Jim's
C Programming Problems

The First 101

gcc –ansi -Wall

A Varied Collection of Problems for Beginning Students

Version 3.0

Last Updated – December 24, 2009

http://www.uml.edu/Sciences/computer-science/faculty/canning-james.aspx

Programming Style Guide for 91.101 and 91.102
Version 1.0

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.
   Exectuble 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, operators will be surrounded by blanks. 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 it arguments.

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.

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@mercury.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@mercury.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@mercury.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.

I hope you enjoy these problems. In the past, many students have enjoyed them, some have not. Which group will you be in? I am rooting for you to be in the first group. Once you start to conquer them, they can all tumble over. I am trying to make you as strong as possible with the time we have together. You cannot learn to play the clarinet unless you practice. You cannot learn to speak a foreign language unless you practice. You cannot learn to solve problems in C within a UNIX environment unless you practice. Please dedicate a portion of each day to solve one or more of these problems. Give yourself a chance to feel great. Start solving as many of these problems as you can. Do not procrastinate. It is not hard once you get oriented. It does take time. You can do this. It will be worth it.

Problem 1: Hello World

Write a C program that prints out the message *Hello World*.

Problem 2: The Value 6

Write a C program that prints out the value 6. I want you to use the %d format code.

Problem 3: The Character P

Write a C program that prints out the single character P. I want you to use the %c format code.

Problem 4: The scanf Function

Write a C program that reads in an integer value from the keyboard via the *scanf* function and then prints it back onto the screen. You should know that *scanf* seeks an address expression. For example, *&n* is an address expression. You should also know that *scanf* returns a value. It is either the number of successful conversions or an error code that is typically EOF. Please read about the *scanf* function in Appendix D of your textbook.

Problem 5: Sum of Two Values

Write a C program that reads two integer values from the keyboard via the *scanf* function, then adds them together, stores the result into a variable called sum, and then prints out the value of the variable sum.

Problem 6: The fscanf Function

Using the emacs editor, create file called *testdata6* that has a single integer value stored into it. Write a C program that opens a file named *testdata6,* reads the single value stored in it using the *fscanf* function, and then prints the value back onto the screen. You should call the function *fclose* before you return to the Unix operating system. Read about the function *fclose* in Appendix D of your textbook. Mel Ott hit 511 home runs.

Problem 7: Bigger than 100?

Write a C program that reads a number from the keyboard via the *scanf* function, and then prints the message *The number is bigger than 100* if it is. If the number is 100 or less than 100 then you should print out the message: *The number is not bigger than 100.* Make certain that your code looks good. No ragged indents.

Problem 8: One Horizontal Line of Asterisks

Using the emacs editor, create a file called *testdata8* that has a single integer value stored into it. The integer will be a number between 1 and 30 inclusive. Create a C program that uses the *fscanf* function to read in this integer value and then prints a horizontal line containing that many asterisks' (*). Use a for-loop in your program. For example,

If the file contained the integer 8 your program would print out: ********
If the file contained the integer 15 your program would print out: ***************

Problem 9: Using a For Loop

Using the emacs editor create a file called *testdata9* that contains 5 positive integer values. Write a C program that reads and prints all five values. Each number should be read using the *fscanf* function embedded inside a for-loop. You should not use an array. You should a for-loop.

Problem 10: Sum of Twenty

Write a C program that reads in 20 positive integer values from a file called *testdata10* and computes their sum, and then prints the value of the sum onto the screen. Your program should not use 20 distinct variables. Your program should not use an array.

Problem 11: Equal to Zero?

Write a C program that reads a single integer number from the keyboard via the *scanf* function, determines if the number is equal to zero or not, and then either prints out: *"The number is equal to zero."* or *"The number is not equal to zero."*

Problem 12: Positive, Negative, or Zero?

Write a C program that reads a single integer number from the keyboard via the *scanf* function, determines if the number is either, positive, zero, or negative and then prints out a message such as: "*The number is positive*." or "*The number is zero*." or "*The number is negative*."

Problem 13: The abs Function

Write a C program that reads a single integer value from the keyboard. The number could be either a negative number, a positive number, or zero. Your program should call the absolute value function – *abs* – and it then prints out the absolute value of number entered. You can browse appendix D to determine which header file you should include. Your textbook is your best friend. I do not want you to create your own absolute value function. Make sure that your code looks snappy. What was the trouble in River City?

For example,

bash> ./a.out -6	would cause 6 to be printed.
bash> ./a.out 4	would cause 4 to be printed.

Problem 14: Argc

Write a C program that prints the number of command line arguments. This is the number that is stored in the argument: *argc*. Do not lose points because your code has a ragged indent or a misspelled word. For example,

```
bash>  ./a.out                                  would print out 1
bash>  ./a.out 511 Mel Ott Life is good         would print out 7
bash>  ./a.out  Jem and Scout                   would  print out 4
```

Problem 15: Using the sqrt Function

Write a C program that asks the user to enter a floating point number from the keyboard and then prints out the square root of that number. You should call the *sqrt* function. You can browse Appendix D to which header file you should include. The textbook is your best friend. Read it. You should know that the *sqrt* function definition is not located in libc.a. It is located in libm.a and thus you will need to use the –lm option when you try to build the executable image via the gcc utility. You also need to include math.h.

Problem 16: Sine Function

Read an integer or floating point value from the command line. It will come into your program as argv[1]. Remember that argv[1]'s type is a pointer to a character that begins a sequence of characters that end in a null byte. In other words, it is what is agreed upon as a string.

For example,

```
bash>  ./a.out 4.3
```

Print out the trigonometric sine value of this number. You would do this by calling the proper function that is found in Appendix D of you textbook.

Program 17: Count Characters

Write a program that counts the characters entered into the program from the keyboard until EOF is reached. Use the *getchar* function. Your program should output the number of characters entered.

Program 18: Solid Box of Asterisks

Your task is to write a C program that accepts two positive integer values from the keyboard, say L and H. Both L and H will be less than 21. Your program then prints out a solid of box of asterisks, with L horizontal stars and H vertical stars. The box is filled with asterisks. For example,

If you were to type in 5 and 3, you output would be:

```
*****
*****
*****
```

Problem 19: Area of a Rectangle

In grade school you learned that the area of a rectangle is its length times its height. Write a C program that inputs two floating point numbers that represents the length and the height of a rectangle. Your program should output the

area of the rectangle. If you use the type double to declare your variables, you will need to use the %lf format code with *scanf*. See the back cover pages of your book. If you use the type float to declare your variables, you will need to use the %f format code with *scanf*. The function *printf* , on the other hand, use %f for both double and float types.

