Assignment 3
Functions, Graphics, and Decomposition

Due: Friday, October 15

"In] making a quilt, you have to choose your combination carefully. The right choices will enhance your quilt. The wrong choices will dull the colors, hide their original beauty. There are no rules you can follow. You have to go by instinct and you have to be brave.

— from the film adaptation of Whitney Otto’s *How to Make an American Quilt*, 1995

This assignment is a simplified version of one that Julie Zelenski developed for CS106X several years ago. The problem is to write a program that draws a picture of a particular type of quilt called a **sampler quilt**, which is composed of several different block types that illustrate a variety of quilting styles. For this assignment, your task is to use the graphics library to generate a drawing of the following sampler quilt design:

The quilt consists of five patterns, illustrated by the first five squares on the bottom row:

- A “log cabin” block consisting of a nested pattern of rectangles
- A flower-petal design
- A bulls-eye pattern
- A hybrid pattern combining elements of the log cabin block and the flower petal
- A do-it-yourself design indicated by a question mark in this diagram that you are free to replace with any pattern of your own choosing
Each of these designs is discussed in more detail later in this handout.

**Arrangement of the quilt blocks**

If you look at the quilt as a whole, one of the first things to notice is that the different block types are arranged in the quilt so that they form a regular pattern. Each successive row in the quilt has the same blocks in the same cyclic order. The only difference is that the blocks in each row are shifted one position to the left relative to the blocks in the row underneath them. If you number the patterns 0, 1, 2, 3, and 4, the blocks in the quilt are arranged in the following configuration:

```
 4 0 1 2 3 4 0
 3 4 0 1 2 3 4
 2 3 4 0 1 2 3
 1 2 3 4 0 1 2
 0 1 2 3 4 0 1
```

The choice to number the patterns beginning at 0 is not arbitrary. Programmers—and C programmers in particular—often number things beginning at 0 because it often makes it easier to use the remainder operator (\%) to represent a cyclical pattern. In the `calendar.c` program, for example, the days of the week were numbered starting from 0 because doing so made it possible to compute the next day of the week using the expression

```
(weekday + 1) \% 7
```

A similar cyclical pattern occurs in the quilt, and it will simplify your program considerably if you figure out how you can use the remainder operator to your advantage.

**Constant definitions**

Like most graphics programs, the exact appearance of the quilt is controlled by a set of constants that are defined at the beginning of the program. The following constants apply to the quilt as a whole:

```
#define QuiltRows     5
#define QuiltColumns  7
#define NBlockTypes   5
#define BlockSize     0.75
```

The constants `QuiltRows` and `QuiltColumns` indicate the dimensions of the quilt vertically and horizontally, measured in terms of the number of blocks; the sample quilt shown in the diagram, for example, is five rows high and seven columns wide. The `NBlockTypes` constant is used to record the number of different block types that appear in the quilt. The `BlockSize` constant is a floating-point value that indicates the size of each block, measured in inches. There are also constants that pertain to the individual block types, which are described later in the handout.

The important thing to remember about defining constants is that the point is to make the behavior of the program easy to change. If you change `BlockSize`, for example, to 1.0, the blocks should get larger, but the basic pattern of the quilt should remain the same. In
fact, you should test your program with different values of these constants to make sure that everything continues to work.
The individual block types

The sampler quilt in this assignment consists of the five block types, each of which is described in detail in one of the following sections.

Block 0: Log cabin

This block is a representation of an authentic quilt pattern that is widely used in traditional quilting. It is composed of a series of frames, each of which is nested inside the next larger one. Each frame is in turn composed of four rectangular boxes laid out to form a square. The outer frame, for example, looks like this:

The next frame fills the space inside the outer frame, as follows:

This same pattern continues for a specified number of frames, giving rise to the following complete picture:

As illustrated in the preceding diagram, the constants that control the log cabin block are

```
#define NFrames     4
#define FrameWidth  (BlockSize / (2 * NFrames + 1))
```

The constant \texttt{NFrames} indicates how many frames appear in the pattern. In this quilt, the log cabin block consists of four frames that surround a small square in the center. The \texttt{FrameWidth} constant represents the width of the rectangles making up the frame and is defined in terms of other constants that have previously appeared. This technique of defining a constant as an expression is quite common, but it is good practice to enclose the entire expression in parentheses to ensure that it acts as a single unit.

At first glance, the definition of \texttt{FrameWidth} may seem cryptic. The idea is to determine how wide the rectangular strips need to be so that each of the frames and the interior square will all have the same width. If you look at the complete pattern
you can see that each frame accounts for twice the frame width in either dimension because there is a frame on each side. Since the square in the middle is supposed to have the same width, the denominator in the fraction must be twice the number of frames, plus one for the square.

