Homework 5  10 points

Note, these exercises may be done in groups of one, two, or three. Working with someone else is strongly recommended.

Scope
1. Consider this Pascal-like program in which each method has the following syntax: header, declarations (including declarations of local procedures), body:

   program Example;
   var
     x: integer; y: float;
   procedure First;
     begin
       writeln(x, " ", y);
     end;
   procedure Second;
     var
       x: integer;
       y: string;
     procedure Third;
     var
       x: integer;
     begin { Third }
       x := 99;
       First
     end; { Third }
     begin { Second }
       x := 88; y = "happy";
       Third
     end { Second }
     begin { Example }
       x := 77; y = 10.7;
       First();
       Second();
     end. { Example }

   a) What will be printed by this program if we assume static scope?
   b) What will be printed by this program if we assume dynamic scope?

2. Using a stack of symbol tables, show what the symbol tables look like at each of the points indicated in the C code below. In showing a symbol table, identify which variables and functions (and their types) are known. If there are multiple symbol tables active, show all of them. Assume brackets open/close a new scope.

   int a, b     // global to entire program
   int p(void)
```c
{ int a,p;                      // local to scope of function p
  a=0 ;b=1;p=2 ;
  { float b :
    a=3 ; b=4.0;
    print a+b
  }
}

return p ;
}

main()
{float  z;
  a  = p() ;
  q() ;
}
```

3. Match each term or concept in the left list with the phrase in the right list which best defines or describes it. (Not all definitions will be used.)

<table>
<thead>
<tr>
<th>activation record</th>
<th>a. the association of an identifier with a memory location</th>
</tr>
</thead>
<tbody>
<tr>
<td>aliasing</td>
<td>b. the meanings attached to syntactic constructs</td>
</tr>
<tr>
<td>curried</td>
<td>c. a change to the global environment which occurs in a function call</td>
</tr>
<tr>
<td>dangling reference</td>
<td>d. the region of a program over which a binding is visible</td>
</tr>
<tr>
<td>garbage</td>
<td>e. the association of multiple identifiers with a memory location</td>
</tr>
<tr>
<td>lambda expressions</td>
<td>f. breaking source text into tokens</td>
</tr>
<tr>
<td>lazy evaluation</td>
<td>g. the environment of a sub-program call</td>
</tr>
<tr>
<td>lexical analysis</td>
<td>h. rules for forming language structures from tokens</td>
</tr>
<tr>
<td>lifetime</td>
<td>i. syntactic analysis; combining tokens to form syntactic structures</td>
</tr>
<tr>
<td>linking</td>
<td>j. rules by which characters are combined to form tokens</td>
</tr>
<tr>
<td>list comprehension</td>
<td>k. function in which arguments are partially applied.</td>
</tr>
<tr>
<td>name binding</td>
<td>l. giving a symbol two or more meanings</td>
</tr>
<tr>
<td>overloading</td>
<td>m. a true-or-false expression</td>
</tr>
<tr>
<td>parsing</td>
<td>n. resolves references to library routines</td>
</tr>
<tr>
<td>polymorphic</td>
<td>o. a function which accepts another function as a parameter or returns a function as a result</td>
</tr>
<tr>
<td>predicate</td>
<td>p. the period of time that an object is bound to an address</td>
</tr>
<tr>
<td>pure</td>
<td></td>
</tr>
<tr>
<td>scope</td>
<td></td>
</tr>
<tr>
<td>semantics</td>
<td></td>
</tr>
<tr>
<td>side effect</td>
<td></td>
</tr>
</tbody>
</table>
q. a method of finding some mapping of constants to variables which will make two logical expressions equivalent
r. polymorphic
s. the computing of argument values only if needed
t. a location which has been de-allocated from memory
u. memory that has become inaccessible
v. function produces the same value given the same arguments
w. different data types can be handled using a uniform interface
x. unnamed functions
y. creating a list from other lists

4. Haskell is described as a **fully-Curried** lazy purely functional language with Hindley-Milner static typing. Explain each part of this description.

5. What are the obvious advantages and disadvantages of not having statically declared types in a language?

6. Give the typical binding times (in C++) for the following attributes:

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Possible binding times</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The maximum number of significant digits in a real number</td>
<td>- Language definition</td>
</tr>
<tr>
<td>2. The meaning of an expression</td>
<td>- Language implementation</td>
</tr>
<tr>
<td>3. The size of an array variable</td>
<td>- Translation time</td>
</tr>
<tr>
<td>4. The total size of an array which is passed as a parameter</td>
<td>- Link time</td>
</tr>
<tr>
<td>5. The location of a local variable</td>
<td>- Load time</td>
</tr>
<tr>
<td>6. The value of a constant</td>
<td>- Execution time - dynamic</td>
</tr>
<tr>
<td>7. The location of the function code</td>
<td></td>
</tr>
</tbody>
</table>

7. Pointers present problems for overload resolution. Consider the following C++ code:

```cpp
void f (int * x) {...}
void f(char* x) {...
int main()
{ f(0);
}
```

What is the concern with this code? How might you fix it?
8. (We haven’t covered this yet. Don’t do yet.)
Here are some type and variable declarations in Pascal-like syntax. State which variables are
equivalent under (a) name equivalence (b) structural equivalence. Note, that two variables
declared in the same declaration (consisting of a variable list and a single type specifier) are
name equivalent (even if the type is elaborated explicitly and not given a user defined name).
We say two types are also name equivalent if they lead back to same declaration through
renaming. (If the exact same unnamed type is repeated on a separate line, the two variables
are NOT name equivalent.)

\begin{verbatim}
type
  range = -5 .. 5;    // This is an example of a subrange type.
  levels = 0..10;
end;

// Variables of type “range” and integers between -5 and 5
  table1 = array[range] of char;
  table2 = table1;
  table3 = array[levels] of char;
var
  x,y : array[-5..5] of char;
  z: table1;
  w: table2;
  q,r,s: table3;
  i: range;
  j:-5..5;
  m: table1;
\end{verbatim}

9. Suppose we were to try to write a short-circuit version of \texttt{and} as the following function:
\begin{verbatim}
boolean function and(a,b:boolean)
{ if a then return b
  else return false;
}
\end{verbatim}
Why doesn’t this accomplish short circuit evaluation? Would it work if lazy evaluation were
used? Explain.

10. Explain how a switch control statement can have constant time (independent of number
    of cases) to make the correct selection?

11. Given an array $T(6..10, 0..20)$ stored in \texttt{column major}, what is the accessing formula (be
    sure to simplify to produce a virtual origin, a multiplier for $i$ and a multiplier for $j$)? Note,
    the notation $6..10,0..20$ means the first subscript begins at 6 and ends at 10, while the
    second subscript begins at 0 and ends at 20. Assume each element of the array takes
    three locations (element size) and begins at location 0. Use the accessing formula to
    compute where $T(8,0)$ is located.