Problem 20: Area of a Circle

In grade school you learned that the area of a circle is pi * r^2 . Write a C program that inputs a single floating point number representing the value of the radius r. Your program should output the area of the circle with a radius r. Now, the only problem is what value should you use for pi? Hmmm. Well, inside of math.h is a the following line:

#define M_PI 3.14159265358979323846 /* pi */

so you need to include math.h and then use M_PI in an expression to calculate the area. I do not want you to type in this #define. I want you to include math.h. However, with the –ansi option you will need to add another option, otherwise things won't go too well.

Try to build this program using: gcc –ansi –Wall p17.c.

You should see that trouble exists. It will not recognize M_PI even though it is inside math.h. To get past this, you need to build the program by either dropping the –ansi option or by adding the –D_GNU_SOURCE option. I prefer you choose the latter:

gcc –ansi –Wall –D_GNU_SOURCE p17.c

You do not need to build the program with the –lm option since you are not missing any definitions from libm.a. You need to understand the previous sentence. Make certain that you do understand it.

Program 21: Argv

Write a C program that will print out each command line argument on a separate line. Use a for-loop. The loop should have an index variable *i* that ranges from 0 to (*argc-1*). Remember the first command line argument is called argv[0] and its type is string so you can print it out with a %s format code. You should also recall that the character combination *\n* means newline.

Program 22: Reverse the Command Line

Write a C program that will print out each command line argument on a separate line. However, you should print them out in reverse order. That is, the last command line argument is printed first. Do not have an off-by-one error. Make sure that your code looks clean. Review the 101 Style Guide.

Program 23: Scanf Returns What?

Create a file and call it *testdata23*. The file should contain an unknown number of integer values. Write a C program that uses a while loop to read in each number and then prints the number back onto the screen. The boolean expression of the while loop must contain a call to the *fscanf* function. You need to read your book. Look at page 63. Study the code there. Make certain you understand why it is this way.

Program 24: One Dimensional Array

Create a file and call it *testdata24*. The file should contain exactly 15 integer values. Read these 15 numbers into a one-dimensional array. Then print the numbers out in the reverse order from which they were entered.
Use an array. I do not want you to create 15 distinct variables. Read your book. Become independent.

Program 25: Sum of a Bunch

Create a file and call it *testdata25*. The file should contain an unknown number of integer values. Write a program that will read in these numbers until end of file is reached and then outputs the sum of the numbers. You should not use an array to do this problem.

Program 26: fgetc and toupper

Write a program that reads in a text file character by character using *fgetc*. The name of the text file is *testdata26*. Your program should print out an exact copy of the file it read in except that all lowercase letters are converted to uppercase letters. You may use the function *toupper* (see appendix D). You should also use the function *putchar*.

Program 27: Reverse

Write a program that reads in 10 integer numbers from the keyboard using scanf. Your program should then print out the numbers in reverse order. Your program should store the 10 numbers into a one-dimensional array. Remember, scanf wants you to provide it an address expression. Do you see that this problem requires you to "hold" all the numbers at once before you can generate the answer? This was not the case when we computed the average of some numbers. To compute the average we did not need to "hold" the numbers so an array is not needed or wanted. But, here it is.

Program 28: Digit Sum

A file contains a bunch of positive integers. Write a program that reads in the numbers, one by one, and then outputs the sum of the individual digits. For example, if the number 6115 is read in, then your program would output 13. Your main program must call a function that receives an integer argument and returns the sum. This function should not print out the answer, it should only compute and return the answer. The main function will print out the answer. The name of the file is given on the command line as argv[1]. You should know that argv[1] is already a string expression. It does not need to be enclosed in double quotes when you use fopen.
Remember, close the file before you exit the program.

Program 29: Find the Average

Get into the emacs editor and create a file named testdata26 that contains 4 integer values. Write a program that will compute the average (mean) of the four numbers. Remember that the value of the average in not necessarily an integer value. Also remember that truncation will occur if you divide an integer by an integer.

For example, if the numbers in the file were: 1 2 3 4 then the answer should be 2.5 (not 2).

Program 30: Unfilled Box

Your task is to read a pair of positive integers, L and H, from standard input. Both L and H will be less than 21. Your program then prints out an empty box of asterisks with L horizontal stars and H vertical stars. The box is empty. For example, if the input data is 6 and 4 then the output should be:

```
******
*    *
*    *
******
```

Program 31: Bubble Sort

A file contains 10 integers. Write a program to read these numbers and then sort them using a standard bubble sort. Your program should output the numbers sorting into ascending order. You should realize that the bubble sort is on Order N^2 comparison based sorting strategy. The name of the file is to be passed to your program as argv[1]. We discussed bubble sort in class.

Program 32: Inner Product of Two Vectors

Write a program that will input two vectors of information. Each vector will contain exactly 8 floating point values. Your program will enter these values in from the standard input. Once the values have been read in, your program should call a function that will compute the inner product of these two vectors. The function should return the value back to the main program. The main program will then print out the value. You should not print the value out inside the inner product function.

The inner product of two vectors is a single number obtained by multiplying corresponding elements and then adding up their sums. For example, if vector u = (5, 1, 6, 2) and vector v = (1, 2, 3, 4) then the inner product is 33 because

(5 * 1) = 5
(1 * 2) = 2
(6 * 3) = 18
(2 * 4) = 8

and 5 + 2 + 18 + 8 is 33.

Your C program will call a function called inner that has the following interface:

```
float inner (  float u[] , float v[],  int size ) ;
```

Program 33: Persistence of a Number

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 a 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".

Program 34: Simulating Call By Reference

Write a program that contains both a main function and a function called swap. Your main program should read in two integer values from the keyboard and then call the function swap. The swap function will swap the original values and then return back to main. The function *main* will then output the values contained in the original variables. The values should be swapped. No global variables are to be used. You must simulate call by reference by passing the address of the variables to the function swap. You must be able to do this.

