CSCI 1730 Breakout / Lab 01

Save the Bits... Get Bit-Shifty!

Last Updated: January 16, 2017

Problem / Exercise

You are NOT allowed to use the computers during this time.

Suppose you have an `unsigned int` variable (4 bytes; 32 bits). If a user intends to store a single unsigned value in that variable, then the minimum and maximum that value can be are 0 and $2^{32} - 1 = 4294967295$, respectively. Now suppose a user wants to store k unsigned integer values in that variable where 32 is divisible by k. Here are some questions to consider:

- Assuming each value uses the same number of bits, how many bits are available for each value?
- Assuming each value has the same range, what is the minimum and maximum that each value can take on?
- How can you assign a particular value without disrupting the other values?
- How can you extract a particular value?

Hints

- Do not use arrays or strings to solve this problem. You don't need them.
- You can use extra variables, loops, and operators as needed.
- You may wish to explore using the bitwise operators (including left shift and right shift).

Textbook & Other Resources

Students *may* find the following sections of the textbook useful for this assignment:

- DEITEL 6.13 References and Reference Parameters (pp. 242–245)
- DEITEL 22.5 Bitwise Operators (pp. 905–914)
- DEITEL 22.6 Bit Fields (pp. 914–917)

Students *may* also find the following resources useful for this assignment:

- Reference Declaration – `http://en.cppreference.com/w/cpp/language/reference`
- `std::bitset` – `http://en.cppreference.com/w/cpp/utility/bitset`
- Bitwise Operators – `http://www.learncpp.com/cpp-tutorial/38-bitwise-operators/`

General Examples

Assume you have `unsigned int x`. In the examples below, spaces are provided for readability only. For any valid value of k each of the 0 through $k-1$ indices denotes one of the k groups of bits within the variable `x`. Although the choice of having the indices start from the right is entirely arbitrary, it does make some of the math a little easier to work out.

- $k = 1$: You can store a single unsigned values in `x` by using all 32 bits.

Indices:	0
Bits:	0000 0000 0000 0000 0000 0000 0000 0000

- $k = 2$: You can store 2 unsigned values in `x` by partitioning the space into 2 groups of 16 bits each.

Indices:	1	0
Bits:	0000 0000 0000 0000	0000 0000 0000 0000

- $k = 4$: You can store 4 unsigned values in `x` by partitioning the space into 4 groups of 8 bits each.

Indices:	3	2	1	0
Bits:	0000 0000	0000 0000	0000 0000	0000 0000

- $k = 8$: You can store 8 unsigned values in `x` by partitioning the space into 8 groups of 4 bits each.

Indices:	7	6	5	4	3	2	1	0
Bits:	0000	0000	0000	0000	0000	0000	0000	0000

- Other valid values of k are 16 and 32. For example, when $k = 32$, you can store 32 unsigned values in `x` by partitioning the space into 32 groups of 1 bit each. Then each of the 32 values can take on a minimum and maximum of 0 and 1, respectively.

Specific Examples

Assume you have `unsigned int x`. In the examples below, spaces are provided for readability only. For any valid value of k each of the 0 through $k-1$ indices denotes one of the k groups of bits within the variable `x`. The values correspond to the base-10 representation of the binary digits within a particular group.

- $k = 4$: In the example below, the values 20, 11, 34, and 3 are all stored using a single `unsigned int` variable, respectively.

Indices:	3	2	1	0
Bits:	0000 0011	0010 0010	0000 1011	0001 0100
Values:	3	34	11	20

- $k = 8$: In the example below, the values 0, 8, 6, 0, 9, 3, 15, and 0 are all stored using a single `unsigned int` variable, respectively.

Indices:	7	6	5	4	3	2	1	0
Bits:	0000	1111	0011	1001	0000	0110	1000	0000
Values:	0	15	3	9	0	6	8	0

Code Description

For the code writing portion of this breakout/lab, you will need to write three functions:

- `void setValue(unsigned int & var, unsigned int k, unsigned int i, unsigned int val)`

  This function takes in our storage variable by reference, the number of values k, the index of a value to set (0 through $k-1$), and a value to set. You may assume that all inputs are valid. For example, assuming your `unsigned int` storage variable is `x`, in order to use the whole variable to store a value, you should be able to type something like `setValue(x, 1, 0, 1337)`. If you want to to be able to store 32 values (one value for each bit) and only assign the second bit to 1, then you should be able to type something like `setValue(x, 32, 1, 1)`.

  NOTE: Do NOT assume that successive calls to `setValue` increment the index by 1. Your function should be able to assign values to index positions in an arbitrary order.

  NOTE: Do NOT assume that the group of bits your are setting are already 0s. Your function should be able to assign values to the same index multiple times.

  NOTE: Since the variable is passed in by reference, changing its value inside of the function will result in its value being changed outside of the function as well. See DEITEL 6.13.