Block 1: Flower petals

This block is a stylized representation of a four-petal flower and is included in the assignment mostly to give you practice drawing arcs. Each petal is composed of two quarter circles superimposed on top of each other as shown in the following enlarged picture of one petal:

\[ + \quad = \]

The only constant required to draw the flower petal is the radius of the arc, which is defined as follows:

```c
#define PetalRadius    (0.3 * BlockSize)
```

Block 2: Bulls-eye

This block is composed of a series of concentric circles centered in the block. The smallest circle has radius `SmallRadius` and the radius of each surrounding circle increases by `DeltaRadius` until `NCircles` have been drawn. For the assignment, the values of these constants are

```c
#define NCircles     4
#define SmallRadius  (0.1 * BlockSize)
#define DeltaRadius  (0.1 * BlockSize)
```

You should not worry if some of the circles appear to be slightly off-center on the screen. The screen is composed of a matrix of individual dots called pixels that constrain the precision of the image. If the radius of each circle is not an exact multiple of the pixel size, the size of the circle must be rounded to something that can actually be drawn on the screen, which results in a certain degree of inaccuracy.

Block 3: Hybrid

This block is a combination of elements that appear in the log cabin (the outer frame) and flower petal designs. The point of including this block is to emphasize the importance of software reuse. If you've designed your program well, you will have to write very little new code to create this design.

Block 4: User-defined

The question mark does not actually appear in the final assignment. This block is left as an opportunity for you to demonstrate your own creativity. You have to put some type of design in this block to get full credit (i.e., a √+) on the assignment, but it need not be at all fancy. If you’re trying for extra credit (i.e., the + and ++ scores), you’ll have to add even more, as outlined in the section entitled “Extending the assignment” later in this handout.
Getting started
To make this program a little easier to start, we have placed a “starter” project in the Assignment 3 folder on the Catacombs. The quilt.c source file in the starter project folder contains all the constant definitions listed in this handout so that you don’t need to retype them. In addition to the starter project, you may also want to make use of code from the various examples in the textbook. All sample programs from the text are available to you in the Textbook Examples folder. If you need some function that is defined in the text, just cut and paste it from the original source file into your program, but be sure to cite your source in the program you submit.

Strategy and tactics
With this assignment, you have your first opportunity to write a program large enough that the issues of top-down design and bottom-up implementation really start to matter. To succeed in managing the complexity of the assignment, you will have to use both. If you view it as a whole, this program is actually quite complex. What makes it possible is that the individual pieces of the program are not so difficult. By breaking the program down into the right individual components and then testing these individual parts in isolation, you should be able to master that complexity.

The following hints should help you as you proceed:

• **Start early.** This assignment is due in one week and can easily be completed in that time. If, however, you choose to start it next Sunday night, you will almost certainly end up using some of your late days.

• **Think carefully about the overall program design before you write any code.** This program decomposes very nicely into pieces; one of your principal responsibilities as a programmer is to determine what the appropriate pieces are. Before you start, it is a good idea to sketch out the decomposition: decide what functions you should define, what arguments those functions need, and how the various pieces fit together.

• **Test your program in small pieces.** Even though it is usually pays to design your programs from the top down, it is very hard to implement them using that approach. Remember that the goal of decomposition is to break things down into separable, easily managed pieces. When you test your program, it is best to build one piece at a time and then to get that piece working before you move on to the next. When you write this program, you should start with one of the blocks—the bulls-eye is probably the easiest—and write a program that just displays a single bulls-eye block on the screen. Once you get that working, write a program that displays one of the other block types. Continue this process until you can display each of the four required blocks individually. At this point, you can then write the code necessary to arrange these blocks into the quilt pattern.

• **Write your program so that others can understand what you’ve done.** In this and all other assignments for this course, your job is not simply to get your programs working. Although correctness is undeniably important, we also expect you to write well-crafted programs that you, your section leader, and anyone else you show them to will be able to understand. In particular, you should ensure that your decomposition produces functions whose effect is easy to describe. Moreover, you should use the comments associated with the function to set that description down in clear, easy-to-follow English. If you find that your functions are hard to describe in comments, maybe you need to decompose the problem in a different way.
Extending the assignment

The grading structure and the individual assignments in CS106 are designed to encourage initiative and creativity. In creating the quilt, part of the assignment is to construct a block of your own design. But you can go much farther than that. There are many ways in which you can embellish the program and make it far more interesting, both in terms of the graphical result it generates and the intellectual challenge represented by the programming task itself.

There are some extensions you could undertake using only the simple graphics library described in Chapter 7 of the text. To do anything really fun, however, you will probably need to use a set of extended functions defined in the `extgraph.h` interface, which is also built into Thetis. The extended graphics interface provides you with a variety of additional tools that add considerable power to the interface, including the following:

- Color
- Filled shapes with shading
- Text in various fonts and sizes
- Simple animation supported by selectively erasing and redrawing parts of the image
- Mouse input

None of the features in the extended graphics library are described anywhere in the text. If you want to use them, you will need to read the interface, which is available on the Catacombs server in the `headers` folder inside the Thetis package and which is reproduced in full in the handout describing the graphics contest (Handout #20). Reading interfaces is a critical skill for advanced work, and extending this assignment gives you an excellent chance to practice this important skill.

When you extend the program, you should make sure that it incorporates the original design required by the assignment, even though you are free to embellish it. For example, you can color the petals or add new structure to the flower, but you must make it clear that you could have done the assigned task just as easily. In general, it is best not to try any extensions until you get the basic assignment working.