Program 35: Passing a Two Dimensional Array

Write a program that will read in 9 integer values into a 3 x 3 two dimensional array. The values should be entered via standard input. The main program should then call a function sum that will calculate the sum of the values and return this sum back to the main program. The main program will output the sum. This program shows that you can successfully pass a two dimensional array in C. You should also pass the number of rows and the number of columns. Remember, the word argument is spelled a-r-g-u-m-e-n-t.

Prime 36: Brute Force Primes

Write a program that will read in positive integers and determine if each integer is a prime number or not. Do so the brute force way by testing for divisors between 1 and the number and if exactly two divisors are found, then the number is prime. Your main program must call a function IsPrime. The function IsPrime returns either a 0 or a 1. It does not return the number of the divisors.

In so doing, your program should read in positive integers from a file. The name of the file is given on the command line via argv[1]. You should read in the numbers until EOF is reached as you do not know the precise number of numbers in the file.

The number 1 is not prime. The number 2 is prime. Output your answer to standard output.

Problem 37: Perfect Numbers

Greek Mathematicians took a special interest in numbers that are equal to the sum of their proper divisors. A proper divisor of n is any number that evenly divides n that is less than n itself. For example, 6 is a perfect number because it is the sum of 1, 2, and 3 which are the factors of 6 that are less than 6. In other words, when you add up the factors of a number, excluding the number itself, you get the number.

Similarly, the number 28 is a perfect number because 1 + 2 + 4 + 7 + 14 add up to 28.

You are to write a program that contains a function called IsPerfect that takes an integer n and returns true (1) if n is perfect and false (0) otherwise. Your program should output all the perfect numbers between 1 and 100,000.

Problem 38: Blank Removal

A text file contains a bunch of characters. There are no tab characters within the file. Write a program that replaces two or more consecutive blanks by a single blank. For example, if the input is

Let's go to the movies.

then the output would be: Let's go to the movies.

The input from this program should come from a file whose name has been supplied via argv[1]. The output from this program should go to standard output. Note: This program came from Chapter 3 in your ANSI C text book. You should rely on the finite state machine concept to see your way through to the solution. This was discussed in class.

Program 39: Left Shift Plus

Create a program that has the following two variable definitions with initializers.

```
         int x = 1 ;
unsigned int y = 1 ;
```

Then, use the *sizeof* operator to capture the number of bytes in an int on the machine. Then multiply the number of bytes found by 8 since there are 8 bits/byte. Assign this value to an identifier named limit. Then place the following inside a for-loop ranging from 0 to limit-1.

```
x = x << 1;
y = y << 1;
printf("%d  %u\n", x, y ) ;
```

Before you enter the for-loop, print out the initial values of x and y. Make your output look readable. That is, make two nice columns with a label on each column. Are you able to explain the output? Make sure that you can explain the output.

Program 40: Greatest Common Divisor (Brute Force Method)

A file contains an unknown number of pairs of positive integer values. One pair per line. You are guaranteed that each member of the pair of numbers will be a positive integer. Your program will learn the name of the file from argv[1]. You are to write a program that will find the greatest common divisor for each pair of numbers. For this program, you should use a brute force method that will use a loop that will start at the minimum of the two numbers and count down by 1 until the first common divisor of both numbers is found.

Program 41: Greatest Common Divisor (Recursive Euclid Method)

A file contains an unknown number of pairs of positive integer values. There will be one pair per line. You are guaranteed that each member of the pair of numbers will be a positive integer. Your program will learn the name of the file from argv[1]. You are to write a program that will find the greatest common divisor for each pair of numbers. You should use a recursive solution that embodies Euclid's method.You can find the definition of this solution in your textbook on page 228. It is problem 4.17.

http://www.uml.edu/Sciences/computer-science/faculty/canning-james.aspx

Program 42: Non recursive Factorial – Use a function!

One way to express the definition of the factorial of a number is

N! is N x (N-1) x (N-2) x (N-3) x (N-4) x … x 3 x 2 x 1

Mathematicians defne 0! to be 1.

For example, 3! is 3 x 2 x 1 which is equal to 6.

Write a C program that computes and outputs to standard output the factorial of a number. Your function main should read in either the value 0 or a positive integer value from the command line and then output the factorial of that number. Your program must call a factorial function. This function cannot and should not print the answer. It should return the number back to the main function. The main function should print the answer. The factorial function should use in iterative solution, non recursive strategy. It should use a for-loop.

Program 43: Recursive Factorial

Write a complete C program that computes the factorial of a number. This problem is similar to program 42 except that the factorial function must be recursive.

Program 44: Random Numbers 2

Write a C program that will generate 10 random numbers between -19 and 19. That is a total of 39 possible numbers since 0 should be able to be generated. Define your own function, say myrand, which then calls rand.

Program 45: Malloc Up Space for One Integer

Write a C program that will malloc up space dynamically. The space should hold exactly one integer value. After the space has been reserve, you should deposit the number 6 into the space. After the number 6 has been deposited into the space you should print the value that the space currently holds(that is, 6). You need to create a variable that contains a pointer to an int.

Program 46: Malloc up Space for a 1 Dimensional Array of n Integers

Write a program that will read in a positive integer number, say n, from standard input. Then you need to create enough space in the heap via a call to *malloc* to hold an array of n integers. You then need to use the array index operators to fill the array with n integer values from entered by the user via calls to *scanf*. The index operator is also referred to as a subscript operator (ie., A[i]). After the array has been filled with numbers, your program should walk down the array and build their sum. Print out the sum.

Program 47: Twin Primes

Twin primes are two prime numbers that differ by 2. For example, 3 and 5 are twin primes, so are 101 and 103. Write a program that prints all twin primes less than 3000. Your program should call a function *IsPrime* that returns 1 if the number it receives is prime and it return 0 otherwise.

Program 48: Relatively Prime to 351

Write a program that will compute and print out all positive integers that are less than and relatively prime to the integer 351. Two numbers are relatively prime if their greatest common divisor is 1. You must call a gcd function.

