Jim's
New C Programming Problems

The *New* First 108

gcc –ansi –pedantic -Wall

A Varied Collection of Problems for Beginning Students

Version 1.0

Last Updated – September 2, 2011

Programming Style Guide for 91.101 and 91.102
Version 2.0

1. All files within programs should contain the standard header. This is a comment based banner box with a perimeter of asterisks. Inside this box will be the programmer's name and the title of the program. It should also contain an entry that estimates that amount of time you spent to create it.

```
/**************************************************************/
/* Programmer: Herschel Burke Gilbert                         */
/*                                                            */
/* Program 1: Hello World                                     */
/*                                                            */
/* Approximate completion time:   15 minutes                 */
/**************************************************************/
```

2. Your program should use functions when reasonable to do so.

3. Your functions, unless that is there sole purpose, should not contain input or output statements.

4. Your program should use our 91.101/91.102 standard main template as follows:

```
int main( int argc, char *argv[] ) {



    return 0 ;
}
```

    Notice that the first { is on the same line as the int main …

5. Your indentation strategy must be reasonable and consistent. No jagged edges

    Declarators shall be indented 2-4 spaces.
    Executable statements shall be indented the same as declarators and they should all line up.
    The body of a loop shall be indented.
    The body of "then" and the body of the "else" shall be indented.

6. Your comments, prompt, and output should not contain any spelling errors.

7. Your code should not appear cramped. Typically, binary operators will be surrounded by blanks.

$$a = b + c \text{ rather than } a=b+c$$

    You should use a single blank line to set off logical chunks of consecutive statements. Let your code breathe.

8. Do not put a blank between the function name and its (.

```
int foo( int a ) ;
```

9. You should not overcapitalize, particularly your prompts to the user.

10. Your sentences should end with a proper punctuation mark.

11. Errors messages typically should not be reported in the function that found them. Rather, a status code should be returned back to the caller.

12. You should strive to write clear code that is not going out of its way to confuse the reader. In general, direct and plain will always be preferred over indirect and clever.

13. If you open a file, you should close it. This is normally done just prior to returning back to UNIX.

14. Do not intermingle your variable definitions with executable statements. Batch up your variable definitions at the top of the body of the function. Once you have written an executable statement you should not be writing additional variable definitions. This can be violated in the case of a compound statement.

15. Header files do not reserve space. They do not ever contain function definitions. They do not ever contain variable definitions. Type definitions do not reserve space and they are often placed within a header file.

16. Do not write useless include statements. For example, do not use `#include <stdlib.h>` if the code does not use anything declared within `stdlib.h`.

17. Your program should be compiled with both the –ansi and the –Wall options. Doing so should not generate any error or any warning messages.

18. Use meaningful variable names. Variables that are used as loop indices are often i, j, and k. You should continue with this convention unless there is a good reason to deviate.

19. You should seek to build and call functions that are useful and functional.

20. You should not create a cascading function that is only there to reduce the size of a body without regard to its purpose. Functions should have a single purpose. The purpose should not be "*code which comes after other code I just wrote.*"

21. Keep your non-local variables to a minimum. Perhaps they help with an abstract data type? Perhaps you really need them for some special purpose? But if you use them, please have a good reason. Convenience is not often a good reason.

22. Use the adjectives static and extern appropriately when building programs that span multiple files. Some say the use "static" functions hamper debugging in large systems.

23. If you need to have global types, global enumerated types, and/or truly global variables, then batch them into a file called globals.h and/or globals.c (for definitions).

24. Before you pass in your code take one last long look at it. Is is clean? Does it appear snappy? Does it have any ragged edges due to random indentations? Does it contain any spelling errors? Do your prompts make sense? Is your output labeled? Are you being courteous to your reader? Are you being courteous to your user? Are you being courteous to your grader? Does your code look rushed? Is it done with a pride?

