Spring 2023
Programming Languages
Homework 4

- This homework is a combination programming and "paper & pencil" assignment.
- Due via Brightspace on Sunday, April 30, 2023 11:59 PM Eastern Time. Due to timing considerations relating to the end of the semester, late submissions will not be accepted. **No exceptions.**
- For the Prolog questions, you should use SWI Prolog. Link is available on the course page.
- For the Java questions, you should use Java 8 or later.
- You may collaborate and use any reference materials necessary to the extent required to understand the material. All solutions must be your own, except that you may rely upon and use Prolog code from the lecture if you wish. Homework submissions containing any answers or code copied from any other source, in part or in whole, will receive a zero score and be subject to disciplinary action.
- Please submit your homework as a zip file, `hw4-<netid>.zip`. The zip file should contain:
  - For Q1, `hw4-<netid>-Q1.pl` which should contain the reordered facts for `part (1)`.
  - For the `Prolog Rules` question (Q2), `hw4-<netid>-Q2.pl` which should contain your implementation of all of the rules. You do not need to submit queries, although we will run our queries on your solutions.
  - For Q4, `hw4-<netid>-Q4.pl` which should contain all *rules* necessary to make the solution work.
  - For Q5, submit a file `DownloadFile.java`.
  - For (Q1, Q3, Q5, Q6 and Q7), a PDF file `hw4-<netid>-sols.pdf` containing the answers to all of the "paper & pencil" questions.
- Make sure your Prolog code compiles before submitting. *If your Prolog code does not compile for any reason, it may not be graded.*

1. [10 points] **Investigating Prolog**

Now, consider a subset of employees in the form of Prolog facts and rules:

```
engineer(brian).
engineer(kevin).
engineer(zhane).
manager(mary).
manager(emily).
senior_manager(sarah).
president(bob).
president(jane).

senior(mary, brian).
senior(jane, emily).
senior(jane, mary).
senior(emily, kevin).
senior(emily, zhane).

president_engineer_relation(X,Y) :- president(X), senior(X,Z), senior(Z,Y), engineer(Y).
```

`Note:` Assume that the universe of facts is **not** complete.

1. Reorder the facts above to provide faster execution time when querying `president_engineer_relation(jane,brian).` List the re-ordered facts and briefly explain what you changed.

   The re-ordered facts are as follows,

   ```
       engineer(brian).
   engineer(kevin).
   engineer(zhane).
   manager(mary).
   manager(emily).
   senior_manager(sarah).
   president(bob).
   president(jane).

   senior(jane, mary).
   senior(mary, brian).
   senior(jane, emily).
   senior(emily, kevin).
   senior(emily,zhane).
   ```

   All that I changed was the order of the senior facts, making *senior(jane,mary)* appear first. This way, the query doesn't explore the senior tree that follows from the previous order (jane to emily, emily to brian), which fails, and thus the query has to backtrack.

2. Explain in your own words why the change above affects total execution time. Show evidence of the faster execution time (provide a trace for each).

   The changed order of the facts provides a more efficient query, as the relation between jane and emily is not explored. In the previous ordering, the subgoal of senior(jane,emily) evaluated to true, and thus the following subgoal of senior(emily, brian) was explored. Emily is not brians senior, thus this subgoal ended failing and the query needed to backtrack to explore the next senior relation belogning to jane (which is mary). Mary is brians senior, thus the query could continue to execute to completion and return true. The re-ordering of the facts eliminates this backtracking, and results in less execution time.