Program 49: Bits of Fun

Write a C program that reads in a list of integers between -20000 and 20000 from a file named *testdata47*. Your program does not know how many integers there are in the file. Your program reads these integers one by one and after each integer is read, it will then print out the exact number of 1s that exists in the number's two's complement binary representation. You will need a loop to read in the number from the file until end of file is determined. You will need another loop – inside this loop – to process each bit with a given number. One strategy could be to test the rightmost bit. See if it is a 1 or a 0. You can do this by seeing if the number is an odd number or not. If the number is odd, then a 1 exists in the rightmost position. You can then rely upon C's right shift operator to move all of the bits to right one position. You could enter the innermost loop sizeof (int) * 8 times. Do you understand these words? Remember: The right shift operator in C does not guarantee that a 0 bit will be rolled in on the left side. You could test to see if the number is negative or not and use a left shift operator instead. The left shift operator in C always guarantees a 0 is rolled in.

If the input was:

53
32
1
44

the output would be:

4
1
1
3

Program 50: Non-polymorphic Linked List

Write a C program that reads in integers from the keyboard. Your program should then build a linked list in the heap. Each node of the list contains an integer and a pointer to the next node in the list. The final node in the list has a null pointer. Your program should insert integers into the front of the list. After end-of-file has been generated by the user, your program uses the list to print out the integers in the order in which they were typed in. That is, you need to print the tail of the list first. This can be easily done using a recursive strategy for printing. Hey, how does the user generate EOF from the keyboard? Your textbook explains this in chapter 1. Your textbook is your best friend.

Program 51: Integer Decomposition

It is known that every integer n greater than 17 may be written as the sum of three distinct integers, each greater than one, that are pairwise relatively prime; that is, no pair of the three integers have a common prime factor. Stated another way, their greatest common divisor is 1. To illustrate this fact, write a program that reads values of n from successive input lines and for each n show a decomposition of n into the sum of three integers a, b, and c by outputting a line of the form:

$$n = a + b + c$$

Your program may output any pairwise relatively prime decomposition. Furthermore the order of a, b, and c does not matter. The input integers are located in a file named testdata51. Each integer is a number between 18 and 1000 inclusive. End of file marks the end of the data stream.

Program 52: Matrix Multiplication

The first line of a file contains an integer number, say n. This number is then followed by 2n rows of integer numbers. Each row contains n integer numbers. The first n rows of numbers represent the values in a matrix A. The second n rows of numbers represent the values of the matrix B. Matrix A and matrix B are both n x n . Write a C progam that will multiply these two matrices together forming a result matrix that is also n x n. The value of n will not be greater than 50. The name of the file is in argv[1]. Your C program must call a function MatrixMult which will cause the matrix A and the matrix B to be multiplied. You must, like all of these programs, compile using the –ansi –Wall options. Your program cannot use global variables. Example data might be:

```
3
1    2   3
-2  -2  -4
5    1   0
8    1   4
2    1   3
2    1   4
```

Problem 53: Close Enough for All Practical Purposes

This problem was taken from the First Annual Northeast Regional High School Programming Contest Championship. The contest was held at Western New England College on May 13, 1986.

This program is not that difficult. For those of you who are math phobic I believe you can handle this. Remember the textbook is your best friend. It is filled with the answers you seek. You are Columbo, Ellery Queen, Agatha Cristi, and Sherlock Holmes all wrapped into one problem solving machine. C'mon, don't tell me you never heard of Columbo. You can do it. If you need to discuss this problem feel free to stop into my office. With help, you can become independent.

The square root of a number A can be computed by successive approximations using the iterative formula

$$X_{n+1} = (1/2) (X_n + A/X_n)$$

Starting with $X_1 = 1$ as an initial approximation for the square root of A a new approximation X is computed using the above formula. This new approximation, in turn, can be substituted in the formula to compute a newer approximation of X. This process can be continued until the square of the new approximation is close enough to A, that is

$$| X^2 - A | < e$$

where *e* is the prescribed degree of accuracy.

Write a program to compute the square root of a number using the procedure described above. The number and the level of accuracy will be specified as command line arguments.

If the program was launched with the following command line,

bash> a.out 24 .001

the output of your program should be a line that states something like this

The square root of 24 is 4.899.

Note: You should check out the function *fabs* in Appendix D. Are you getting away from distractions? Go to the library. Bring your mathematics book. Do all the problems in the first four chapters.

Program 54: Fibonacci Sequence (Iterative)

The Fibonacci sequence is 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

The first term is 1.
The second term is 1.
Each term thereafter is the sum of the previous two terms.

Write a C program that contains a function called Fibonacci. This function, when given a integer n >= 1, will return the nth Fibonacci number. The function should use a non-recursive, iterative strategy. The main function should prompt the user to enter numbers until EOF is generated by the user. Do not use an array in the program.

Program 55: Fibonacci Sequence (Recursive)

The Fibonacci sequence is 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

The first term is 1.
The second term is 1.
Each term thereafter is the sum of the previous two terms.

Write a C program that contains a function called Fibonacci. This function, when given a integer n >= 1, will return the nth Fibonacci number. The function should use a recursive strategy. The main function should prompt the user to enter numbers until EOF is generated by the user. Do not use an array in the program.

Program 56: Maximum Sum Subvector

A file contains a list of integers. The integers can, and are likely to, contain a mix of positive and negative values. The name of the file comes into your program via argv[1]. The first *n* numbers of the file are read by your program. The number n comes into your program via argv[2]. The number *n* will be no greater than 200,000.

For example, the file could contain the values:

-3 100 -4 -2 9 –63 -200 55

Your program will output a number that is the largest sum found in a subvector of this list. In the example above, the output would be 103 since that is the largest sum that can be formed by adding the elements of any subvector.

A subvector can be the entire list, a single member of the list, or it can be any collection of adjacent numbers. If the entire list of numbers contains negative numbers then your program should output 0.

Here is another example: 1 4 3 -4 8
The answer in this case would be 12 since the entire list of numbers forms the maximum sum.

Program 57: 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 will need to 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].

Program 58: Guardian Angles

Write a program that reads in three positive integers. The three numbers represent the lengths of the sides of a triangle. You are to decide what kind of triangle it is:

- Scalene – no sides equal
- Isosceles – exactly two equal sides
- Equilateral – all sides equal
- Right triangle – Pythagorean theorem is true.
- Invalid triangle – see hint below.

The Pythagorean Theorem can be stated as follows: the sum of the squares of any two sides is equal to the square of the third side if and only if the triangle is a right triangle.

Therefore, there are essentially 5 different triangle types: scalene, isosocles, equilateral, scalene right, and isosceles right. Your output should be the most relevant type .

