Name Sample Answers

CS471, Programming Language Structure I
Spring, 2005

Final Examination

Closed Book. Answer all the questions. The total points for each question or part of a question follows it in parentheses, thus: *(12 pts)*

1

Consider the following C++ declarations (note, `cout` prints its argument on the console):

```
class Animal {
public:
     virtual void print() {
        cout << "Unknown animal type" << endl;
      }
protected:
     int nLegs;
};

class Fish : public Animal {
public:
     Fish(int n) { nLegs = n; }
     void print() {
          cout << "A fish has " << nLegs << " legs" << endl;
     }
};

class Bird : public Animal {
public:
     Bird(int n) { nLegs = n; }
     void print() {
          cout << "A bird has " << nLegs << " legs" << endl;
     }
};

class Mammal : public Animal {
public:
     Mammal(int n) { nLegs = n; }
     void print() {
          cout << "A mammal has " << nLegs << " legs" << endl;
     }
};

//Continued on next page
```

```
int main() {
     Animal *P[4];
     P[0] = new Fish(0);
     P[1] = new Bird(2);
     P[2] = new Mammal(4);
     P[3] = new Animal;
     for (int i = 0; i < 4; i++)
          P[i]->print();
     return 0;
}
```

What does the program print? *(25 pts)*

Explain why the program will not compile if the protected access control word is changed to private? *(5 pts)*

How can the class be turned into an abstract class? *(5 pts)*

The program prints:

```
A fish has 0 legs
A bird has 2 legs
A mammal has 4 legs
Unknown animal type
```

Even though the classes for Fish, Bird, and Mammal inherit from the class Animal, if the data member nLegs is made private in Animal, no other class can access the member directly. (The protected access type allows direct accees for derived types only).

If the declaration of the function member `print` is changed to:

```
     virtual void print()=0;
```

then it is not possible to create an object of type Animal, i.e. the class is abstract.

2

Consider the following Haskell function definitions:

```
mystery :: Integer -> Integer -> Integer
mystery x n
     | n < 0 = error "negative"
     | otherwise = ff x n


ff :: Integer -> Integer -> Integer
ff x n
   | n == 0       = 1
   | mod n 2 == 0 = ff (x*x) (quot n 2)
   | otherwise    = x * ff x (n-1)
```

[`mod` is the standard modulo function for integers, and `quot` is integer division]

Write out a derivation (a trace of function calls) of the expression `mystery 4 3` *(25 pts)*

In a sentence, what does this function do? *(5 pts)*

```
mystery 4 3
= ff 4 3
= 4 * ff 4 2
= 4 * ff 16 1
= 4 * 16 * ff 16 0
= 4 * 16 * 1
= 64
```

The function returns x^n, i.e. it is exponentiation.

3

The following is a Prolog database of facts about people and their family relationships:

```
child(min,bill).
child(john,bill).
child(sandra,bill).
child(frank,john).
child(maria,sandra).
child(praveen,sandra).
```

A first cousin relationship is defined by:

```
first_cousin(X,Y) :- child(X,A),child(Y,B),child(A,C),child(B,C).
```

Draw a goal tree for the following query until the point where a solution is found *(25 pts)*:

```
?- first_cousin(praveen,F).
```

How can the order of sub goals in the rule be changed to improve the efficiency of the search? *(10 pts)*

```
first_cousin(praveen, F)
   | X = praveen
   | Y = F
child(praveen,A),child(F,B),child(A,C),child(B,C)
   | A = sandra
child(F,B),child(sandra,C),child(B,C)____________________________
   | F = min                            | F = john               |
   | B = bill                           | B = bill               |
child(sandra,C),child(bill,C)      child(sandra,C),child(bill,C)|
   | C = bill                           | C = bill               |
child(bill,bill)                   child(bill,bill)              |
   |                                    |                        |
fail                                fail                         |
             ___________________________________________________|
             | F = sandra              |
             | B = bill                | F = frank
     child(sandra,C),child(bill,C)     | B = john
             | C = bill                |
       child(bill,bill)             child(sandra,C),child(john,C)
                                       | C = bill
                                    child(john,bill)
                                       |
                                    yes, F = frank
```

The order child(X,A),child(A,C),child(B,C),child(Y,B) will ensure that every query has one argument bound, eliminating the search with child(F,B) with neither argument bound.