Prince Kannah
CS-354: Programming Languages
HW3

<u>P167: 3.1</u>
Using the C language as an example:

- The number of built in functions (math, type queries etc): The number of built-in functions bond at language design time. A number of built-functions are defined by the standard library.
- The variable declaration for pointing to a particular variable reference: The variable declaration is bound at compile time in C which uses static scoping.
- The maximum length allowed for a constant character string: The maximum length for a constant character string is decided at the time of implementing a language.
- The referencing environment to implement a subroutine that is passed as a parameter: C uses static scoping and static scoping rules specifies that the referencing environment depends on the nesting of subroutines in which these are declared. This decision is likely made during the time of writing a program.
- The address of a particular library routine: The address of any library function is bound at link time because we really don't know until run time.
- The total amount of space occupied by a program and its data: As binding of values to variables occur at run time, this decision is bound at run time.

<u>P167: 3.4</u>
Scope is the area or part of the program where a variable can be accessed. The life of a variable is the time until which a variable in a program is valid or till the memory is allocated for it.

The first example is a C program where a static variable *sum* is declared in a function has life when execution is not inside the function, but it is not in the scope at that time.

```
// Header files
void sum(void);

int main(){
  for(int i = 0; i < 3; i++){
       //Call the function
               sum();
  }
  getch();
  return 0;
}
void sum(void){
 // Declaring static variable sum. Its life is throughout the program
// Here the static integer sum has life but not in scope when the for-loop
// is executed in main. The value of sum is valid for the whole program and does // not
initiate over and over every time the loop is called.
```

```
    static int sum;
    int n;
    // the sum of number provided till now
    s = s + n;
  }
```

The second example is C++ where a private field variable is live but outside of scope when execution is outside the method of class.

```
// Header files
class Example {
  private int mem = 10;
  public int getMem(){ return mem;}

  int main(){
       // At this point the field member 'mem' is live but
       // not in scope
       Example ex;

       int num = 10;
       cout <<endl<<"The value of num is "<<num;
       // Adding num with value returned from method call
       num = num + ex.getMem(); return 0;
  }
};
```

The final example is similar to example two except it's done using java

```
public class Example {
  private String name = "Prince";
  public String getName(){return name;}

  public static void main(String[] args){
       // Similar to example two, the name variable is live
       // but not in scope
       Example n = new Example();
       // brings name into current scope
       System.out.println("Hi " + n.getName());
  }
}
```

<u>P167: 3.5</u>

Using the declaration-order rules of C, names are declared before use and their scope extends from their declaration until the end of the block. So, with that the execution starts with main and *a* and *b* are initially assigned values 1 and 2 respectively. Inside of main we call the middle() function which assigns b to the value of a (b is now 1). The inner() function is then called which prints 1, the current value of a and b. At line 7 the values of a and b are from lines 2 and 5 respectively thus the print statement at line 11 prints a as 3 and b as 1 after taking declaration of a from line 8 and b from line 5. At this point control returns to the main block and scope of variables *a* and *b* in the middle() function ends there. Here values of a and b will be as they were initially declared on lines 2 and 3. So at line 14 a is printed as 1 and b is printed as 2. Using C# the middle() method on line 5 will result in an error because b is assigned a value a even before a's declaration. This will also lead to an error on line 7 of the inner() method since printing a and b before declaring is not allowed. Things are a bit different with Modula-3. Because the inner() method resides in the middle() method and there is no declaration of a and b in the inner() scope it will use reference environmental of the middle() and hence print statement at line 7 takes the declaration of a from line 8 and b from line 5 thus printing 3 and 3 respectively. This also applys to the print statement on line 11. Since the print statement on line 14 is within the main block the declarations for a and b are taken from the scope of main, meaning a is 1 and b is 2.

<u>P169: 3.6a</u>
The execution starts with main where the B() method is called with 3 as a parameter. Within B() 3 is assigned to a followed by the computation $a \times a$ giving the output of 9.

<u>P171: 3.14</u>
If the language uses static scoping then the global variable x is used to store the values and has life throughout the program. The final output of the program will be $1122$ . If dynamic scoping is used the output depends on the most recent, active binding for x. That output depends on whether the variable assignment is done locally or globally. Once the second() method ends control is returned to main. So, in the final statement print_x will print the global value of x which is 1 as set in the first() method. The final output using dynamic scoping is $1121$