To determine if the given vertices from a valid triangle you can use the *triangle inequality* (google this). The shortest distance between two points is a straight line.

Example input

3 4 5
34 17 38
3 4 10

Expected output

3 4 5 is a scalene right triangle
34 17 38is a scalene triangle
3 4 10 is not a triangle

Program 59: Goldbach's Conjecture

There is a famous mathematical conjecture called the Goldbach conjecture, that every even integer greater than 2 has the property that it is the sum of two prime integers. Write a C program that will prove that this conjecture is true for even numbers between a START value and a FINISH value. For example, if START = 700 and FINISH = 800 the initial output of your program would be:

700 = 17 + 683
702 = 11 + 691

The values of START and FINISH are to be supplied by command line arguments.

Program 60: Products of Unequal Primes

Write a C program to compute all integers between 300 and 500 that are a product of a pair of unequal primes. Write each integer to the screen on a separate line in the following form:

301 = 7 * 43
303 = 3 * 101
…

Program 61: Number Puzzle

Write a C program that will read in two lines of data. The first data line contains an integer N and the second data line contains N different positive integers. The program should arrange these integers so that every number that is less than or equal to N occurs in its proper numeric place and all other numbers are in increasing order. You may assume that N <= 20. The data should be read into your program via calls to the *fscanf* function. The name of the data is located on the command line as argv[1].

Example: If the input is

```
7
2   7   8   1   20  9  5
```

The output would be: 1 2 8 9 5 20 7

Program 62: File Comparison

A very useful program is one that is used to compare two text files to determine if they are exactly the same. Write a program to compare two text files. The names of the files are to be supplied as command line arguments. The program should write a message to the screen indicating that the files are exactly the same or that there are differences. If the files differ, the program should write to the screen the line numbers for the lines that are not the same.

Program 63: Knight Moves

This is a challenging problem for 91.101 students. It was taken from a programming competition held a number of years ago. You should first understand the backtracking solution to the Knight's Tour.

There are appearing some very strange phone pads that contain digit buttons("0" through "9") and other special buttons represented by "." . A phone pad consists of a set of horizontal rows, each containing characters representing the digit and other non-digit buttons. For example, the following is the existing phone pad:

123
456
789
.0.

For a given phone pad, you are to determine the total number of seven-digit phone numbers one can dial without repeating a digit by moving between digits using only Knight moves as in chess. No seven-digit phone number can start with the zero digit; also the phone number must only contain digits. Recall that a Knight moves in an L-shape, 1 space horizontal then 2 vertical, or 1space vertical then 2 horizontal:

```
- * - * -
* - - - *
- - K - -
* - - - *
- * - * -
```

If the input phone pad is incorrect (because of missing digits, duplicate digits, or rows of uneven length) the program should output on a single line ERROR and then exit. The program first reads an integer between one and ten that specifies the number of horizontal rows in the phone pad; thereafter, each line contains a row of the phone pad. There will never be more than 10 characters per input line.

Examples

Input	Output
4 123 456 789 .0.	32
3 1.034 56.9. .782.	15

http://www.uml.edu/Sciences/computer-science/faculty/canning-james.aspx

Program 64: Square Deal

Input for this program will be two positive integers entered from the keyboard via scanf. The first will be an odd integer N in the range of 3 to 15. N will be used as the size of a square array. The second integer will be an initial value I. Both N and I will be small enough so the I + N squared is less than 1000.

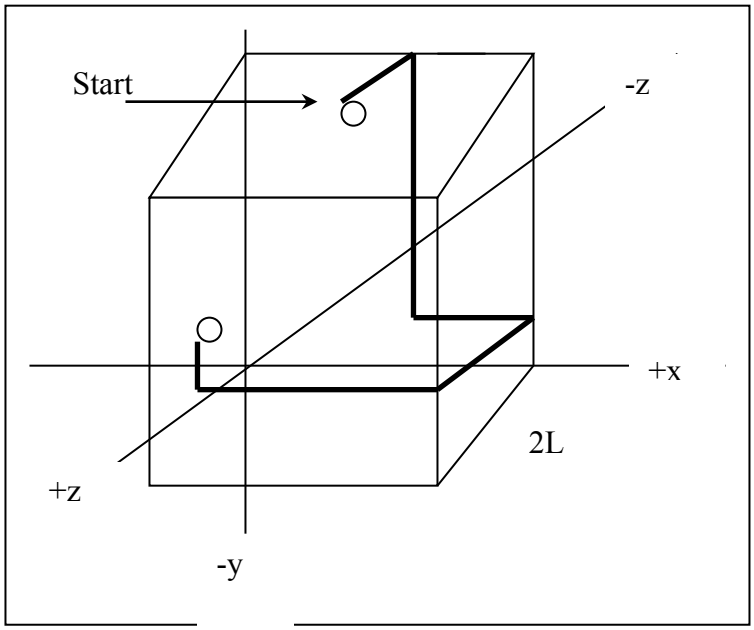
Begin in the center of an N x N array, with the integer I. If I is prime, then print the number I in that position of the square. Otherwise, print *** in that position. Move to the right one square, and test the integer I+1 for primality. Print I+1 if it is prime and *** if it is not. Proceed in a counterclockwise spiral through the square until the square is full of numbers and ***. Then print the array.

Example: If N=5 and I = 16 then your output should be:

```
***  31  ***  29  ***
***  ***  19  ***  ***
***  ***  ***  17  ***
***  ***  23  ***  ***
***  37  ***  ***  ***
```

Program 65: Cube Crawling *Taken from a Programming Competition at WPI*

Honor is ready. Her new ship Punisher is capable of interstellar flight and she wants to test its performance. In simple terms, interstellar flight is accomplished by crawling around the faces of an imaginary 3-dimensional cube of edge length 2L. Not that L is an integer parameter based on the power of a ship's engine. Locations are represented as (x, y, z) where x, y, and z are all non-negative integers.



