Brock University

Final Examination, Fall 2004
Course: COSC 2P90
Date of Examination: Dec. 13, 2004
Time of Examination: 4:00 pm

Number of Pages: 4
Number of Students: 67
Number of Hours: 3
Instructors: D. Hughes, M. Winter

Instructions

1) **Place all answers in the examination booklet provided.**
2) **Attempt question 10 and *any seven* of questions 1 through 9.**
3) **Unless required to write an entire class, you do not have to write `import` statements when using routines from the standard Java libraries.**
4) **You do not have to comment your code.**
5) **The paper totals 85 marks. The marks for each question are indicated on the left (e.g. [10]) and the breakdown within the question in italics (e.g. *(2)*).**
6) **No examination aids, specifically no electronic devices including calculators and electronic dictionaries, are permitted. Use or possession of unauthorized materials will automatically result in an award of zero for this examination (FHB 5.1.2.A).**
7) **A mark of 40% must be achieved on this examination in order to obtain a passing grade in the course.**

You may answer any *seven* of questions 1 through 9.

[10] 1. What are the three alternatives for binding times in programming languages*(3)*? Differentiate between the scope and extent of a variable*(2)*. Differentiate between statically-typed and dynamically-typed languages in terms of bindings and binding times*(2)*. Java and C++ take different approaches to recovery of dynamically-allocated storage, what are these approaches and what are the advantages and disadvantages of each*(3)*?

[10] 2. Abstraction mechanisms in programming languages have evolved from procedures through modules to classes. What has been the consistent theme in this evolution (i.e. what has been the reason for the development from one type of language to another)*(2)*? How is each an improvement on the previous (including procedures over none)*(3)*? There are three common parameter passing mechanisms: call-by-value, call-by-value-result and call-by-reference, describe the mechanism for each*(3)*. Large structures, such as arrays, can be expensive to pass using call-by-value. Compare the solution to this efficiency issue for input parameters as addressed in Pascal and Ada*(2)*.

[10] 3. Variant records in Pascal and Ada provide a limited form of polymorphism. For example, the following Pascal code segment provides for two kinds of shapes that can be manipulated in a graphical environment.

```
type ShapeKind = (rectangle,circle);
type Shape =   record
                 xc : integer;  (* x-coordinate *)
                 yc : integer;  (* y-coordinate *)
                 case kind : ShapeKind of
                    rectangle : ( width : integer; height : integer );
                    circle : ( radius : integer );
             end;
var shapes : array [1..100] of Shape;
```

... 2

```
procedure translate ( s : Shape; x, y : integer ) {
   s.xc := s.xc + x;
   s.yc := s.yc + y;
};
function area ( s : Shape ) : integer {
   case s.kind of
      rectangle : area := width * height;
      circle :    area := PI * radius * radius;
   end;
end;
      :
      shapes[i] := …
      :
      translate(shapes[i],dx,dy);
      :
      a := area(shapes[i]);
```

Rewrite the above code in Java to provide flexibility for working with shapes*(5)*. Compare the implementation (that is, in terms of the compiler-generated code) of the Pascal and Java versions. Specifically discuss how the `Shape` objects are represented and how the correct code for the body of the methods (procedures) is determined*(3)*. Describe how the code would have to be modified for each version (Pascal, Java) of the program to support an additional kind of shape (say a polygon centered on `(xc,yc)`)*(2)*.

[10] 4. A subclass can be considered a subtype if it fulfills the principle of substitutability. What is the weak form of the principle of substitutability*(2)*? What is necessary in the definition of a subclass to fulfill the weak principle of substitutability*(3)*? Describe why object-oriented languages usually involve: heap-based allocation, reference semantics for assignment and equality testing and garbage collection*(3)*. Why is extension by composition (aggregation) favored over extension by inheritance*(2)*.

[10] 5. Choose *one* of the following Design Patterns: Object Adapter, Observer and Abstract Factory. For the chosen pattern, describe the problem the pattern addresses*(2)*, give an example of the use of the pattern for a problem (you may include a class diagram)*(4)* and describe how the pattern solves the problem (you may use a class diagram of the model)*(4)*.