25. Re-read item 24 above. I think you should create a checklist to remind yourself of item 24.

```
26.                          ssh (secure shell)
```

So, you are at home or you are in the dorm room or at the library or Barnes and Noble. You have your laptop or computer with you and you are filled with passion to get some 91.101 work done. You need to get to a terminal window. If it were me, I would then type:

% ssh canning@.cs.uml.edu <ret>

You would insert your own username in place of canning.

You'll be prompted for your password. After typing your password in, you will be working on mercury. You will be home free.

```
                        scp (secure copy)
```

So, you are at home and you have created working program, say p1.c, on mercury, but now you wish to print this file out on your home printer. You will need to copy p1.c from mercury to your local machine. One way for you to do this is to pull the file from mercury and have it placed on you machine.

So, while you have a terminal window on your machine, …

%scp canning@cs.uml.edu:~canning/101/p1.c   p1.c

Of course, this is an example. You will need to use your username and the correct pathname of the file.

You will be prompted for you password.

Now, a copy of the file will be sitting on your local machine.

Perhaps you have a C compiler and a UNIX environment with emacs on your local machine. You do your work in that environment, but now you need to get your program onto mercury so that you can electronically submit it.

%scp p1.c canning@cs.uml.edu:~canning/101/p1.c

You will be prompted for your password.

You should try these out as soon as possible so that you will have mastered them before you need them in a crisis.

Problem 1: Hello World and a Bit More

Write a C program that prints out two things. First it prints out the message Hello World. It will do this by calling the standard C function *printf*. Know that when the message gets printed out, that is the side-effect. I want you to also print out the value returned by the call to the function printf. Know that the call to *printf* is an expression. It reports out a value. That value has type…. and .... there is a side-effect. I do not want you to explicitly count the number of characters in Hello World and just print this out. You must print out the value that is returned by the call to the function *printf*. A working version of this program can be found in ~canning/public/101/fall2011/programs. It is called helloworld.

Problem 2: The sizeof Operator

The C programming language has 45 operators. Thirteen of these operators have a side-effect, the other 32 operators do not have a side-effect. One operator that does not have a side-effect is the sizeof operator. When this operator is applied to a type, it will evaluate to the number of bytes it takes to store an object of that type. I want you to write a C program that will output the number of bytes it takes to store an a char, a short int, int, a long int, a long long int, a float, a double, a pointer to an int, and a pointer to a float. A working version of this program can be found at ~canning/public/101/fall2011/programs. It is called sizeof.

Problem 3: Left Turn, Right Turn, or Straight?

Watch two Youtube videos.
Here is the first one: http://www.youtube.com/watch?v=21LWuY8i6Hw
Here is the second one: http://www.youtube.com/watch?v=-3FnLLfvKXs&feature=related

Recall our discussion in class.

You are to write a C program that will prompt the user for three points. Each point has an integer x value and an integer y value. The points are orders. The first point entered is the first point. The second point entered in the second point. The third point entered is the third point. Your program is figure out if the there points make a left turn, go straight, or make a right turn. A working version of this program is located in ~canning/public/101/fall2011/programs. It is called turns. All of your logic should not be inside of the main function. You should create a separate function.

Problem 4: Go Left Young Folk, Go Left

You are to write a C program that will prompt the user to enter 5 points. Each point has an integer x value and an integer y value. The points are ordered and they form a convex quadrilateral. Your program should determine and output whether or not the 5th point is located inside the quadrilateral or not.

Problem 5: Iterative Persistence

Multiplying the digits of an integer and continuing the process gives the surprising result that the sequence of products always arrives at a single digit number. For example,

       715 ---- 35 ---- 15 ---- 5

27 ---- 14 ---- 4

4000 ---- 0

9

The number of times products need to be calculated to reach a single digit is called the persistence number of that integer. Thus, the persistence number of 715 is 3, the persistence number of 27 is 2, the persistence number of 4000 is 1, and the persistence number of 9 is 0.

You are to write an iterative program that will continually prompt the user to enter a positive integer until EOF has been entered via the keyboard. For each number entered your program should output the persistence of the number. Please note that the correct spelling of persistence is p-e-r-s-i-s-t-e-n-c-e. The word does not contain the letter "a".

