Embedded Software Design Techniques

Submitting Exercise problems

- **Send to: isshiki@ict.e.titech.ac.jp, fengpll@ku.ac.th**
- **Cc: to yourself → so that, in case your email does not arrive here, you can resend it**
- **Subject: ESDT exercise #xx**
- In the text body, describe your name, student ID, and a brief description of the contents of the attached file.
- In the attached file, put your source codes and result outputs
- To dump the printf output to a file, use fprintf(fp, "…", …);

```
FILE * fp = fopen("filename", "w");
fprintf(fp, "%d = %d¥n", data0, data1); // replace printf
fclose(fp);
```

Lecture Outline

- Embedded software overview
  - What are “embedded systems” and “embedded software”?
- C programming 1: C language overview
  - Function, declaration, statement, expression
  - Data types, data structure, pointers and pointer dereferences
- C programming 2: algorithm complexity, program execution model
  - Bubble sort vs quick sort
  - Stack memory and program execution
- C programming 3: programming techniques in image processing
  - Dynamic memory allocation, image array implementation
  - Greyscaling, filtering, binarization, color quantization, dithering
- C programming 4: programming complex applications
  - Program development steps (ex. Huffman coding)
  - Binary tree construction, tree traversal
  - Bitstream handling
- Real time operating systems and application development
  - RTOS services, kernels
  - Context switching, task scheduling
  - Multi-task programming model

Setting Up the C Programming Environment

- Windows environment
  - MinGW (http://www.mingw.org/) : GNU compilers
  - To install:
    - Go to: http://www.mingw.org/node/24 (HOWTO Install the MinGW (GCC) Compiler Suite)
    - Follow the instructions “Using the (possibly "Proposed"/"Candidate") Installer ” and execute the installer
    - By default, MinGW package will be installed in **`C:\MinGW`** (you should not redirect the installation path which has “space” characters in the path name (like “My Documents”)
  - To use GCC compiler:
    - Create your working directory (also should not have “space” characters in the path name)
    - Start the Windows command prompt window (“cmd.exe”) from the Windows Start Menu → Accessories → Command Prompt, and “cd” to your working directory
    - Set the path to **`C:\MinGW\bin`** directory by typing

      ```
      set path=C:\MinGW\bin;%path%
      ```
    - If you have something to compile (like “test.c”), simply type

      ```
      gcc test.c
      ```

      This will create the executable file “a.exe”
  - Documents are in **`C:\MinGW\info\gcc.info`** (this is a very long text file which explains all options for gcc)

Setting Up the C Programming Environment

- Windows environment
  - Microsoft Visual Studio: contains full package of programming environment with nice user-interface and relatively easy GUI (graphics user-interface) development tools
- Linux, Unix
  - GCC is there...
- MAC
  - Xcode…

Exercise 1

1. First, write those programs shown in this slides and get used to GCC and CMD console (and DOS)
   - Use "Notepad", "Wordpad" or any other text editor that you have to write programs
2. Write a program that prints all prime numbers up to N (value N should be given from the command argument)
   - Recall that `(a % b)` means "remainder of `a / b`"
3. Write a program that prints the "prime factored form" of N, such as:
   - 10 = 2 * 5
   - 81 = 3 * 3 * 3 * 3
   - 100001 = 11 * 9091
4. Write a program that multiplies two signed integers WITHOUT using "`*`" operator
5. Write a program that divides two signed integers WITHOUT using "`/`" operator
6. Write your own "`atoi`" function

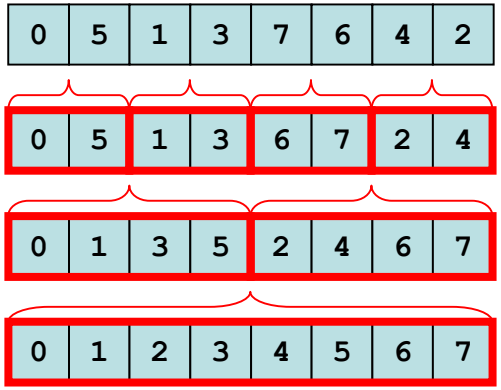
7. Write a program that prints a sequence of numbers (given from the command argument), sort these numbers, and print the sorted number sequence
   - You can assume some "maximum" length of number sequence that is fixed inside your program.
   - But you should give a warning message that "there are too many numbers" in the command argument.

Exercise 2 (sorting)

1. Write a program that sorts words in dictionary order (words should be given from the command arguments)
   - “this” “is” “a” “pen” → “a” “is” “pen” “this”
2. Write a program that prints the “median” value
   - A median is the element which is in the middle of the sorted list, so you can compute the median by sorting the list and printing the middle element
   - Think about how you can get the median value without sorting the entire list by modifying the quick_sort program

Exercise 2 (sorting)

3. Write a sorting program using "merge sort" algorithm on $M = 2^N$ array elements:
   - For k = 0, 1, ..., N – 1: Do sorting on every 2^{k+1} adjacent elements in the array
     - At k = 0, sorting each 2 adjacent elements requires one comparison and a swap (if order is reversed)
     - At k > 0, on each 2^{k+1} adjacent elements, the first 2^k elements and the second 2^k elements are already sorted by the previous iteration



4. Modify your above program so that it can also work on the array size which is not a power of 2

Exercise 3 (image)

1. Generate a “negative” greyscale image
   - A “negative” image is an image where white and black are reversed
2. Flip the image upside down
3. Rotate the image 90 degrees
4. Shrink the image width to one half
5. Enlarge the image width by 1.5 times
6. In the color quantization example, each colormap value was simply calculated as the average of min/max range values. Consider improving the quantized image quality by setting the colormap value as the average RGB values of all pixels that map to that colormap
7. We can improve the quantized image quality further by adaptively setting each quantization steps and levels. Most popular method is called “median-cut” algorithm. Try programming this method. To learn how the median-cut algorithm work, refer to http://micro.magnet.fsu.edu/primer/java/digitalimaging/processing/colorreduction/index.html

Exercise 4 (Compression)

1. Try compressing different kinds of files and observe the compression rate (see what happens when you compress an already compressed file)
2. Write a program that combines color quantization and Huffman coding
   - Input : 24-bit RGB image
   - Color quantization : 8-bit/pixel (colormap_size <= 256)
   - Huffman coding
     - Write compressed file (*.hmc)
     - Read compressed file and write the decompressed file as "result_huffman.bmp" → confirm that you can open this with the image viewer

   → Try different quantization levels, especially observe what happens to the compression rate when you decrease colormap_size