UNIVERSITY OF TEXAS AT AUSTIN
Dept. of Electrical and Computer Engineering

EE382C-9 Embedded Software Systems
Problem Set #1:
Languages, Digital Signal Processors, Object-Oriented Software, Filters

Date assigned: February 17, 2004
Date due: February 25, 2004

Late homework will not be accepted.

Reading: *Languages for Embedded Systems*, ch. 1 and ch. 5–9

You may use any computer program to help you solve these problems, check answers, etc.

Please submit your homework solutions to the grader (Mr. Ming Ding) by e-mail at ming@ece.utexas.edu. You must submit the source code and makefile for problem 1.4 by e-mail to Mr. Ding. Mr. Ding's work number is 512-232-2769.

Problem 1.1 Languages for Embedded Systems. 15 points.

(a) What is nondeterminism?

(b) When does nondeterminism arise? Give one example. On the course Web page for the second lecture, I list the following two examples of non-determinism given by Prof. Edward Lee at UC Berkeley (so you may not use them as answers for this question):

i. User interfaces require nondeterminism. For example, mouse clicks and keyboard presses are events that can be arbitrarily interleaved when sent to a handler.

ii. Real-time applications also often require nondeterminism. Interrupts [esp. prioritized interrupts] are a usual way of managing this.

(c) What are the advantages and disadvantages of supporting nondeterminism in a computer language?

Problem 1.2 Digital Signal Processors. 15 points.

Pick a VLIW digital signal processor and a desktop general-purpose processor, and list five differences in their instruction set architectures.

Problem 1.3 Evaluation of Object-Oriented Programming. 10 points.

Name five advantages and five disadvantages of object-oriented programming. Your answers should not be based on a particular object-oriented language, such as C++ or Java, but instead should be more general.

Problem 1.4 Digital Finite Impulse Response Filter C++. 30 points.

Digital finite impulse response (FIR) filters are widely used in audio, image, video, and communication systems. Given an input signal, $x[k]$, where k is the discrete-time sample index, the output signal, $y[k]$, of an FIR filter can be written as a discrete-time convolution as follows:

$$y[k] = \sum_{n=0}^{N-1} a[n]\, x[k-n]$$

The coefficients $\{a[0], a[1], \ldots, a[N-1]\}$ are typically designed off-line, e.g. using the `filtdemo` command in Matlab, to meet a certain set of specifications, e.g. on magnitude frequency response. We can store the coefficients (a.k.a. taps) in a linear array of N words.

In the summation, the terms $\{x[k], x[k-1], \ldots, x[k-(N-1)]\}$ represent the current input and the $N-1$ previous inputs. By storing these input terms in a circular buffer, we can efficiently update the circular buffer as a new input data arrives to be processed (filtered).

An FIR filter has memory because it "remembers" $N-1$ previous input values.

In this problem, you will implement an FIR filter using object-oriented programming in C++. All data members in the C++ classes that you write should be private. The only visible access to the data members from the outside should therefore be through methods. All numeric data are integers. **The C++ source code needs to compile using version 2.95.2 of the GNU C++ compiler (g++) installed on the Sun Solaris 2.8 machines sunfire1 and sunfire2 in the ECE Learning Resource Center.** No Microsoft Foundation Classes may be used.

(a) Develop a C++ class to implement a circular buffer. The constructor should take an integer argument that is the length of circular buffer. The values of the circular buffer should be initialized to zero. Define a method `addValue` to add a new value to the circular buffer. Overwrite the oldest data value with the new value. Define a method `readValue` to read an element from the circular buffer given an index into the circular buffer. This method should support modulo addressing. Define the appropriate destructor.

(b) Develop a C++ class to implement an FIR filter. The constructor should take two arguments: an integer number of the number of coefficients, and an array of coefficient values. The coefficient values should be copied. A circular buffer of zero values of length equal to the number of coefficients should be allocated. Define a method `outputValue` to compute a new output value given a new input value. Define the appropriate destructor.

(c) Write a main program to filter a short signal. Use the filter coefficients $\{-1, -2, -3, -2, -1\}$. For the input signal, initially use $\{1, 2, 3, 4\}$. Stream the data into the filter one sample at a time. (This "streaming" is what a scheduler would do if the FIR filter were for example in a block diagram representation such as synchronous dataflow.) When the input stream is finished, stop the program.

(d) Update your main program to filter a long signal. Use the same filter coefficients $\{-1, -2, -3, -2, -1\}$. Generate a long array of 100,000 elements that holds the values 1, 2, ... Time the execution. Get the start time with a resolution of at least 1 msec. When the input stream is finished, get the stop time with a resolution of at least 1 msec. Do not include the time to initialize this array in the execution time. Report the execution time. If you are on a multiuser system, then also report the load (in Unix, you could use the `uptime` command).

Problem 1.5 Digital Finite Impulse Response Filter in Java. 30 points.

Repeat problem 4 in Java. The Java code must compile under JDK 1.3, which is installed on the Sun Solaris 2.6 machines in the Learning Resource Center. As much as possible, try to make the Java version a direct port of your C++ code in Java. You will not need to define a finalize method (which performs many of the tasks that a destructor in C++ would perform) for the new classes. For part (d), be sure to run the C++ and Java versions on the same machine. In addition, compare the execution times of the C++ and Java implementations found in part (d) after normalizing for the load on the machine at the time of execution.