Your ship starts at location (L, 2L, L) in the middle of the top face of this imaginary cube. In a hypercrawl, you select a direction (+/-), an axis (x, y, or z), and a distance to travel. Once this distance has been traversed, you can select a new direction and distance. If the ship crosses an edge, it will immediately wrap around the edge to exist on the face neighboring that edge and continue traveling until the total distance has been covered. In the figure, assume L is 10. The ship makes the following moves:

Move	New Location
-z 12	10 18 0
+x 34	16 18 20
+y 1	16 19 20

As the ship moves in the –z direction, it crosses an edge and wraps around to the back face, now continuing for some distance in the –y direction. When it stops, the ship is told to travel in the +x direction; this forces the ship to wrap around two cube edges, first changing direction to +z and then to direction =x, eventually coming to rest on the front face. Finally, there is a +y move. At no time can the ship stop exactly on an edge, because that would translate the ship to an alternate universe.

You must write a program for the navicomputer that will enable Punisher to determine the (x, y, z) coordinates of the final location of the ship on the cube.

INPUT
The first line will be an integer L. You can assume that L >= 4. Note that the cube has sides of length 2L. Each successive line of input will contain three values separated by spaces: a direction(+/-), and axis (x, y, or z) and a positive integer representing the distance to travel in that direction. The final input will be a single "." character.

You can assume that no move will place the ship exactly on an edge of the cube. You can assume that all directions are valid (e.g., an invalid direction from the top face is +y).

OUTPUT
Your output will be three integers, separated by spaces, representing the final location.

Sample Input	Sample Output
10	16 19 20
- z 12	
+ x 34	
+ y 1	

Program 66: Tromino Tiling

Some of you are ready to take up the challenge that this problem offers. Some of you are not. If you cannot do this problem right now, do not get overly discourage. You just need to solve a bunch of smaller problems first. If you cannot solve this problem, then let it go. Solve the ones you can solve. Perhaps you can come back to this one later.

An N x N (where N is a power of two) chessboard can be tiled with L-shaped tiles that cover three squares so that any given square and only that square is left uncovered. Assume that the rows of the board are numbered from one to N from the top down, and the columns of the board are numbered from one to N from left to right. The input will be three integers. N (a power of two, will be no more than sixteen), the row number of the square that should not be covered, and the column number of the square that should not be covered. The output should be an N x N array with the square not covered indicated with a blank and three squares covered by each tile indicated by a distinct ASCII character starting with !, which is 33 in decimal, and continuing with successive ASCII characters.

Example: For input of 8 3 6, the output could(solution not unique) be:

```
# # $ $ ( ( ) )
# " " $ ( ' ' )
% " & & *   ' +
% % & ! * * + +
- - . ! ! 2 3 3
- , . . 2 2 1 2
/ , , 0 4 1 1 5
/ / 0 0 4 4 5 5
```

If may help to think about the blank cell as being “tiled with a blank character”.

This is a classic divide and conquer. For the most part, the solution to this problem is in your textbook, but the output here is different and looks nicer.

Program 67: Order N-Squared Maximum Sum Subvector

This is the same problem as problem 54. The only difference is that you should produce a solution that is O(n log n).

Program 68: Word Reversal (medium difficulty)

Write a program that will read in strings from standard input. Each string contains no more than 60 characters. The program will output each string with the words reversed. Strings will terminate with a period. Any normal printing character can be part of the string (except a period which terminates the string). A period as the first character of the string will indicate the end of input.

Your program should continually request input from the user and then output the new string. Do not put the period at the end of your output string. Words are defined to be entities that are separated by one or more blanks.

Example input:	Example Output:
This is a sample string.	string sample a is This
This is another simple sample string.	string sample simple another is This
This, I say, is a great day!.	day! great a is say, I This,

Program 69: String Pattern Matching – Naïve Approach

Write a program that will prompt the user to enter a string of characters. There will be a maximum of 10 characters entered for the string. The program will then ask the user to enter lines of text. A line is terminated with a \n character. After each line is entered, your program should determine if the pattern exists within the line. Is the pattern a substring of the line? If it is, then you should output "Yes" and then you should identify the starting position in the line where the pattern matches. Do this for each line. The end of the output is signified by an empty line. The user hits the carriage return with entering anything. Your program should not call the strstr function. I want you to create your own strstr function. You can call it strstr.

Program 70: Walsh Transform

In class we discussed the Walsh transform. It transforms a vector of numbers into another vector of numbers. It does so by multiplying the original vector on the left by a special matrix. It is called a Hadamard matrix. A special matrix has the property that it is orthogonal. The matrix contains only 1's and -1's.

For a 2-element vector the matrix is a 2 x 2 of the form:

1 1

1 -1

We discussed how to build the 4 x 4 and 8 x 8 matrices in class. We also discussed in class how to write a divide and conquer function that will create an *n* by *n* Hadamard matrix on the fly. Write a C program that first inputs a value *n* that is the size of the problem. It then inputs five *n* element vectors and computes their Walsh Transforms. Each element of the vector should have a type *double*. Output the Walsh transform of each vector. Remember to divide by *n*. Your program should call a function called Walsh with the following interface:

```
double *walsh( double A[], int size ) ;
```

In doing this problem, make sure that you realize that the space for answer matrix must be located on the heap and not on the run time stack.

Program 71: Partition Problem

Write a program that prompts the user for a number n. This number represents the number of positive integers that will follow. Read in the n positive integer values and determine if a partitioning of the numbers exist. Numbers can be partitioned if they can be divided into two groups, say Group A and Group B and the sum of the numbers in Group A equals the sum of the numbers in Group B.

For example, the numbers 1, 4, 3, 6, 3, 5 can be partitioned into two groups: 1, 4, 6 and 3, 3, 5.

The numbers 1000, 1 ,1, 1, 1 cannot be partitioned.

Your program should never say that a partition does exist when it does not. Your program should never say that a partition does not exist when it does. If a partition does exist, then you should output one example partition. Your program should be able to input up to 100 positive integers.

This is a good problem.

An executable of my solution is called p71 and it is located in ~canning/public/101/problems. Can you make your correct program run faster than mine? How do you know that your program is correct?

Program 72: Quicksort

Implement the quicksort algorithm. Who created it? Do you understand it? We discussed it in class. Read in the first *n* numbers from a file that contains lots of integers. Sort them. The number *n* is given to your program via argv[1]. The name of file of numbers is given in argv[2]. After your program is done sorting, it should output the first number and the last number of the sorted list.

Program 73: Wallpaper

Read Chapter 1 of the Turing Omnibus. You should implement the wallpaper algorithm that is found within that chapter. The output should just be characters that you placed inside a two-dimensional array. You should then output the array.

