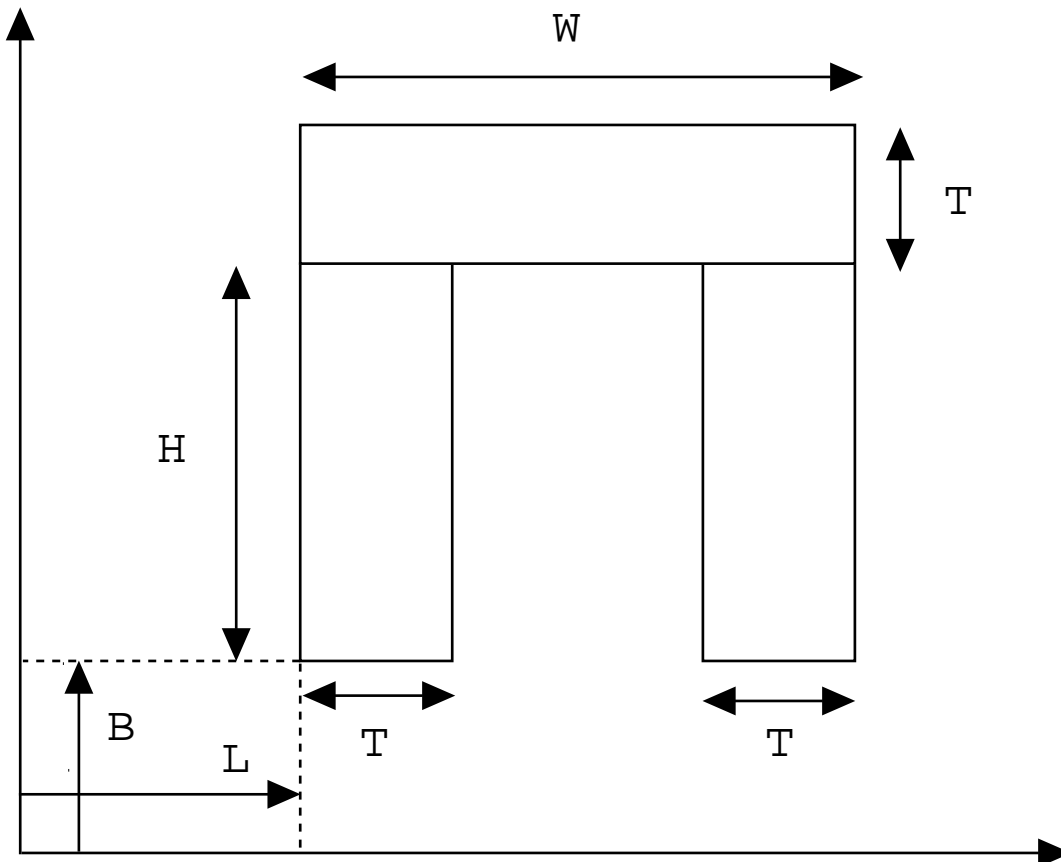


CS 101 Computer Science I (Spring 2001)
Lab 6: February 28 - March 9, 2001

In this lab you will develop your knowledge of data abstraction in a simple computer graphics application. We will provide you with a file called “`cs101-graphics.scm`”. This file contains a collection of Scheme procedures for drawing pictures of triangles, rectangles, houses and towers. You will extend this program so that it can draw a picture of an *arch*, similar to the one shown below. The lab will illustrate the idea that an abstract data type can be implemented in more than one way. For this purpose, you will implement the arch data type twice, using a different representation each time. We will provide you with the constructor functions for creating arch data objects in each of the two representations. You will find these constructors in the file `cs101-graphics.scm`. Your job is to write the selector functions that complete the implementation of the arch data type in each of the two representations. Once you do so, you will be able to use a procedure called “`draw-object`” to draw arches on the screen of your computer.



1. Start Scheme and open the file `cs101-graphics.scm`. Click on “Execute” to evaluate all of the definitions in this file. Evaluate the expression `(initialize-graphics)`. You should see a new, small window pop up on your screen. Adjust the size of your other windows so that they do not overlap the new graphics window. Now evaluate the expression `(finalize-graphics)` to make the graphics window go away. Next open the graphics window again by evaluating the expression `(initialize-graphics)`. Then evaluate the expression `(draw-object *house*)` and observe that a drawing of a house appears in the graphics window. Now evaluate the expression `(finalize-graphics)` to make the graphics window go away. Finally open the graphics window again by evaluating the expression `(initialize-graphics)`. Then evaluate the expression `(draw-object *tower*)` and observe that a drawing of a tower appears in the graphics window. Now evaluate the expression `(finalize-graphics)` to make the graphics window go away.
2. In the first of our two implementations, an arch is represented as a list of four items: the symbol `arch`, which indicates the type of the data; two rectangles representing the left and right supports of

the arch, and a third rectangle representing the top beam of the arch. In this implementation, the `build-arch` and `make-arch` functions are implemented as shown below. Write the selector functions that complete this implementation of the arch data type. The functions you must implement are listed at the end of this document. Test your implementation of the arch data type by evaluating the two expressions for building and drawing an arch that appear at the end of the file `cs101-graphics.scm`.

```
(define build-arch
  (lambda (left bottom width height thickness)
    (make-arch (make-rectangle left bottom thickness height)
              (make-rectangle (+ left (- width thickness)) bottom thickness height)
              (make-rectangle left (+ bottom height) width thickness))))

(define make-arch (lambda (left-support right-support top-beam)
  (list 'arch left-support right-support top-beam)))
```

3. In the second of our two implementations, an arch is represented as a list with the symbol `arch`, indicating the type of the data, and five numbers: the horizontal coordinate of the left side; the vertical coordinate of the bottom; the height of the supports; the width of the top beam; and the thickness of the supports and top beam. In this implementation, the `build-arch` and `make-arch` functions are implemented as shown below. Write the selector functions that complete this implementation of the arch data type. The functions you must implement are listed at the end of this document. Test your implementation of the arch data type by evaluating the two expressions for building and drawing an arch that appear at the end of the file `cs101-graphics.scm`.

```
(define build-arch (lambda (left bottom width height thickness)
  (make-arch left bottom width height thickness)))

(define make-arch (lambda (left bottom width height thickness)
  (list 'arch left bottom width height thickness)))
```

Arch Selector Functions

- `arch-left-support` takes an arch as input and returns a rectangle representing the left support of the arch.
- `arch-right-support` takes an arch as input and returns a rectangle representing the right support of the arch.
- `arch-top-beam` takes an arch as input and returns a rectangle representing the top beam of the arch.
- `arch-left` takes an arch as input and returns an integer representing the vertical coordinate of the left side of the arch.
- `arch-bottom` takes an arch as input and returns an integer representing the vertical coordinate of the bottom of the arch.
- `arch-width` takes an arch as input and returns an integer representing the width of the arch.
- `arch-height` takes an arch as input and returns an integer representing the height of the arch.
- `arch-thickness` takes an arch as input and returns an integer representing the thickness of the supports and top beam of the arch.
- `arch?` takes any input and returns `#t` if the input is an arch and returns `#f` otherwise.