Before any changes the trace is as follows:

```
    [trace]  ?- president_engineer_relation(jane,brian).
   Call: (10) president_engineer_relation(jane, brian) ? creep
   Call: (11) president(jane) ? creep
   Exit: (11) president(jane) ? creep
   Call: (11) senior(jane, _34774) ? creep
   Exit: (11) senior(jane, emily) ? creep
   Call: (11) senior(emily, brian) ? creep
   Fail: (11) senior(emily, brian) ? creep
   Redo: (11) senior(jane, _34774) ? creep
   Exit: (11) senior(jane, mary) ? creep
   Call: (11) senior(mary, brian) ? creep
   Exit: (11) senior(mary, brian) ? creep
   Call: (11) engineer(brian) ? creep
   Exit: (11) engineer(brian) ? creep
   Exit: (10) president_engineer_relation(jane, brian) ? creep
true.
```

After the changes:

```
    [trace]  ?- president_engineer_relation(jane,brian)
|     .
   Call: (10) president_engineer_relation(jane, brian) ? creep
   Call: (11) president(jane) ? creep
   Exit: (11) president(jane) ? creep
   Call: (11) senior(jane, _30088) ? creep
   Exit: (11) senior(jane, mary) ? creep
   Call: (11) senior(mary, brian) ? creep
   Exit: (11) senior(mary, brian) ? creep
   Call: (11) engineer(brian) ? creep
   Exit: (11) engineer(brian) ? creep
   Exit: (10) president_engineer_relation(jane, brian) ? creep
true
```

3. Can we define a new rule `director_engineer_relation(X, Y)` that calls into the goals presented above to correctly arrive at an answer? Why or why not?

   At this point we could not define a new rule `director_engineer_relation(X, Y)` as we have no facts relating to directors, and no senior relations to link directors to subordinates. If we did have the aforementioned facts, we could feasibly write a new

2. [20 points] **Prolog Rules**

   Write the Prolog rules described below in a file `hw4-<netid>-Q2.pl`. All rules must be written using the subset of the Prolog language discussed in class. You may not, for example, call built-in library rules unless specifically covered in the slides. You may call your own rules while formulating other rules. You do not need to turn in queries, although you should certainly test your rules using your own queries.

   1. Write a rule `remove_items(I,L,O)` in which O is the output list obtained by removing every occurrence of each item in List I from list L. The items in O should appear in the same order as L, only without the elements present in I.

      ```
      Example:
      ?- remove_items([1,3],[2,6,7,7,8,3,1,1,4,3],O).
      O = [2,6,7,7,8,4]
      ```

   2. Write a rule `my_flatten(L1,L2)` which transforms the list L1, whose elements could be nested lists, into a "flat" list L2 by replacing each list with its elements.

      ```
      Example:
      ?- my_flatten([a, [b, [c, d], e]], X).
      X = [a, b, c, d, e]
      ```

   3. Write a rule `compress(L1,L2)` where all the repeated consecutive elements of the list L1 should be replaced with a single copy of the element. The order of the elements should not be changed.

      ```
      Example:
      ?- compress([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).
      X = [a,b,c,a,d,e]
      ```

   4. Write a rule `encode(L1,L2)` where all the repeated consecutive duplicates of elements are encoded as terms [N,E] where N is the number of duplicates of the element E. The order of the elements should not be changed.

      ```
      Example:
      ?- encode([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).
      X = [[4,a],[1,b],[2,c],[2,a],[1,d][4,e]]
      ```

   5. Modify the result of previous problem in such a way that if an element has no duplicates it is simply copied into the result list. Only elements with duplicates are transferred as [N,E] terms. Name the rule as `encode_modified`.

      ```
      Example:
      ?- encode_modified([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).
      X = [[4,a],b,[2,c],[2,a],d,[4,e]]
      ```

   6. Write a rule `rotate(L1,N,L2)` where the list L2 is the modified version of list L1, with the elements rotated as N places to the left if N is a positive number, otherwise rotated as N places to the right. You can assume that list L1 is never empty and $|N| < length(L1)$, where $|.|$ is the absolute value operator.

