CS 351: Programming Languages Test 3

Professor Ellen Walker

Spring 2001 12 Week

NAME: ______________________________________________________________

Answer all questions in the space provided. Use the backs of the pages if you need extra space. You may have a single-page one-side crib sheet for this exam.

Please make sure that you have all 6 pages before beginning the test.

If you are stuck on a question, move on and come back to it later.

<table>
<thead>
<tr>
<th>No.</th>
<th>Pts.</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
1. (30 points) Give a specific example (code fragment) that illustrates each of the following concepts. You may use any appropriate programming language, or pseudocode, as long as I can understand your intent. Add enough explanation so that it is clear to me that you understand what the concept is.

- Explicit widening conversion
  ```
  int x;
  double y = (double) x;
  ```

- Binary operator
  ```
  The '+' in a+b;
  ```

- Expression whose value differs depending on associativity (give values for left-associative evaluation and right-associative evaluation)
  ```
  2-4-6 is -8 if left associative, and 4 if right associative
  ```

- Expression whose value differs depending on operator precedence (give at least two different orderings and their results)
  ```
  2+4*6 is 36 if + is higher than * and 26 if * is higher than +
  ```

- Expression where short-circuit evaluation prevents a runtime error
  ```
  while((x>0) && (z!= y/x)) prevents divide by 0
  ```

- Expression with both a side effect and a value (explain what each is)
  ```
  in C++: y=5 - side effect is that the value of y is changed, value is 5
  ```

- Information hiding
  ```
  class list (){  
  public:  
  list();
  //insert, delete, etc. here
  private:  
  int val;
  list *next;
  };
  ```

  The details of the data structure such as the next pointer are hidden from the user of the class.

- A derived class or subtype
  ```
  class circle(shape){
  public:
  area()
  ...
  private:
  ```
double radius;
}; //circle inherits all properties from shape and adds its own
area function and radius member

Polymorphism

shape* Shapes[2];
shapes[0] = new circle(5);
shapes[1] = new rectangle(1,2);
for(int i=0;i<2;i++) cout << i->area() << endl;
//calls circle (not shape) area for the first shape and rectangle
(not shape) area for the second.

Multiple inheritance

class cartoon-dog(cartoon,dog){
}
//inherits properties of cartoon as well as properties of dog
//conflicting properties (i.e. does it talk or bark?) can cause
problems

2. (8 points) In creating an object-oriented language, one design decision is whether to force
everything to be an object, or whether to have both objects and base types.

a. Give a reason that a designer might choose to have nothing but objects.

The overall design would be cleaner and standard properties of object-
oriented languages would take care of things like overloading. The
language would be smaller and more self-consistent.

b. Give a reason that a designer might choose to allow base types as well

Base types are closer to the hardware; they can likely be implemented to
run more efficiently.

3. (10 points) In the language of your choice, write a purely functional program that removes
every other element from a list. (example: [1 2 3 4] becomes [1 3]) Remember, no side
effects allowed!

(defun remove-alt (list)
  (cond ((null list) nil)
        (t (cons (car list) (remove-alt (cddr list))))))
;; this uses the fact that (cdr nil) is nil without error.
4. (10 points) What does the following SmallTalk code do? Explain how the result is accomplished (i.e. what are the objects and what are the messages passed)

```
Last <- 1.
This <- 1.
Count <- 1.
[Count <= 10]
whiletrue: |
| Temp| Temp<- This. This<-This+Last. Last<-Temp |
^This
```

Sets last, this and count to 1.

Next sets up a test object [Count <= 10] and a loop body object (after whiletrue). The loop works by sending a whiletrue message (with the body of the loop as its parameter) to the [Count <=10] message. Unfortunately, since I forgot to include any kind of increment, it would simply create an infinite loop! (Temp is a local variable to the "body" class). If the additional statement Count <- Count+1 were included in the body, then it would add up the numbers from 1-10 and return This, which is then 55.

