CS106X Handout #08A
Spring 99-00 April 3, 2000

Section Solutions 1: C basics

Problem 1: Warm-up questions on style

a) Constants should be used in place of numbers to make a program more readable and easier to change later. If you can think of a name to describe a value's use (e.g. `MaxStudents`, `MinRadius`, `WhitePaint`), then you should use a constant. Any value referenced in more than one place should be made into a named constant so that it takes just one edit to change the value. Pretty much the only time you should use an explicit number is when it is functioning only as a number, not as a measurement or particular setting. An example of this is dividing by 2. All you're doing is dividing by 2—there is no name for the number (except `Two` which adds nothing to readability).

b) A good variable or function name makes it clear what that variable is or function does (e.g. `numScores`, `userName`, `width`, `DrawDiamond()`, `Average()`, `GetInteger()`). In general, these names are descriptive, readable, and use abbreviation sparingly.

c) If lines of code are repeated, they generally warrant being moved to a function that can be called from other places in your program. If a code section can be thought of as a separate , distinct part of the problem with a clearly defined interface (i.e. something you can easily turn into parameters), it is a candidate for its own function. Often you move sections of code into separate functions to decrease the length and complexity of the calling function and aid readability. Refer to the handout on decomposition for more suggestions.

d) It depends. One-line functions aren't bad in and of themselves, but should be evaluated for their effect on readability. A good example of a one-line function that increases readability is the `CelsiusToFarenheit` function we wrote in class. It's not immediately obvious what that one line of code is doing (i.e. `9.0 / 5.0 * Celsius + 32`), so making it a function with a descriptive name adds to readability. An example of a one-line function that adds nothing to readability (and actually decreases readability) is replacing `sum = x+y` with `sum = AddTwoNums(x, y)`.

Problem 2: Arithmetic expressions

a) `(1 - (2 + 8 / 6) * 4 % 3 * 1) / 2 = 0`

b) `num4 = 10.0`

To see the rules of arithmetic expressions and precedence, please see pp. 20-27 in your text book. Also, make sure you understand integer division and truncation.

Problem 3: Boolean expressions

a) The expression will evaluate to FALSE, regardless of the values of the variables.

b) This will always be TRUE.

c) As seen on page 111 of your text, the line should be written as follows:

```
        ((y % 4 == 0) && (y % 100 != 0)) || (y % 400 == 0)
```

There are precedence rules that govern the order boolean expressions (sometimes called predicates) are evaluated, but these rules are easily forgotten, so it's best not to rely on them and explicitly place the parentheses in such expressions.

d) The second expression evaluates to TRUE.

The important thing to notice about these expressions is that in both cases, it never even calculates the second half of the expression. In the first case, when the first part is found FALSE, the entire esult is known because anything conjuncted with FALSE is FALSE. In the second case, the first expression is TRUE, and anything disjuncted with TRUE will be TRUE, so it doesn't need to calculate any further. Hopefully you didn't bother to do these extra calculations either! This feature is referred to as short-circuit evaluation, and is a feature of C's logical operators that can prove useful in programs to come.

Problem 4 : Divisors

```
main()
{
   int number, i;

   printf("Please enter a number: ");
   number = GetInteger();

   for (i = 1; i <= number;i ++) /* note unusual loop bounds */
      if ((number % i)==0)
          printf("%d * %d = %d\n", i, number/i, number);
}
```

Problem 5 : XOR

```
bool XOR (bool x, bool y) {
   return ((x || y) && (!(x && y)));
}
```

Note logical XOR is not the same thing as bitwise xor (the ^ operator)!

Problem 6 : Graphics

```
   void DrawSwirl(double cx, double cy, int numSpokes, double radius,
                   bool clockwise)
    {
       int i;

       for (i = 0; i < numSpokes; i++) {
           MovePen(cx,cy);
           DrawArc(radius/2.0, i*(360.0/numSpokes),
                   clockwise? -180 :180);
       }
   }
```

Problem 7: Random graphics

```
/* Stacey Doerr              1/23/94
 * File: donut.c
 * ----------------------------
 * This programs draws a series of donuts on the screen until the user
 * clicks the mouse button.  The donuts are randomly place and of
 * random "color" (greyscale), but all fit completely in window.
 */

#include "extgraph.h"
#include "random.h"

#define DONUT_IN_OUT_RATIO          .5
#define FULL_CIRCLE                 360
#define WHITE_FILL                  0
#define MIN_DENSITY                 .1
#define MAX_DENSITY                 1.0
#define MIN_RADIUS                  .5
#define MAX_RADIUS                  1.0

void DrawRandomDonut(double scrWidth, double scrHeight);
void DrawDonut(double cx, double cy, double radius, double density);
void DrawFilledCircle(double cx, double cy, double radius, double
density);
void DrawCircle(double cx, double cy, double radius);


main()
{
      double scrWidth, scrHeight;

      InitGraphics();
      Randomize();
      scrWidth= GetWindowWidth();
      scrHeight= GetWindowHeight();
      while (!MouseButtonIsDown()) {
            DrawRandomDonut(scrWidth, scrHeight);
            UpdateDisplay();  // To see our new creation
      }
}

/*
 * Function: DrawRandomDonut
 * ------------------------
 * This function draws one donut of random density centered on a random
 * pt on the screen. It takes the screen width and the screen height as
 * as parameters in order to make sure the donut fits on the screen.
 */
void DrawRandomDonut(double scrWidth, double scrHeight)
{
      double cx, cy, radius, density;

      radius = RandomReal(MIN_RADIUS, MAX_RADIUS);
      cx = RandomReal(radius, scrWidth - radius);
      cy = RandomReal(radius, scrHeight - radius);
      density = RandomReal(MIN_DENSITY, MAX_DENSITY);
      DrawDonut(cx, cy, radius, density);
}


/*
 * Function: DrawDonut
 * ------------------
 * Draws a donut by drawing an outer circle of density
 * outerDensity and a white inner circle.
 * The donut is centered at cx, cy and the outer circle has
 * the radius specified.
 */
void DrawDonut(double cx, double cy, double radius, double
      outerDensity)
{
      DrawFilledCircle(cx, cy, radius, outerDensity);
      DrawFilledCircle(cx, cy, (radius * DONUT_IN_OUT_RATIO),
      WHITE_FILL);
}



/*
 * Function: DrawFilledCircle
 * -------------------------
 * This function draws one circle filled in with the
 * density passed in.  The circle is centered at cx,cy and has
 * the radius specified.
 */
void DrawFilledCircle(double cx, double cy, double radius, double
      density)
{
      StartFilledRegion(density);
      DrawCircle(cx, cy, radius);
      EndFilledRegion();
}



/*
 * Function: DrawCircle
 * --------------------
 * DrawCircle draws a circle centered at the point cx, cy and
 * with the radius passed in. Since DrawArc actually starts at
 * the point the pen is at, I move the pen to be at the right
 * edge (0 degrees) of the circle I want to draw before I
 * call DrawArc and then tell DrawArc to start drawing as if
 * the point the pen is at is the 0 degree point of
 * the circle.
 */
void DrawCircle(double cx, double cy, double radius)
{
      MovePen(cx + radius, cy);
      DrawArc(radius, 0, FULL_CIRCLE);
}
```