# Computer Org & Systems
# Comp 21000

Practicum 1

2 March 2020

**Due:** ~~Monday, 23 March~~. **Monday, 30 March** Name: _____

## Objectives

- Practice using masks
- Understand how compression works
- Understand how to use Make and multiple source files.

# 1 Masks and Packing

**Overview.** Write a C program that takes as its input *up to* 16 decimal integers, each integer with a value less than 256. You must store these integers into a dynamically allocated array. Each number must then be packed into an unsigned int. If an int has 32 bits, then 4 of the input numbers can be packed into one storage unsigned int. If you input more than 4 numbers, then you must pack the numbers into multiple unsigned ints. Also write an inverse packing function. It should unpack an unsigned int and return the original numbers. You must print out your packed integers and the hexidecimal number corresponding to the packed integer. Your program must repeat this process as long as the user wants.

So where would this be used in real life? Consider the C++ `enum`. The `enum` is represented at the machine level as an `int` but the C++ standard does not require an `int`. So if the `enum` is small (say, days of the week), then each value can be represented in a `byte` or even a `nibble`. That way the days of the week could be stored in 4 `bytes` (i.e., 7 days $\times$ 4 `bits`/day = 28 `bits`, but since memory is `byte` addressable, 32 bits or 4 bytes would be used). *This is a favorite interview question, by the way.*

## 1.1 Specification

- Input
  - Ask the user to enter the number of integers to compress

- There will be no more than 16 integers.
- Read and store the integers in a dynamically allocated array.
- Each integer may take a value from 0 to 255 (don't let the user enter a larger number).
- After you have read in all the numbers, print them out.

- Packing.
  - Pack all of the input decimal integers into unsigned int variables.
  - Determine the size of an int in your program (use the sizeof operator). This size will vary from computer to computer, but your program must work on any UNIX computer. You may assume that the size will be at least 32 bits.
  - Each unsigned int variable must contain 4 of the input integers (in their binary form). For example, if the first four numbers entered were 35, 9, 100, and 208, You would pack the first integer variable with 00100011 00001001 01100100 11010000.
  - Note that the first number entered, 35, is the leftmost number in the packed integer variable.
  - If less than 4 integers are entered, the first entered integer must still be packed into the leftmost position of the unsigned integer.

- Printing.
  - Print out all of the packed integers as unsigned integers. For the example given above, you would print 587818192 as the integer value of the packed variable. Sometimes the printed number will appear as a negative number.
  - You must only print integers that contain packed numbers. If, for example, 8 integers are entered you must only print two packed integers. Do not print integers that are not used.

- Converting.
  - Next convert the packed integers into their corresponding hexadecimal format using only bitwise operators and print them out. Note that these are printed out after you have already printed all of the input numbers and the packed numbers as integers. The packed integer above would print out 23 09 64 D0 as the hexadecimal value.
  - You must use an algorithm like that described in class (similar to the octal.c example) to convert the integers to hex (you may **not** use the %x formatting parameter in printf).
  - You may **not** use the div ('/') or the mod ('%') operator in your hex conversion function; rather use bitwise operators.
  - You may **not** convert numbers to hex by putting hexadecimal digits into an array and indexing into the array.

- Unpacking.

  - After you print out the packed integers, call your unpack function, unpack each number and print it out as a decimal using `printf`.

- Continuing. Now ask the user if she wants to repeat the process. If she answers 'y' or 'Y', continue this entire process, otherwise halt.

## 1.2 Constraints and Observations

- You must use `Make` and a `Makefile`. Your makefile must check on *two* .c files and compile them together if either is out of date.
- Your program must reside in at least two separate .c files. You may not include one of these in the other, instead compile them together.
- You must only use pointer syntax with arrays. All arrays must be dynamically allocated.
- You **may** use a library function to convert a string to a number or vice versa.
- Only positive integers may be entered. You must check the range and allow the user to continue to input a number until it is in the proper range.
- The conversion to hexadecimal numbers and the packing must be done using bitwise operators. You may use addition and division in your program, but **not** in the conversion or packing code. Note that the packed number is unsigned, so your corresponding hex digits should just be a translation of the bits in the unsigned number; *don't interpret the bits as a two's complement number*. Note that you cannot use an "if" statement or "switch" statement to determine values. You may **not** put hexadecimal digits into an array and index into the array.
- Conversion to hex, packing and unpacking must all be done in separate functions. So you'll need *at least 3 functions* in addition to the main function.
- **You can assume that there will be no more than 16 integers entered.**
- You may not use output formating (e.g., in printf) to print hexadecimal numbers. Rather you must convert the packed integers to hexadecimal strings explicitly and print out the resulting characters.
- Your program must determine the number of bits in an integer on your machine. The library file limits.h contains a constant CHAR_BIT represents the number of bits in a char variables, which will be the size of a byte (normally 8 bits) on your machine.
- The first number read in must be packed into the leftmost bit positions of the packed integer.
- You may **not** have hard-coded numbers in your program. If you need to use a number, create a constant using the `define` preprocessor command. This includes numbers that indicate shift amounts, sizes, etc.
- Your code must be commented in accordance with the programming guide (see the link under the "Course Materials" menu item on the class web page).
- Don't forget to use `classes` in `scanf` if necessary to get the correct input.

## 2 Submission

1. Save your two programs and your makefile in a **directory** named `YourLastName`, `chmod` permissions to 700 and `cp` it to the directory `/home/barr/Student/comp210/Turnin/pract1` on the `server`. Note that you can write to this directory but cannot read from it so you will not be able to see the file once you've written it to the directory.