CPSC 126 Alternate Midterm
February 27th, 2003

[12] 1. In this problem, you are given descriptions for two languages. In each case, write an EBNF for the language described. Remember that `a | b` means "a or b", that `[ a ]` means "a or nothing", and that `{ a }` means "zero or more copies of a".

[4] a. A *palindrome* is a word or sentence that reads the same backwards as forwards, such as esoperesteicietserepose ("Esope reste ici et se repose"). Write an EBNF for palindromes made using the letters e, s, o and p.

Solution:
```
base: "e"  |  "s"  |  "o"  |  "p"
pal: base  |  "e" pal "e"  |  "s" pal "s"  |  "o" pal "o"  |  "p" pal "p"
```

[8] b. A number is a multiple of 3 precisely when the sum of its digits is divisible by 3 (e.g., 252 and $2 + 5 + 2 = 9$ are multiples of 3). Write an EBNF for positive multiples of 3. Hints: (i) Use a category for multiples of 3, a category for numbers whose digits sum to a multiple of 3 plus 1, and another category for numbers whose digits sum to a multiple of 3 plus 2. (ii) If you know the last digit, what can you say about the remaining ones?

Solution:
```
digit30: "0"  |  "3"  |  "6"  |  "9"
digit31: "1"  |  "4"  |  "7"
digit32: "2"  |  "5"  |  "8"

mul30: digit30 [ mul30 ]  |  digit32 mul31  |  digit31 mul32
mul31: digit31 [ mul30 ]  |  digit30 mul31  |  digit32 mul32
mul32: digit32 [ mul30 ]  |  digit31 mul31  |  digit30 mul32

multipleof3: mul30
```

[6] 2. Binary Representation

[3] a. The Untel P2000 is a 14-bits CPU that uses two's complement notation to represent negative numbers. How would the Untel P2000 represent the integer -342? Hint: 342 = 256 + 64 + 16 + 4 + 2.

Solution: We would represent 342 as 00000101010110, and hence -342 would be 11111010101001 + 1 = 11111010101010.

[3] b. Consider now the following fact: If a, b are integers, then $(2^{32} - a)(2^{32} - b)$ and ab have the same remainder when divided by 2^{32}. Explain what this fact tells us about integer multiplication on 32-bit signed integers that are represented using two's complement.

Solution: It means that we can do the multiplication ab as if the values were unsigned (since ab and the product of their two's complements will have the same last 32 bits).

[8] 3. Values and Expressions

[4] a. What will the value of x be after the following expression has been evaluated? No marks will be given for an answer that is not justified. Here is a list of C++ operators listed from higher to lower precedence: (1) $+$ $-$ (2) $<$ $<=$ $>$ $>=$ (3) ?: (4) =. All of them except = associate left to right.

```
x = 4 > 6 ? 9 : 6 - 3;
```

Solution: This expression is evaluated the same was as if it was parenthesized as:

```
x = ((4 > 6) ? 9 : (6 - 3));
```

which means that it is the same as

```
x = (false ? 9 : 3);
```

and so it assigns 3 to `x`.

[4] b. Rewrite the following using only a single if-statement:

```
if( bool1 )
{
  if( bool2 )
  {
    std::cout << "Hi there." << endl;
  } else
  {
    std::cout << "Bye bye." << endl;
  }
} else
{
  std::cout << "Bye bye." << endl;
}
```

Solution:

```
if (bool1 && bool2)
{
    std::cout << "Hi there." << std::endl;
}
else
{
    std::cout << ‘‘’Bye bye." << std::endl;
}
```

[9] 4. In class, we mentioned that a parameter to a member function can either be a *value* parameter, or a *reference* parameter.

[3] a. Consider the following function declaration:

```
int someFunction(int& x, int y);
```

Explain the difference between the manner in which `x` and `y` will be treated in the body of function `someFunction`.

Solution: If we called `someFunction` as `someFunction(a, b)` then every change to `x` in the body of function `someFunction` will change `a` also. However `b` will not change, no matter how `y` is mutated.

[6] b. The following program will compile, and run without crashing. What output will it produce?

```
#include <iostream>

int jump(int skate, int& blade)
{
    skate = skate + blade;
    return skate;
}

int glide(int& skate)
{
    skate = skate + 2;
    return skate;
}

int main()
{
    int skate = 14, blade = 9;
    int x = jump(skate, skate);
    int y = glide(skate);
    std::cout << skate << x << y << std::endl;
}
```

Solution: 162816

[7] 5. Object-Oriented Design

[4] a. When should one put a member function in the `public` section of the class declaration, and why? And when should one put a member function in the `private` section of the class declaration, and why?