5. (12 points) One source of confusion for beginning programmers and language switchers is the fact that languages have two different operators (assignment and comparison) that both are represented in English as "equal" and in mathematics as "=".

a. Give an example of a language that uses = as its assignment operator but not its comparison operator. What is the comparison operator for this language? (This is language A for part c of this question).

   C++ uses = as the assignment operator and == for the comparison operator.

b. Give an example of a language that uses = as its comparison operator but not its assignment operator. What is the assignment operator for this language? (This is language B for part d of this question).

   Pascal uses := as the assignment operator and = for the comparison operator.

c. In language A, is the expression x=y=z legal? If so, what is its result?

   Yes - it will set both x and y to the current value of z

d. In language B, is the expression x=y=z legal? If so, what is its result?

   Actually, no (as per the Pascal standard at http://www.pascal-central.com/docs/iso10206.pdf) However, if it were, the result would be to set x to either true or false, depending on whether y and z are the same.
6. (12 points) The "dangling else" problem is a well-known potential ambiguity in programming languages. It can be avoided by carefully specifying the syntax of a given language. Assume that you have the following simple language. The definition of <expression> will be assumed to be similar to C++ style expressions.

<statement> -> <assignment> | <loop> | <if-statement> | <statement-sequence>
<statement-sequence> -> begin <statement> [<statement>] end
<assignment> -> <variable> := <expression>
<loop> -> repeat <statement> while <expression> ] repeat <statement> until <expression>
<if-statement> -> if <expression> then <statement> else <statement>

a. Show that the grammar as written has the dangling-else problem, using an example if statement.

Example: if (a<b) then if (b<c) then result = true else result = false.

```
<statement> -> <if-statement>
- > if <expression> then <statement> else <statement>
- > if (a<b) then <if-statement> else <statement>
- > (etc).
```

Alternately, using the rule I accidentally left out (!)

```
<statement> -> <if-statement>
- > if <expression> then <statement>
- > if <expression> then <if-statement>
- > if <expression> then if <expression> then <statement> else <statement>
- > (etc)
```

b. Rewrite the grammar so that dangling else is avoided by making the above statement illegal in the language, without adding any new keywords.

change the <if-statement> rule to :

```
<if-statement> -> if <expression> then <non-if-statement> else <statement>
```

```
<non-if-statement> -> <assignment> | <loop> | <statement-sequence>
```

c. Rewrite the grammar so that dangling else is avoided by requiring if statements to be explicitly closed.

Change the if-statement rules to

```
<if-statement> -> if <expression> then <statement> else <statement> fi
```

```
<if-statement> -> if <expression> then <statement> fi
```
7. (10 points) The FOR loop, originally a simple counter controlled loop (e.g. FORTRAN's "DO 100 I=1,10") was generalized in many different languages, including Algol 60, C/C++/Java, and Ada. Choose 2 of these 3 languages and describe how the FOR loop in each language has generalized the simple concept of a counting variable.

**Algol 60:** the FOR loop can be a standard counter, a counter with step, or a combination FOR (setting a variable) and WHILE (exiting on a Boolean condition) The main generalization is the exit condition which can now be something other than the counter reaching a predefined value.

**C/C++/Java:** the FOR loop has 3 components, the initialization, the continuation test, and the update operation that occurs at the end of the loop. (Counting was generalized to the update operation, as well as generalizing the exit condition).

**Ada:** the counter in the FOR loop takes on all values of a range type which are not necessarily consecutive, nor are they necessarily evenly spaced (e.g. 1,2,5,6,7,9,13). (This generalizes counting).

8. (8 points) The ideal of "structured programming" is that each code block has one entry and one exit. In practice, however, many languages allow multiple-entry and/or multiple-exit code blocks.

In the C/C++/Java language, give an example of …

a. A multiple-entry code block

```c
switch(x){
    case 1:    do_something1(); break; //entry point1
    case 2:    do_something2(); break //entry point2
}
```

b. A multiple-exit code block

```c
for(i=0;i<10;i++){ //when i=10, second exit point of loop
    if (A[i] = looking_for) return i; //first exit point (of function & loop)
}
return -1; //second exit point (of function)
```