      ```
      Examples:
      ?- rotate([a,b,c,d,e,f,g,h],3,X).
      X = [d,e,f,g,h,a,b,c]

      ?- rotate([a,b,c,d,e,f,g,h],-2,X).
      X = [g,h,a,b,c,d,e,f]
      ```

3. [10 points] **Unification in Prolog**

For each expression below that unifies, show the bindings. For any expression that doesn't unify, explain why it doesn't. Assume that the unification operation is left-associative.

1. a(15) & c(15)

   This will not unify, as a(X) is a different functor than c(X) so a(15) $\neq$ c(15)
2. 53 & X & 76

   This will not unify, as X cannot unify both 53 and 76, that is $53 = X \neq 76$
3. a(X, b(3, 1, Y)) & a(4, Y)

   This unfortunately does not unify. First we have X = 4, but then we have to unify $b(3, 1, Y) = Y$ which would cause an infinite recursion problem as Y appears in both the LHS and the RHS of the expression.
4. b(1,X) & b(X,Y) & b(Y,1)

   This unifies, as first we have $1 = X$, then we have $X = Y$ and $Y = 1$, thus unifying with $X = Y = 1$.
5. a(1,X) & b(X,Y) & a(Y,3)

   This does not unify as a and b are different functors. Additionally, if the middle functor was also A, this would fail to unify as X cannot be both 1 and 3.
6. a(X, c(2, B, D)) & a(4, c(A, 7, C))

   This unifies, with $X = 4$, $A = 2$, $B = 7$ and $D = C$
7. e(c(2, D)) & e(c(8, D))

   This does not unify, as 2 doesn't unify with 8, that is: $2 \neq 8$
8. X & e(f(6, 2), g(8, 1))

   This unifies with $X = e(f(6, 2), g(8, 1))$
9. b(X, g(8, X)) & b(f(6, 2), g(8, f(6, 2)))

   This unifies with $X = f(6, 2)$
10. a(1, b(X, Y)) & a(Y, b(2, c(6, Z), 10))

    This does not unify for several reasons, first if Y must equal 1 to unify the first atom in a, then $Y \neq c(6, Z), 10$. Also, the arity (number of args) of b is not the same in each expression.
11. d(c(1, 2, 1)) & d(c(X, Y, X))

    This unifies with $X = 1$, $Y = 2$

4. [20 points] **Make your own rules - Prolog**

   Consider a list in Prolog with the following constraints:

   1. Length of the list is 4.
   2. Every entry in the list is a single lowercase English character between `a` to `j` (both `a` and `j` inclusive).
   3. The list has a few characters in common with the following lists:
      (a) `[i, j, f, b]` - DD
      (b) `[c, b, g, j]` - DS
      (c) `[d, g, j, e]` - DS
      (d) `[e, h, c, b]` - DS
      (e) `[b, c, d, h]` - DDD

      Here, `D` means one character in common but at different place and `S` means one character in common in the same place.

   Your task is to find the list(s) that satisfy all the above constraints using Prolog. Specifically, write a Prolog rule, `solution([A, B, C, D])` that returns the list. Here, `A` binds to the first character in the list, `B` binds to the second character and so on. If there are multiple lists satisfying the above constraints, your solutions should return all such lists.

   `For example:` A list that satisfies the constraint 3(b) (`[c, b, g, j]` - DS) is `[c, j, d, e]`. Here, `j` is the character in common to both the lists but at different place and `c` is the character in common at the same place.

   `Note:`

   1. `Hint:` Try to formalize the constraints in the form of Prolog facts and rules.
   2. The list manipulation library (`https://www.swi-prolog.org/pldoc/man?section=lists`) can help simplify your code.
   3. You can make your own helper rules as well.
   4. Your code must be your own work.