[10] 6. Differentiate between a scenario, a use case and a use case diagram*(3)*. Differentiate between the three class relationships: association, aggregation and inheritance*(3)*. What is the purpose of a state diagram*(2)*? What is shown in a sequence diagram*(2)*?

[10] 7. Expand the following application of a LISP function to its arguments using rewrite rules (reduction)*(8)*.

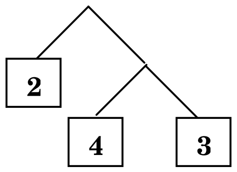
```
(DEF Q (LAMBDA (X Y Z)
   (COND ((NULL X) NIL)
         ((EQ (CAR X) Y) (CONS Z (Q (CDR X) Y Z)))
         (T (CONS (CAR X) (Q (CDR X) Y Z)))
   )
))

(Q '(A B B A) 'B 'C)
```

In general, what does the function `Q` do*(2)*?

...3

[10] 8. Define in Haskell or ML a data type `BinTree` for binary trees that store elements of an arbitrary type `a` at the leaves and the interior nodes of such a tree do not store an element*(2)*. Define a tree `myTree`, which represents the following tree as an element of `BinTree`*(2)*.



Define a function `collapse` that collects the elements stored in a tree into a list*(3)*. (Hint: concatenation of lists is denoted by the infix operations ++ in Haskell and @ in ML.) Implement a higher-order function `foldBinTree`*(3)*. The function should have type:

(Haskell notation)
```
foldBinTree :: (a -> b) -> (b -> b -> b) -> BinTree a -> b
```

(ML notation)
```
val foldBinTree = fn : (‘a -> ‘b) * (‘b * ‘b -> ‘b) -> ‘a BinTree -> ‘b
```

so that `collapse` is equal to:

(Haskell notation)
```
foldBinTree (\x -> [x]) (++)
```

(ML notation)
```
foldBinTree (fn x => [x],op @)
```

[10] 9. Explain how the cut can improve the efficiency of a prolog program*(2)*. Consider the following Prolog program:
```
book(book1).
book(book2).
book(book3).
has(library1,video1).
has(library1,book1).
has(library1,book2).
has(library2,book1).
has(library2,book3).
both_in_library(X,Y,L) :- has(L,X),book(X),!,has(L,Y),book(Y).
```

Generate a proof tree used in answering the following question*(5)*.
```
?- both_in_library(book2,X,library1)
```

Show that the use of a cut in the example above is **not** appropriate (that is, it is a red cut) by giving an example where the predicate `both_in_library` fails and would succeed without a cut*(3)*.

You must answer question 10 (next page)

10. A store producing motorcycles wants to use a program to automate the production of motorcycles according to the existing orders and the availability of the parts needed. A motorcycle has a name (`String`), a number (`int`) of existing orders and list of parts (each together with a multiplicity—the number required to build a motorcycle) needed to produce the motorcycle. A part has a name (`String`), a part number (`int`) and the number "in stock" (`int`). A part may be "on order" in which case it has an expected date of shipment (`Date`) and the number (`int`) of parts ordered. Operations on a motorcycle include accessing (not updating) the name, the number of orders and the list of parts and in/decreasing the number of orders. Furthermore, a motorcycle should provide an operation `produce(n)` that first determines how many motorcycles (say `m`, from `0` to `n`) can be produced with the parts in stock and then decreases the number of orders for the motorcycle and the parts in stock according to `m`. The operation returns `m` as a result. Operations on a part include accessing the name, the part number and the order information attributes described above if the part is on order and accessing and updating the number in stock. Furthermore, a part should provide an operation `order(n,d)` indicating that `n` units of the part are ordered with the expected date of shipment d.

    The store also produces custom built choppers. A chopper is a motorcycle that doesn't initially have a list of parts. A chopper cannot be produced until a designer has added this list of parts.

[8] a. Draw a UML class diagram describing the relationship between the classes described above. Show relationships, multiplicities, association names and the attributes and operations and indicate overriding of methods where appropriate.

[7] b. Give implementations in Java of the `produce` method required by motorcycles and choppers according to the UML diagram that you have provided above.