Problem 6: Recursive Persistence

Redo problem 5, only this use recursion.

Problem 7: Taxi! Taxi!   *(Taken from a programming competition)*

Background information: In the early part of this previous century and Indian bookkeeper named Ramanujan traveled to England to work with the famous English mathematician Hardy. Hardy had been sent papers written by the untrained Ramanujan, determined that the writer was a genius, and sent money so that Ramanujan could get to Cambridge. The two men collaborated for many years. Once when Ramanujan was ill and in the hospital, Hardy went to visit him . When Hardy walked into the room his first remark was, "I thought the number of my taxicab was 1729. It seemed to me a rather dull number." Ramanujan replied, "No, Hardy! It is a very interesting number. It is the smallest number expressible as the sum of two cubes in two different ways."

If you check, you will see that Ramanujan was correct.

$$1729 = 1^3 + 12^3 = 1 + 1728$$
$$1729 = 9^3 + 10^3 = 729 + 1000$$

The next positive integer with this property is 4104.

$$4104 = 2^3 + 16^3$$
$$4104 = 9^3 + 15^3$$

We call numbers like 1729 and 4104 Ramanujan Numbers.

You are to input from the keyboard a positive integer N less than or equal to 1,000,000.

You are to output to the screen a table of Ramanujan Numbers less than N with the corresponding pairs of cubes, and the cube roots of these cubes. The table must have appropriate headings and also include the order of each Ramanujan Number. The order of 1729 is 1 because it is the first such number.

Example: For input value N = 5000, the output should be:

```
Ramanujan    First    Second    First   Second   Order
Number       Cube     Cube      Root    Root
-------------------------------------------------------
 1729          1      1728        1       12        1
             729      1000        9       10

 4104          8      4096        2       16        2
             729      3375        9       15
```

Program 8: Using Atoi

Write a program that obtains two positive integer values, one from argv[1] and the other from argv[2]. The second number is a single digit. Recall that these numbers will have a "string" type and thus they could be converted to a two's complement representation(a type integer). Your program is to determine the number of times the single digit found in argv[2] occurs within the integer value contained in argv[1]. For example, the digit 3 occurs 4 times within the number 89323313.

Problem 9: Binary Printing

The C programming language has %c to print a character. It has %d to print an integer. It has a %f to print a float. You can find all of these codes in your best friend. However, there is no mechanism to print a number out as a binary. Please write a program that will input an integer value via scanf and then it will print out this value as a sequence of 0's and 1's that represents its two's complement representation. You cannot use either itoa or sprintf. I want you to do this at the bit level and use a shift operator.

Program 10: It's a Message

The underdeveloped but upwardly mobile country of Galosh has need of a secret code in which to express its diplomatic dispatches, so the head cryptographer proposed that first, all sequences of nonvowels (including spaces and punctuation) be reversed. The letter "y" is NOT considered to be a vowel. Then, the entire string message should be written backwards.

A program is to be written so that a message sent from the Prime Minister to various aides can be decoded. Two examples of messages and their encoded versions are:

HELLO THERE.
.ERE THOLLEH

MAKE THAT EXTRA CHEESE. PEPPERONI GIVES ME HEARTBURN.
RN.URTBAE HES MEVI GINOREPPE. PESEE CHAXTRET A THEKAM

All messages will contain either the capital letters, digits (0-9) , a blank, or the following two punctuation characters: .   ,   and will be restricted to at most 60 characters. Your program should accept any number of encoded messages from the file named message.dat and write their associated decoded results to the display.

Program 11: Command Line Sum

Write a C program that will sum up the integers on the command line. For example, suppose the program is launched with

% a.out 4   511 6   <ret>

then your program should output the number 521. If the program is launched with

%a.out 1 4 9 16 25 36 <ret>

then your program should output the number 58.

Program 12: MyAtoi

Write a program that converts a string into an integer. The string will be a string of digits. You must do this by building your own MyAtoi function. You should not use the C library atoi. Your program should obtain its data from argv[1]. Your program should output the integer.

Problem 13 Tiling Patterns

This problem comes from a programming competition.