5. [15 points] **Java Exceptions**

Consider the following Java program:

```
import java.net.URL;
import java.io.BufferedInputStream;
import java.io.FileOutputStream;

public class DownloadFile {
    private static final String FILE_URL = "https://cs.nyu.edu/~av2783/pl_hw4_q5.txt";
    private static final String BKP_FILE_URL = "https://cs.nyu.edu/~av2783/pl_hw4_q5_bkp.txt";
    private static final String FILE_NAME = "pl_hw4_q5_local_copy.txt";

    public static void main(String args[]) {
        BufferedInputStream in = new BufferedInputStream(new URL(FILE_URL).openStream());
        FileOutputStream fileOutputStream = new FileOutputStream(FILE_NAME);
        byte dataBuffer[] = new byte[1024];
        int bytesRead;
        while ((bytesRead = in.read(dataBuffer, 0, 1024)) != -1) {
            fileOutputStream.write(dataBuffer, 0, bytesRead);
        }
    }
}
```

This program is supposed to read a file on a given URL and write its contents to a local file in your machine. However, it fails to compile. The reason is the absence of exception handling. Answer the following questions:

1. List all the checked exceptions that can be thrown in this program. You can find some useful information about the classes being used in the above program in the Java documentation(`https://docs.oracle.com/javase/8/docs`).

   The checked exceptions are as follows:

   - `java.net.MalformedURLException`
   - `java.io.FileNotFoundException`
   - `java.io.IOException`

2. Add the exception handling code in the above program so that it compiles successfully and behaves as expected at runtime.

3. Sometimes, the `FILE_URL` mentioned in the program is not reachable due to server error. Which exception will be thrown in this case?

   If the `FILE_URL` is not reachable due to a server error, (and not a malformed URL), then the exception that will be thrown is `java.io.IOException`.

4. In case the `FILE_URL` is unreachable, a backup URL must be used to read the file contents. Using the exception handling code of the exception in `part (3)`, add the functionality in your code from `part (2)` to read the contents from `BKP_FILE_URL` when `FILE_URL` is unreachable.

6. [15 points] **Virtual Functions**

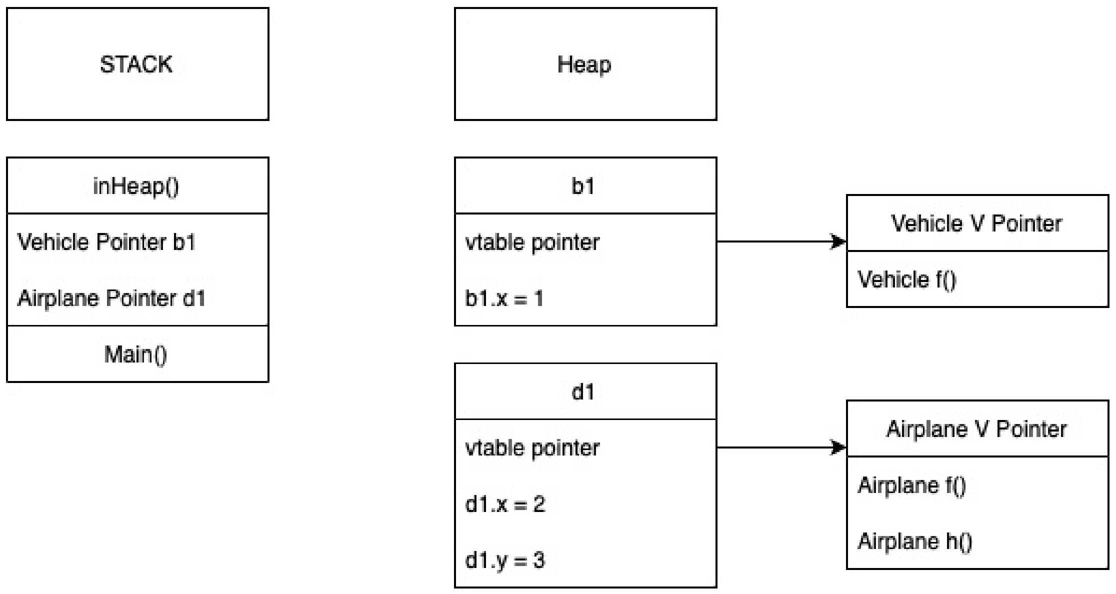
   This problem examines the difference between two forms of object assignment. In C++, local variables are stored on the run-time stack, while dynamically allocated data (created using the new keyword) is stored on the heap.

```
class Vehicle {
public:
    int x;
    virtual void f();
    void g();
};

class Airplane : public Vehicle {
public:
    int y;
    virtual void f();
    virtual void h();
};

void inHeap() {
    Vehicle *b1 = new Vehicle; // Allocate object on the heap
    Airplane *d1 = new Airplane; // Allocate object on the heap
    b1->x = 1;
    d1->x = 2;
    d1->y = 3;
    b1 = d1; // Assign derived class object to base class pointer
}

void onStack() {
    Vehicle b2; // Local object on the stack
    Airplane d2; // Local object on the stack
    b2.x = 4;
    d2.x = 5;
    d2.y = 6;
    b2 = d2; // Assign derived class object to base class variable
}

int main() {
    inHeap();
    onStack();
}
```