Program 74 Maximum Sum Subvector Once Again. (N log N version)

Implement the divide and conquer (N log N) solution to the maximum sum subvector problem that was given to you in class. The pseudocode was given. Do you understand it? Can you explain it? Understanding the solution to this problem is a good thing. Try to figure it out in your head. You can do it.

Program 75: Malloc Up A Two Dimensional Array

The purpose of the program is for you to practice referencing a two dimensional array, but the space for that two dimensional array was created at run-time via a call to malloc. Realize that C's [] operator (index operator) cannot be used in a double subscripting context unless you do a work-around. In this problem you will *malloc* up enough space to hold r x c integer values via a single call to *malloc* where r is the number of rows and c represents the number of columns.

```
malloc (  r * c * sizeof (int) ) ;
```

You should read in the values of r and c via the keyboard using scanf.

In this problem, I am not asking you to create r pointers to integers. I am asking you to create space to hold r * c integers – all of which are consecutively laid out in memory. You need to visualize this.

I want you to then fill the space. I want you to use a nested for loop to fill the array. Each cell should be assigned the number 6. Thus, you need to keep track of you own offset via i*c + j within a single subscript. After you have done this, you should walk the entire array and then add up each element to form the sum. You should output the sum.

I want you to discover that you need to use a single [ ]. I do not want you to use a [][]. In fact can you? Remember, when you compile your code you should not have any warnings.

Program 76: Complex Number Module

I want you to create three files. The first file is called p76.c and it contains, among other things, the definition of the main function. The second file is called complex.h and it contains declarations only. It also contains a type definition (not a variable or a function definition). It is a struct that has two fields: one to hold the real part of a complex number and the other field will hold the imaginary part of a complex number. This was discussed and presented in class. The third file contains the code for two extern functions. They are:

```
extern Complex Add_Complex ( … )
extern Complex Multiply_Complex ( … )
```

These two functions return a Complex, that is, a struct that contains the two fields (real, im).

Your main routine should prompt the user to enter the first complex number, then it should prompt the user to enter a second complex number. Your program will then print out the addition of the two numbers followed by the multiplication of the two numbers. ANSI C permits the return of a struct.

You can do this.

Problem 77: Taxi! Taxi! *(Taking 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
```

Problem 78: State Transition Machine

Consider the following state transition diagram/table:

	0	**1**	**2**	**3**	**4**
A	B	C	D	E	A
B	C	D	E	A	B
C	D	E	A	B	C
D	E	A	B	C	D
E	A	B	C	D	E

The digits 0 through 4 are the input values and the letters A through E are the state values. Given an input I and a state s, the next state is located at row s, column i. For example, if I = 2 and s = c, then the next state is A.

Given a start state s1 and an input sequence i1, i2, i3, …, in, the final state for the sequence is calculated by locating the next state (s2) for s1, then locating the next state for (s3) for s2 and i2, then locating the next state (s4) for s3 and I3, etc. The state sn+1 is the final state.

For example, if D is the start state and the input sequence is 2, 1, 0, 4, then the final state is E.

Your task is to write a program that reads the start state, reads the input sequence, and displays the final state.

The start state and each value of the input sequence will be on a separate line. A negative number will mark the end of the input. You may assume that the input is valid.

For example, if the input to the program is:
B

12
-1
The final state is: B

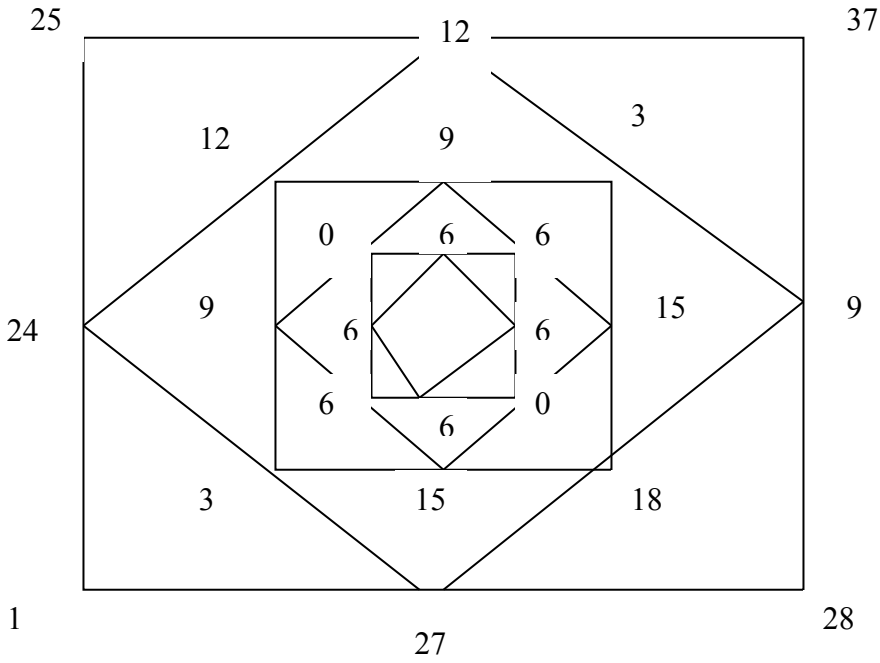
If the input to the program is:

A
4
2
0
-1

The final state is: E

Problem 79: Fun With Subtraction

The square pattern shown below was obtained by putting the numbers 37, 28, 1, and 25 at the corners of a square. By joining the midpoints of its sides, a smaller square was drawn inside the first square. Each corner of this new square was allocated a number by finding the difference between the two numbers at the ends of the line (eg. 37-28=9 ; 28-1=27). This new square was then taken as the starting point and the process was repeated until the numbers in the corners were the same, in this case, 6. The sign of the differences is to be ignored.



Your program should read in the numbers at the corners of the initial, outermost square. The input will come from the standard input. They will be provided on one line starting with the value in the upper right hand corner of the square and continuing clockwise. You are to print out the values at the corners of all the squares, including the initial square, until all the corners are all the same. You may assume that the maximum number of squares will be 20. The original four numbers will integers between 0 and 100 inclusive.

Sample Input:

37 28 1 25

Sample Output:

