CS 603 warm-up exercises **Spring 2006** **Solution**

These problems are in-class exercises which cover topics that you hopefully learned in undergraduate CS courses, and that will be used frequently in CS 603. Specific topics include recursion, functional-style programming, object-oriented programming, context-free grammars, and logic. The purposes of these exercises are (i) to let you know what topics you need to review individually, and (ii) to help the instructor assess the current overall knowledge of the students at the beginning of this course.

1. Trace the following recursive function to evaluate G(3,4). Also briefly explain what common mathematical operation the function G(m,n) computes.

```
int G(int m, int n) {
    if (n<=0) return 1;
    else return m*G(m,n-1);
}
```

81

G(m,n) computes m^n, or m raised to the power n

2. Write a C++ function sum(int array[ ], int size) that returns the sum of the elements in array[0] through array[size-1].

   a. Write the function sum using iteration.

```
int sum(int array[ ], int size) {
        int total=0;
        for (int k=0; k<size; k++) total += array[k];
        return total;
}
```

   b. Write the function sum using recursion.

```
int sum(int array[ ], int size) {
        if (size==0) return 0;
        else return array[size-1] + sum(array, size-1);
}
```

3. Write recursive functions as described below. You may use a functional language such as Lisp, Scheme, or ML. Or you may use C++, Java, or pseudo-code. You may write a helper function if you wish, but do not use iteration.

   a. Function count(x, L) returns the number of times the value x appears in the list L.

```
count(x, L) {
        if (L == empty_list) then return 0;
        else if (x == L.head) then return 1 + count(x, L.tail);
        else return count(x, L.tail);
}
```

   b. Function reverse(L) returns the reverse of list L.

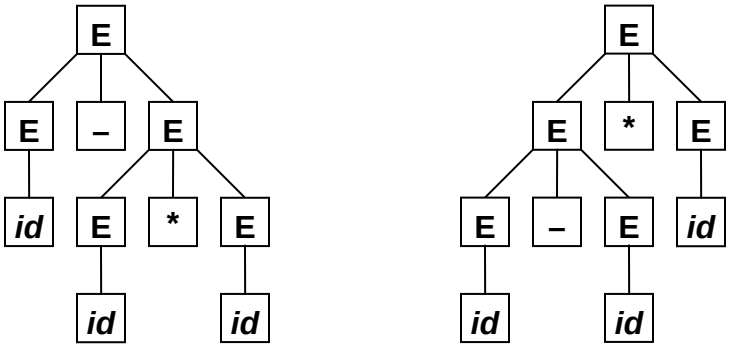
```
reverse(L) {
        return helper(L, empty_list);
}

helper(L1, L2) {
        if (L1 == empty_list) then return L2;
        else return helper(L1.tail, make_pair(L1.head, L2));
}
```

4. Consider this context-free grammar for arithmetic expressions.

   E → E + E | E – E | E * E | E / E | (E) | *id*

   a. Draw all possible parse trees for this string: *id – id * id*



   b. Write an equivalent unambiguous grammar that uses standard rules of precedence and associativity.

E → E + T | E – T | T
T → T * F | T / F | F
F → (E) | *id*

5. Rewrite each logical expression as an equivalent completely simplified expression.
   [¬ is "not", ∧ is "and", ∨ is "or", → is "implies", ∀ is "for all", ∃ is "there exists".]

   a. $(p \wedge q) \rightarrow (p \vee q)$

      true

   b. $\neg(p \vee \neg q) \wedge \neg(q \vee \neg p)$

      false

   c. $\neg(p \wedge \neg q) \wedge \neg(q \wedge \neg p)$

      $p \leftrightarrow q$

   d. $\neg\ (\forall x)\ (\neg p(x))$

      $(\exists x)\ p(x)$

   e. $\neg\ (\exists y)\ (\neg q(y))$

      $(\forall y)\ q(y)$

6. Write the output of this C++ program.

```
class A {
public:
        virtual void f( ) { cout << 1; }
        virtual void g( ) { cout << 2; }
        virtual void h( ) { cout << 3; }
        virtual void j( ) { f( ); }
        virtual void k( ) { g( ); }
        virtual void m( ) { h( ); }
        void n( ) { j( ); k( ); m( ); cout << endl; }
};
class B: public A {
public:
        virtual void f( ) { cout << 4; }
        virtual void m( ) { g( ); }
};
class C: public B {
public:
        virtual void g( ) { cout << 5; }
        virtual void j( ) { h( ); }
};
class D: public C {
public:
        virtual void h( ) { cout << 6; }
        virtual void k( ) { f( ); }
};
int main( ) {
        A *z[4] = {new A, new B, new C, new D};
        for (int x=0; x<4; x++) z[x]->n( );
}
```

Output:

1 2 3
4 2 2
3 5 5
6 4 5

7. Using C++, write a subclass Penguin of the given class Bird. The goal is to produce the specified output using good object-oriented programming style.

```
class Bird {
        string name;
public: Bird(string n): name(n) { }
        virtual bool canFly( ) { return true; }
        virtual bool canSwim( ) { return false; }
        void print( ) {
                cout << name << ": ";
                if (canFly( )) cout << "can fly, "; else cout << "cannot fly, ";
                if (canSwim( )) cout << "can swim"; else cout << "cannot swim";
                cout << endl;
        }
};

// your subclass Penguin should go here


class Penguin: public Bird {
public:
        Penguin(string n): Bird(n) { }
        virtual bool canFly( ) { return false; }
        virtual bool canSwim( ) { return true; }
};



int main( ) {
        Bird *bird[3] = { new Bird("Tweety"), new Penguin("Opus"), new Bird("Daffy") };
        for (int k=0; k<3; k++) bird[k]->print( );
}
```

Output:

```
Tweety: can fly, cannot swim
Opus: cannot fly, can swim
Daffy: can fly, cannot swim
```