We plan to lay a floor of M by M tiles. Each tile is square and we have an inexhaustible supply of N different color tiles. Label the colors C1, C2, C3, ... , CN. We begin by placing a square of C1 in the lower left corner and labeling this tile the (1,1) tile. We then put a tile of color C2 to the right of (1,1) and label it (1,2). We put a second tile of color C2 above (1,2) and label it (2,2). Now, we put a tile of color C3 o the left of (2,2) and call it (2,1), a tile of color C3 above (2,1) and call it (3,1), a tile to the left of (3,1) and label it (3,2), etc. Our pattern is to use 1 of color C1, 2 of color C2, 4 of color C3, 6 of color C4, etc. When we get to color CN, we start over again with color C1. For each color, we use two more tiles than we did for the previous color. (This is not true at the very beginning. It is true for all other cases.) The label of the tile represents its row and col beginning the labeling from the lower left. Carefully examine the example below to see the developing pattern for laying the tiles.

Ex. If N is 4, our pattern is

```
    7  C3  C3  C3  C3  C3  C3  C3
    6  C3  C3  C3  C3  C3  C2  C4
    5  C1  C1  C1  C1  C1  C2  C4
R   4  C1  C1  C1  C4  C2  C2  C4
O   3  C3  C3  C3  C4  C2  C2  C4
W   2  C3  C2  C4  C4  C2  C2  C4
    1  C1  C2  C4  C4  C2  C2  C4

       1   2   3   4   5   6   7

            COLUMN
```

Write a program to read the number of colors, N, from the first line of input, the number of rows of tiles, M from the second line of input. (The number of columns is the same as the number of rows.) Then read two values of X and Y and print the number of the color for the tile labeled (X,Y). X represents the row and Y represents the column.

The value of M will be less than or equal to 10000. All input will be legal.

Example: If the input is:

4
100
3
5

the output of your program should be 2.

Note: You must malloc up your space for the array.

Problem 14: Lowermost and Leftmost

A file named *points* has a positive integer value, say n, as its first line. The next n lines contain a pair of integer values. The first value on each line is x and the second value is y. These (x,y) values represent points in the cartesian plane. You are to read in these points into an array of structs. The space for the one-dimensional array needs to be *malloced-up*. Of course, each element of the array is a struct. The struct for this problem will contain two fields: an x field and a y field. There are no duplicate points.

After you read in the all the points, your program is to determine the point that is lower most - leftmost. Strictly speaking, you do not need an array to solve this problem, but I want you to practice the following:

- creating space for a one-dimensional array during run-time.

- passing an array to a function.

- returning a struct

You must use the fopen and fclose functions.

Problem 15: Bubble Sort of n Random Positive Integers

You are to write a C program that enters a positive integer via the command line. Remember this number comes into your program as a string. You should malloc up enough space to create an array big enough to hold n integers. You should call the random number generator, rand(), to fill that array with random numbers between 0 and 99,999. After the array has been filled, you should call a bubblesort sorting function which will sort the numbers into ascending order. Your program should output the smallest number, the largest number, and the median value of the numbers. Recall, that the median value of an odd number of numbers is the middle number. The median value of an even number of numbers is the average of the middle two numbers. The bubblesort algorithm you should write is a straight bubblesort -- like the one we discussed in class. Do not increased its speed with special optimizations.

Program 16: Inner Product

The first value in a file is a positive integer, say n. Then next n values represent the values of a vector A. The subsequent n values represent the values of the vector B. The values of vector A and vector B are real values and thus the one dimensional arrays that will hold them should be defined and declared to be *double*. The inner product of two vectors is the sum of the product of corresponding entries. Write a program that will compute the inner product of the vector A and the vector B. The name of file that contains the values is to be entered on the command line as argv[1]. Your program should call a function that, when called, will pass A, B, and n. The name of the function is *inner* and it should return a double.

Your program is to output a value which is the inner product of these two vectors.

