Amara Tariq
CS354
Buffenbarger
March 19th, 2020

TA2

3.2) These differences are mainly due to the fact that each language was built and designed to interpret and function differently. Each language has certain reasons why it would work a specific way.

```
//Pascal Example
int i:=Value#1;
void someFunction(){ //if static allocation
  int i:=Value#2;    //wouldnt work because the variables share the same name
}                    //but equal 2 difference values and are 2 different variables
```

```
;;Scheme Example
(define (sum seq)
  (if (null? seq)
    0
    (+(car seq) (sum(cdr seq)))
  )
)

(display (sum '(5 6 1 8 3 7)))  ;;because of the way recursion in this program works
(display "\n")                  ;;wouldn't run as it is designed to
```

3.4)

a)

```
int main(){
  //some code
  CallToSomeMethod();
  //some code
}

int CallToSomeMethod(){
  static int i=10;     //int i is created/ alive in memory but not
  i--;                 //for the scope after this method returns
}
```

b)

```
public class someClass{
  int i=0;
  public static void main(String args[]){
    System.out.print(i); //i is created/alive in memory but not within the scope
                         //because i is an instance variable
  }
}
```

c)

```java
public class someClass{
  public static void main(String args[]){
    int i=10;
    System.out.print(i); //this is where i is within the scope
  }
  System.out.print(i); //i is alive in memory but this location is where it is out of scope
}
```

3.5)

Using the C rules for declaration-order, what should be printed should be:

1 1 //a =1 b=1
3 1 //a =3 b=1
1 2 //a =1 b=2

Using the Modula-3 rules for declaration-order, what should be printed should be:

3 3 //a =3 b=3 //because modula-3…
3 3 //a =3 b=3
1 2 //a =1 b=2

3.7)

a) Every time reverse() is called, its creating a new list. So when Brad takes what is returned from that function and reassigns it to L, the list L previously pointed to is lost in memory.

b) Janet will tell him that now with his call to delete_list(L), he is deleting the pointer that is pointing/referencing to the list, and then the pointer T that he created prior to the deletion is now pointing to the same space L was pointing to. Now there are dangling references to the nodes that were in the previous L list.

3.14)

With static scoping, '1020' will be printed out because the variable 'x' is declared globally, where it can not be accessed by the functions. With dynamic scoping, '1122' will be printed out because where the calls are made which access the variable x, those places are within the scope of x. In dynamic scoping, variables are not specific for the scope they are defined, calls outside of the scope of declaration can still access them.

3.18)

With shallow binding, the functions that call x, set_x and print_x, are accessing the local x, not the global x, thus what is printed out is:

1 0
2 0
3 0
4 0

With deep binding

1 0
5 2
3 3