Local	(place holder)

Problem 10.4

4. Show the stack with all activation record instances, including the dynamic chain, when execution reaches position 1 in the following skeletal program. This program uses the deep-access method to implement dynamic scoping.

```
void fun1() {
  float a;
  . . .
}

void fun2() {
  int b, c;
  . . .
}
void fun3() {
  float d;
  . . . <------------- 1
}

void main() {
  char e, f, g;
  . . .
}
```

The calling sequence for this program for execution to reach `fun3` is

```
main calls fun2
fun2 calls fun1
fun1 calls fun1
fun1 calls fun3
```

Problem 10.4 (cont'd)

Subprogram	Params	Locals
main	--	e, f, g
fun1	--	a
fun2	--	b, c
fun3	--	d

ARI	Address	Value	Variable	Content	Description
fun3	15		d	Local	
	14	11		Dynamic Link	Link to caller's ARI: fun1
	13			Return (to fun1)	
fun1	12		a	Local	
	11	7		Dynamic Link	Link to caller's ARI: fun1
	10			Return (to fun1)	
fun1	9		a	Local	
	8	3		Dynamic Link	Link to caller's ARI: fun2
	7			Return (to fun2)	
fun2	6		c	Local	
	5		b	Local	
	4	0		Dynamic Link	Link to caller's ARI: main
	3			Return (to main)	
main	2		g	Local	
	1		f	Local	
	0		e	Local	

Problem 10.5_____

5. Assume that the program of Problem 4 is implemented using the shallow-access method using a stack for each variable name. Show the stacks for the time of the execution of `fun3`, assuming execution found its way to that point through the sequence of calls shown in Problem 4.

fun1						
fun1	fun2	fun2	fun3	main	main	main
a	b	c	d	e	f	g