Remember. The real practice here is for you to dig into your text book. It is your best friend. You are striving to become independent. You can do it. You are not rushing about just trying to get the program to work so you can collect points. Yikes, No. Are you reading five pages per day. If not, start right now. Make it 7 pages per day. There is value in using your textbook's index. This is your time to be a student. Do not waste it. Find the time to be a student. Read a novel. Even though none of your teacher's have asked you to do this. You can start with The Little Prince. A very short book. There are some things for you to know. Here are some. Every memory cell is associated with two numbers: an address and a content. Bill Russell is the greatest basketball player of all-time. Do not equivocate on this. Do not wear a baseball cap in class. Do not talk with students who sit around you. The function scanf reports out a value. This value is the number of successful conversions AND there is a side-effect. Do not slouch. Your body language screams, it never whispers. This is your time to be a student. Be a student. It is your primary job. It is your secondary job. Read the book. Read it twice. Do this for all of your classes. Play catch. You could go back and retake every single math MCAS test from 3rd grade on. It would not be a waste of time to do so.

Problem 17: Igpay Atinlay

For this problem, you are to translate a paragraph of ordinary English text into Pig Latin. Unlike real Latin, the translation rules for Pig Latin are simple.

- If a word starts with a consonant, then translate the word by placing the initial letter at the end of the word and appending "ay". Example: cow becomes owcay and blank becomes lankay.

- If a word starts with a vowel, then simply append "way" to the word. Example: apple beomes appleway and order becomes orderway.

The letter y should be considered to be a consonant.

The input format of the text is a series of lines, each line containing a set of words to be translated. A line containing five periods indicates the end of the data. For example,

the quick brown fox

jumped over the lazy dog

because it would not move

.....

The name of a file containing the text will come into your program via argv[1].

Words will be separated by one or more blanks.

No other punctuation will be used.

Of course, the newline character \n is used to end a line.

Dig into your best friend if you need to. Part of your training is to practice finding information. The textbook is your best friend.

Program 18: Hadamard or Bust. Hydrate that Thing

You need to master the concept of divide and conquer. You will do this by coding up a number of such algorithms. Practice is good. We are not Alan Iverson. Of course, Alan Iverson was on top of his game when practice became boring. He did practice to become the best. At any rate, we will start our journey by populating a matrix with ones and negative ones to form a Hadamard Matrix. Your program will enter a power of 2 from the command line via argv[1]. This will be n. You need to malloc up the space for an n x n two dimensional array using either option 1 or option 2. Populate the matrix as was discussed in class. This is a beautiful little assignment. It will help you take the first necessary step. The output of your program is the final n x n matrix. It should be readable to the grader. Do not annoy the grader. Be courteous.

Program 19: Quick Sort Them Thetas

In class I showed you how to calculate the angle theta that is formed by using the lowest most, left most point as the reference point. So please read in n points from a file. Find the lowest most, left most point, and then calculate the thetas. After you calculate the thetas, please sort the points in increasing angle. The lowest most left most point is the first point. A file, whose name is given in argv[1], contains a value n. It is then followed by n lines. Each line contains an x value and a y value. You may assume that no point is duplicated.

Program 20: Parenthesis Checker with a Non-Polymorphic Stack Module

Create a file called stack.h. Create a file called stack.c Together they will form a non-polymorphic stack module that can push and pop single characters. This was discussed in class. Use these two files in conjunction with main.c to build a parenthesis checker to determine if parenthetical are balanced or not. The only symbols are (,), {, }, [, and]. In addition to these three files, I want you to create a makefile as was also discussed in class. Remember! Come to class! When you submit your program, you should submit your makefile.

Program 21: Complex Numbers

Create a file called complex.h. Create a file called complex.c Together they will form a complex number module that can add, subtract, multiply, and divide complex numbers. These functions should receive structs as arguments and they should return a struct. For this assignment, I do not want you use pass and return pointers to structs. You need to create a grader friendly main.c that will prompt the user to enter two complex numbers and then provides the user with a simple menu allowing the user to select either add, subtract, multiply, and divide the two complex numbers.