   Answer the following questions:

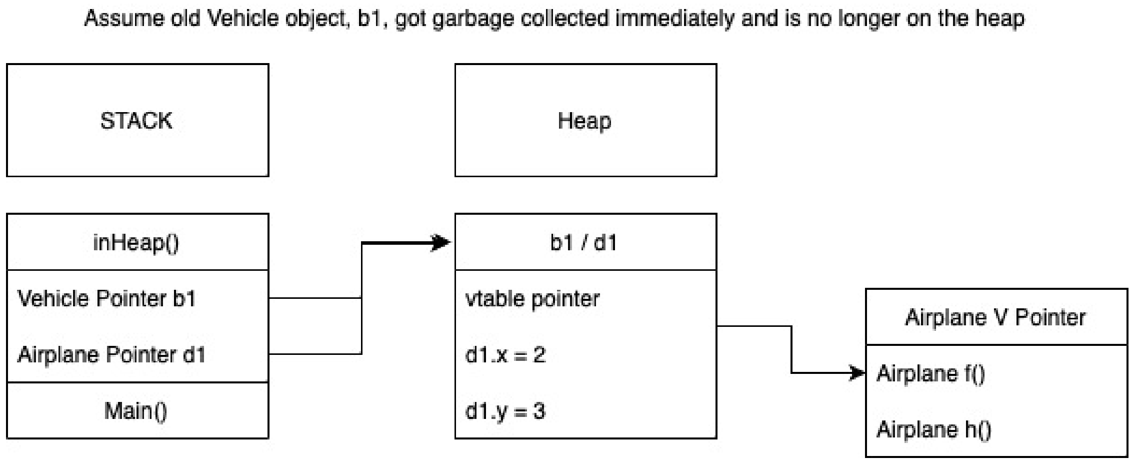
   Note: In the questions below, *vtable pointer* refers to an object's hidden vtable pointer.

   1. Draw a picture of the stack, heap and vtables just before `b1=d1` is executed during the call to `inHeap`. Be sure to indicate where the `instance variables` and `vtable pointers` of the two objects are stored, and to which vtable(s) the respective `vtable pointers` point.