- `unsigned int getValue(unsigned int var, unsigned int k, unsigned int i)`

  This function takes in our storage variable by value, the number of values k, and the index of a of a value to get (0 through $k-1$). You may assume that all inputs are valid. For example, if `x` is the `unsigned int` variable corresponding to the first example in the "Specific Examples" section, then `getValue(x, 4, 0)` would return 20 and `getValue(x, 4, 2)` would return 34.

- `int main()`

  This function, the entry-point to your program, should prompt the user for a value of k, then prompt the user for k values. The `setValue` function should be used to set these values into a single `unsigned int` variable. After that, the `main` function should extract each of the k values from the storage variable using the `getValue` function and print them to the screen.

1 Group Brainstorm

You are NOT allowed to use the computers during this time.

Breakup into groups based on your seating and brainstorm about how to solve the problem or exercise. Make sure everyone understands the problem, and sketch out potential ways to move towards a solution. Perhaps something that was discussed during lecture might be useful?

2 Submit Individual Brainstorm

You may use a computer from this point forward.

Login to eLC and submit a version of your group's brainstorm, written in your own words. You may add additional information if you want. You need to write enough in order to convince the grader that you understand the problem or exercise and that you have a plan for moving forward towards a solution. Please include the last names of the other people in your group in your submission. The brainstorm submission should be available on eLC in your assignment dropbox. You should submit your individual brainstorms before the end of your breakout period.

NOTE: Submissions that do not include an individual brainstorm will not be graded.

3 Setup your Nike Account

Follow the instructions in this section to setup your `.bash_login` file.

1. Login to Nike.

2. Open up your `~/.bash_login` file (create it if it does not exist) in the plain text editor of your choice (e.g., emacs or vi). For example, if you are using Emacs, then this can be done with the following command:

   ```
   $ emacs ~/.bash_login
   ```

3. Copy and paste the following lines into the bottom of the file:

   ```
   ##################################################################################

   # cs1730 paths
   export GCC_HOME=/usr/local/gcc/6.2
   export GDB_HOME=/usr/local/gdb/7.11

   # update link library path
   export LD_LIBRARY_PATH=$GCC_HOME/lib64:$LD_LIBRARY_PATH

   # update exec paths
   export PATH=$GDB_HOME/bin:$PATH
   export PATH=$GCC_HOME/bin:$PATH

   # custom Prompt
   export PS1='\u@\h:\w\$ '

   ##################################################################################
   ```

   NOTE: If you already have a custom prompt, then you don't have to copy that part.

4. Logout of Nike, then log back into Nike. You can now verify if everything was setup correctly:

   - If you type in:

     ```
     $ gcc --version
     ```

     Then it should say you're using `gcc (GCC) 6.2.0`.

   - If you type in:

     ```
     $ g++ --version
     ```

     Then it should say you're using `g++ (GCC) 6.2.0`.

   - If you type in:

     ```
     $ gdb --version
     ```

     Then it should say you're using `GNU gdb (GDB) 7.11`.

4 "Save the Bits" C++ Program

4.1 Setup

Make sure that all of your files are in a directory called `LastName-FirstName-lab01`, where `LastName` and `FirstName` are replaced with your actual last and first names, respectively.

4.2 Code

For this lab, you should place your code into a file called `lab01.cpp`. Here is some more information about what is expected.

- All functions must be documented using Javadoc-style comments. Use inline documentation, as needed, to explain ambiguous or tricky parts of your code.
- The resulting executable must be called `lab01`. The expectation is that after your program is compiled and linked, the grader should be able to run your program by typing:

  ```
  $ ./lab01
  ```

4.3 `README` File

Make sure to include a `README` file that includes the following information presented in a reasonably formatted way:

- Your Name and 810/811#
- Instructions on how to compile and run your program. **Hint:** If you setup your submission to include a `Makefile`, then these instructions are pretty simple.
- A Reflection Section. In a paragraph or two, compare and contrast what you actually did to complete the problem or exercise versus what you wrote in your initial brainstorm. How will this experience impact future planning/brainstorms?

Here is a partially filled out, example `README` file: `https://gist.github.com/mepcotterell/3ce865e3a151a3b49ec3`.

NOTE: Try to make sure that each line in your `README` file does not exceed 80 characters. Do not assume line-wrapping. Please manually insert line breaks if a lines exceeds 80 characters.

5 Submission

Before your next breakout lab session, you need to submit your code. You will still be submitting your project via nike. Make sure your work is on `nike.cs.uga.edu` in a directory called `LastName-FirstName-lab01`. From within the parent directory, execute the following command:

```
$ submit LastName-FirstName-lab01 cs1730a
```

It is also a good idea to email a copy to yourself. To do this, simply execute the following command, replacing the email address with your email address:

```
$ tar zcvf LastName-FirstName-lab01.tar.gz LastName-FirstName-lab01
$ mutt -s "lab01" -a LastName-FirstName-lab01.tar.gz -- your@email.com < /dev/null
```