Program 22: Rational Numbers

In my ~canning/public/101/rational directory you will find four files: globals.h, rational.h main.c and a.out. You should copy these three files to your directory by using the unix cp command. You need to write a C program that will test a rational number module. The rational number module should include the functions:

I have provided you the exact main.c file that you should use. I have provided the exact globals.h file that you should use. I have provided you with the exact rational.h file. You need to one source file and a meaningful Makefile. The one source files is rational.c. Your working program, when executed, should provide the same results as my program does. You should run my a.out many times so you can get a feel for what it does. I want your output to be presented in reduced form. This will cause you to create a greatest common divisor function as well. I will orient you to all of this in class.

Program 23: Filling the Fourier Matrix

Write a program that inputs a positive integer n from the command line. The integer should be a power of 2. Your program should then fill a Fourier matrix with sine and cosine values as discussed in class. Your program should first print out the values of the sine matrix then it should output the values of the cosine matrix. Label your output so the grader knows what is going on.

Program 24: N Choose K Recursive

Write a program that continually prompts the user to input two positive integer values from the keyboard, say n and r, where r <= n. For each pair of n and r you should output the value of **n choose r** as discussed in class. Your solution should be recursive. Your program should exit the loop when end of file is generated from the keyboard. Make sure you make your program into a "thing of beauty".

Program 25: Rational Numbers 2

This problem differs from problem 22 since we will be passing pointers to structs rather than just the structs. We will also be returning the value of the "answer" through the argument list rather than using the return mechanism of a function. The prototypes given below provide you with the declaration-onlys of the functions for the new rational module. From this you should be able to make a new main.c, a new rational.h, and a new rational.c. You can use the same globals.h as in problem 22.

The new main.c you create should perform the same functionality as the one in problem 22, however you will need to convert it. You should submit all the files (main.c , rational.h, and rational.c) to the grader.

Keep in mind, in this assignment are motivated to use pointers to structs for two separate reasons. First, in some cases data in the calling function will be altered/changed/updated and this change needs to be done to the original data structure, not a copy of the data structure. The other reason is that the programmer simply does not wish to have an entire struct copied -- there is too many bytes to be copied ... thus, the programmer just copies the address of the struct even though no data is being altered. The data is just being read (used).

```
void load_rational(  rational *p_rational, int numerator, int denominator ) ;

void retrieve_rational( rational *p_rational, int *p_numerator, int *p_denominator ) ;
```

void add_rational(rational *p_result, rational *p_rational1, rational *p_rational2) ;

void subtract_rational(rational *p_result, rational *p_rational1, rational *p_rational2) ;

void multiply_rational(rational *p_result, rational *p_rational1, rational *p_rational2) ;

status divide_rational(rational *p_result, rational *p_rational1, rational *p_rational2) ;

bool equal_rational(rational *p_rational1, rational *p_rational2) ;

 void print_rational(rational *p_rational1) ;

Problem 26: Graham Scan Method - Convex Hull

In class we discussed all the steps necessary to find a convex hull using the graham scan method. Now it is time for you to implement this in C. I want you to use a non-polymorphic stack module. Your program should obtain its data from a file. The name of the file is given on the command line via argv[1]. The first line of the file contains a single positive integer value, say n. The next n lines of the file each contain a pair of integer values separated by a blank. Each pair of numbers represents a point in a two dimensional cartesian coordinate system.

Your program should output a list of all the extreme points of the convex hull formed by these collection of points.

Program 27: Matrix Times a Vector

The first line of a file contains two positive integer values, say r and c. They represent the size of an r x c matrix of doubles. The next r lines of the file each contains c numbers separated by a space. You need to read these values into an r x c matrix. The next c lines in the file each contain a single number. Taken together they represent the values to be stored into a one-dimensional array of doubles. As discussed in class, I want you to write a program that will multiply the matrix times the vector Your main program should dynamically create space for the array using the option 2 malloc strategy. You should also malloc up the code for the one-dimensional array. Your main program should call a function that will malloc up the space for the answer, computer the answer, and then return the answer using the return mechanism of functions. Your main program should print the answer. The name of the file is given on the command line as argv[1].