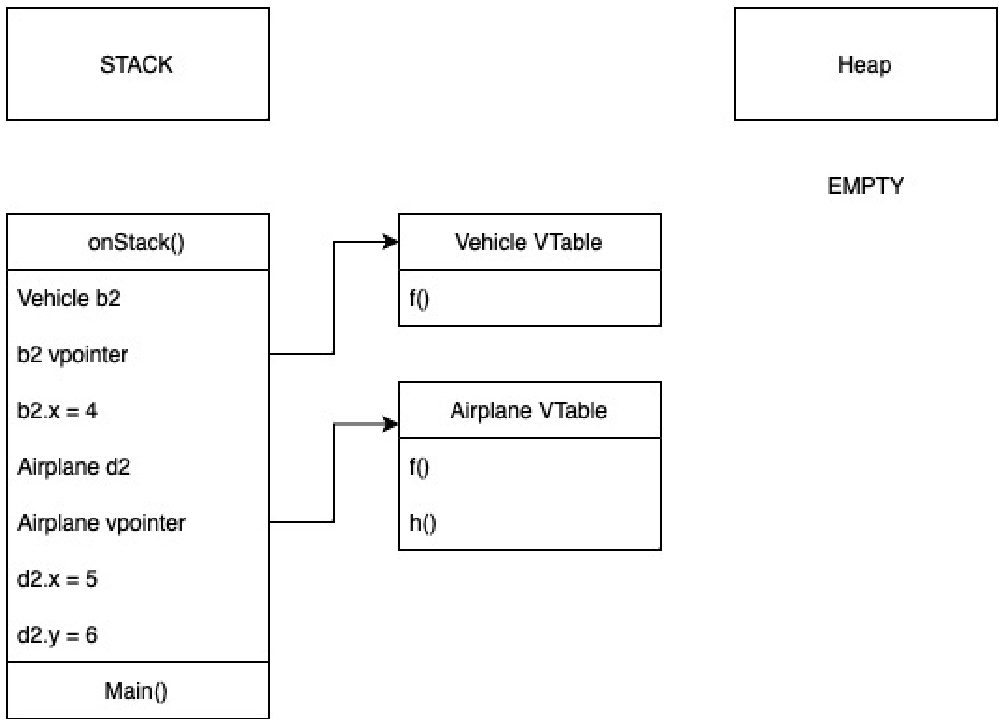
2. Re-draw your diagram from `(1)`, showing the changes that result just after the assignment `b1=d1`. Be sure to clearly indicate where `b1`'s vtable pointer points. Explain why `b1`'s vtable pointer points where it does.

   In my illustration, I show what happens when the pointer value in `b1` is reassigned to the pointer value in `d1`. In my example, I assume that garbage collection happens immediately, and thus the old `b1`'s object (a Vehicle object) got cleaned up immediately. The `b1` pointer (which the program in the stack seees as a Vehicle Object), really behaves as a Airplane object, as the `vtable pointer` points to the `Airplane` V Table.



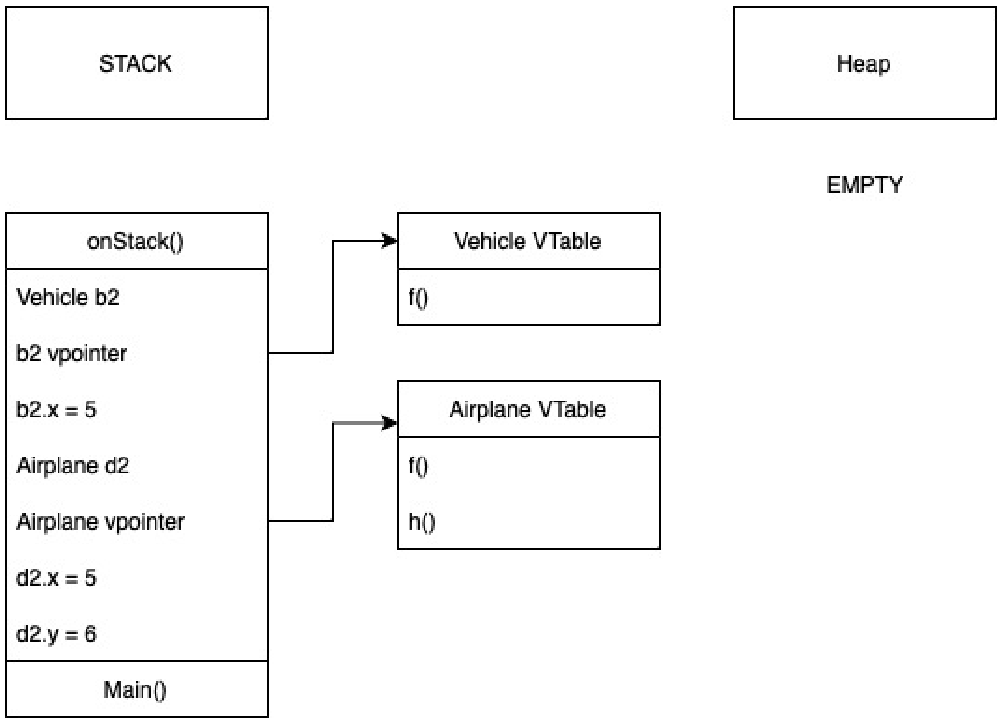
3. Draw a picture of the stack, heap and vtables just before `b2=d2` is executed during the call to `onStack`. Be sure to indicate where the `instance variables` and `vtable pointers` of the two objects are stored, and to which vtables the respective `vtable pointers` point.