Solution: A member function should be in the `public` section of the class declaration if it is part of the class' interface, and should be callable by the rest of the program. It should be in the `private` section if it is used as a helper by other member functions, but should not be accessible from outside the class.

[3] b. A CPSC 126 student tried to modify the *Rational* class discussed in class by adding a member function *invert* that switches the numerator and denominator of the object. That is, the code

```
    Rational r(3,4);
    r.invert();
    r.print();
```

would display the value `4/3`. In his implementation, the student tried to call the constructor as follows:

```
void Rational::invert()
{
    Rational(denom, num);
}
```

Explain why this member function does not actually do anything (`num` and `denom` contain the numerator and denominator of the Rational object).

Solution: The `invert` member function constructs a temporary value of type `Rational`, but does not do anything with it. In particular, it does not mutate the objects' numerator and denominator.

[6] 6. The following program will compile without any errors and run without crashing. What output will it produce?

```
#include <iostream>

class Obj {
  private:
    int  m_intValue;
    char m_charValue;

  public:
    Obj();
    Obj(int v1, char v2);

    void print();
    int funny(int val1, int val2);
};

Obj::Obj() {
  m_intValue = 0;
  m_charValue = '7';
}

Obj::Obj(int v1, char v2) {
  m_intValue = v1;
  m_charValue = v2;
}

int Obj::funny(int val1, int val2) {
   m_intValue = val1;
   m_charValue = m_charValue + val2;

   return val1 * val2;
}

void Obj::print() {
   std::cout << m_intValue << m_charValue;
```

```
}

int main() {
   Obj obj1;
   Obj obj2( 7, 'g' );
```

```
   int num = obj2.funny(9, 3);

   obj1.print();
   obj2.print();
   std::cout << num << std::endl;
}
```

Solution: 079j27

[17] 7. A software developer working for a video games company has come up with the following class to describe general attributes (properties) of objects on the screen during the game:

```
//
// Note: colors are represented by integers; Black is 1, Red is 2, etc.
//
class CommonInfo {
    public:

        CommonInfo(); // Position = (0,0)
        CommonInfo(int origcolor, int xorig, int yorig);

        int x();      // Return the x-coordinate
        int y();      // Return the y-coordinate
        int color();  // Returns the object's color.

    // The rest of the class declaration is omitted.
};
```

The `origcolor` parameter to the second constructor indicates the color of the object. The `xorig` and `yorig` parameters are the position of the object's reference point.

You have been asked by the company to write a class `hole` that is used to store black holes, red holes and green holes (three types of objects that occur frequently is space-adventure games). They have provided the following class interface:

```
class hole {
    public:
        // Set up a black hole with radius 10.0 and center (100,100)
        hole();

        // Another constructor
        hole(double radius, CommonInfo cinfo);
        // Accessors
        double radius() const; // returns the hole's radius.
        int color() const;     // returns the hole's color.

        // The following returns true if the two holes overlap.
        bool overlap(hole otherhole) const;

    private:
        // To be supplied by you.
};
```

[3] a. Complete the private section of the `hole` class by supplying the necessary member variables. Note: you **must** use the `CommonInfo` class defined at the beginning of this problem.

Solution:

```
    double rad;
    CommonInfo common;
```

[14] b. Provide the code for the two constructors, and for the member functions `color`, `radius` and `overlap`. Use the back of this page if you run out of space. Note that two holes overlap if the distance between their centers (positions) is smaller than the sum of their radii. Recall also that the distance between two points (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, and that C++ provides a builtin `sqrt` function that computes the square root of a number.

Solution:

```
    hole::hole()
    {
        common = CommonInfo(0, 100, 100);
        rad = 10.0;
    }

    hole::hole(double radius, CommonInfo cinfo)
    {
        common = cinfo;
        rad = radius;
    }

    int hole::radius() const
    {
        return rad;
    }

    int hole::color() const
```

```
{
    return common.color();
}

bool overlap (hole otherhole) const
{
    int dx = commoninfo.x() - otherhole.commoninfo.x();
    int dy = commoninfo.y() - otherhole.commoninfo.y();

    double dist = sqrt(dx * dx + dy * dy);
    return dist <= rad + otherhole.rad;
}
```