   Here we assume that garbage collection has occurred, thus the unreachable objects `d1/b1` on the heap have been cleaned up.



4. Re-draw your diagram from `(3)`, showing the changes that result after the assignment `b2=d2`. Be sure to clearly indicate where `b2`'s vtable pointer points. Explain why `b2`'s vtable pointer points where it does.

   After we assign `b2=d2`, we see something different happen than we did in the `inHeap()` function call. Since we're not dealing with object references, when we overwrite the value in `b2` with `d2`, we actually have a case of C++ Coeercion happening. The copy constructor is implicitly called, and the object `d2` (an `Airplane` object) is "sliced" to fit into a `Vehicle` object. Thus, `b2` is still a `Vehicle` object, whos vpointer points to a `Vehicle` Vtable. The value in `x` is overwritten, and `b2` does not have a `y` instance variable.



5. We have used the assignment statements `b1=d1` and `b2=d2`. Why are the opposite statements `d1=b1` and `d2=b2` not allowed?

   This is because the direction of inheritance, upcasting is allowed from a subclass to be upcast to its parent classes's type, as it inherits the parents methods so the methods or member functions invoked from the pointer are consistent (everything a parent can do the child can as well). However, a parent class does not necessarily contain the same functionality and behavior as its child classes, thus why downcasting (going from parent to child), is not allowed in most langugages.

7. [10 points] **Prototype OOLs**

Consider the following code below, written in a prototype OOL:

```
var obj1;
obj1.x = 20;
var obj2 = clone(obj1);
var obj3 = clone(obj2);
var obj4 = clone(obj1);
obj2.y = 5;
obj4.x = 10;
obj3.z = 30;
```

Assume that the program fragment above has executed. Answer the following:

1. What fields are contained locally to `obj1`?

   Only the field, x, is contained locally in obj1.
2. What fields are contained locally to `obj2`?

   Obj2 has the fields X and Y, but only the field Y is locally stored in Obj2.
3. What fields are contained locally to `obj3`?

   Obj3 has the fields X, Y, and Z, but only the field Z is locally stored in Obj2.
4. What fields are contained locally to `obj4`?

   Obj4 only has the X field, which is stored locally as it its initial value was overriden.
5. To what value would `obj1.x` evaluate, if any?

   `obj1.x` $= 20$
6. To what value would `obj2.x` evaluate, if any?

   `obj2.x` $= 20$
7. To what value would `obj3.x` evaluate, if any?

   `obj3.x` $= 20$
8. To what value would `obj4.x` evaluate, if any?

   `obj4.x` $= 10$
9. To what value would `obj4.y` evaluate, if any?

   Y is undefined in obj4.
10. To what value would `obj2.y` evaluate, if any?

    `obj2.y` $= 5$
11. To what value would `obj3.y` evaluate, if any?

    `obj3.y` $= 5$
12. To what value would `obj3.z` evaluate, if any?

    `obj3.z` $= 30$