```
37 28  1 25
9  27 24 12
18  3 12  3
15  9  9 15
6   0  6  0
6   6  6  6
```

Problem 80: Building the Fourier Matrix

The construction of the *n* x *n* Fourier Matrix was discussed in class. The purpose of this assignment if for you to input a number *n* from the command line with *n* being a power of two and then have your program construct the *n* x *n* Fourier Matrix. In this assignment you should use your complex.h file and your complex.c file from problem 79 above. Recall that the Fourier matrix is a matrix of complex numbers. The real part is the cosine of (2*PI)/*n* * i * j. The imaginary part is the sine of (2*PI)/n * i * j.

This problem gives you an opportunity to practice building a multi-file program.

I want you to allocate your space for the *n* x *n* array using malloc. You should do this in such a way that will enable you to use double subscripts: [ ] [ ]. Recall that this will require multiple calls to the malloc function not just one call. It is important to me that you understand this.

Problem 81: Selection Sort

A file contains at most 100,000 integer numbers. The name of the file is given on the command line as argv[1]. Sort these numbers into ascending order using the Selection Sort strategy discussed in class. You should remember that selection sort is an order n-squared algorithm and it will generally outperform bubble sort. After you have sorted the numbers, your program should output the first 5 numbers. It should also output the last five numbers too.

Problem 82: Insertion Sort

A file contains at most 100,000 integer numbers. The name of the file is given on the command line as argv[1]. Sort these numbers into ascending order using the Insertion Sort strategy discussed in class. You should remember that insertion sort is an order n-squared algorithm and it will generally outperform selection sort. After you have sorted the numbers, your program should output the first 5 numbers. It should also output the last five numbers too.

Problem 83: Discrete Fourier Transform

To compute the discrete Fourier transform of a vector of length *n* requires that you multiply the vector on the left by an n x n matrix. Each element of the matrix contains a single complex number. You computed the Fourier matrix in a previous assignment.

In this transform, for now, we will not divide each resulting complex number by *n*.

In this problem you will read in 8 floating point numbers from a file. The name of this file will be given on the command line as argv[1]. Each of these values represents the real value of a single complex number. The imaginary portion of the number would be initialized to zero.

Compute the Fourier transform of the vector containing these 8 values and output the real and imaginary values of the resulting (transformed) vector.

Your program should not be completely written in a single file. It should have a main function written in main.c. It should have a complex.h and a complex.c file.

It should also have a *Makefile*.

When you submit your solution to the grader, you should submit all four files.

When you pass in your paper solution to me, you should pass in a paper copy of all four files to me.

You should create an array of structs.

You should create vectors of structs.

This is not a difficult coding assignment. You can do it.

If you need some help from me, ask.

Best wishes.

Program 84: Reverse the Last n Bits

From time to time, in the course of human events, it becomes necessary to take an integer value and reverse its last n bits. In this program, you are to complete the program below. A working version of this program can be found in ~canning/public/101/p84. You can run my copy so you can see what's what. This is likely to be a challenging problem. I do not want to discourage the average student so do not fret too much if you do not get it. I would like to see the students who are getting all the problems in a timely manner try this one. Our ability to reverse the last n bits of a number positively affects your life each and every day. The answer produced will have the uppermost bits zeroed out and the last n bits will be the reverse of the last n bits of the argument x.

```
/*******************************************************************************/
/* Bit Reverse                                                                 */
/*******************************************************************************/

#include <stdio.h>
#include <stdlib.h>

int reverse ( int x , int n ) {

  int answer = 0  ;

   /* insert additional declarators and executable lines here as you */
   /* see fit. My solution does not add too many lines.              */

  return answer ;

}

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

  int i ;

  for ( i = 0 ; i < 16 ; i++ )

     printf( "%d %d \n", i, reverse( i, atoi( argv[1] ) ) ) ;

  return 0 ;

}
```

```
0 0
1 8
2 4
3 12
4 2
5 10
6 6
7 14
8 1
9 9
10 5
11 13
12 3
13 11
14 7
15 15
```

This is an example output for bash> ./a.out 4

For example, you know that the number is 0000 0000 0000 0000 0000 0000 0000 0001. If you reverse the last 4 bits, you would get 0000 0000 0000 0000 0000 0000 0000 1000, which is the number 8.

If you ran bash> ./a.out 5 then the pair 15 30 would be part of the anwer. Why? Well, 15 is

0000 0000 0000 0000 0000 0000 0000 1111 and if you reverse the 5 bits you'd get 0000 0000 0000 0000 0000 0000 0001 1110 which is 30.

One more: If you ran bash> ./a.out 3 then the pair 15 7 would be part of the answer. Why? If you reverse the last 3 bits of 15 and zero out the uppermost bits you'd get:

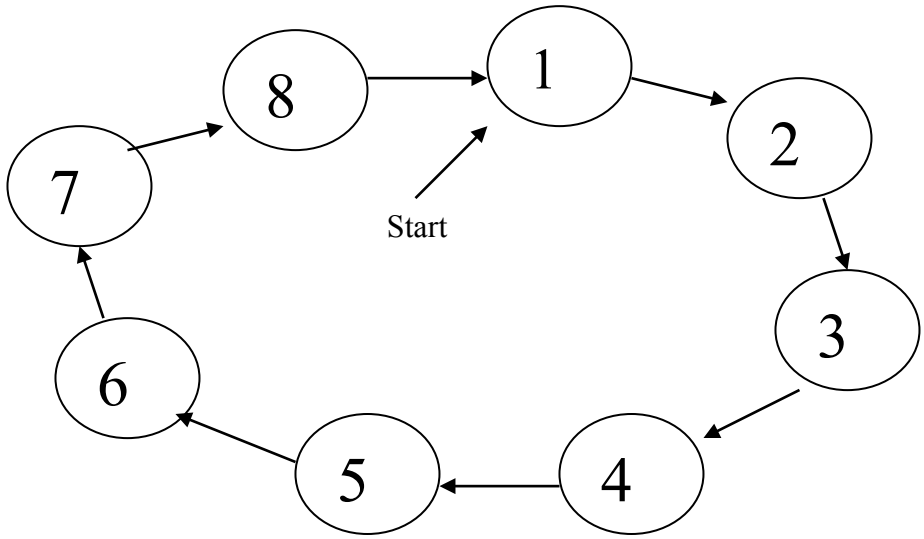
0000 0000 0000 0000 0000 0000 0000 0111

Got it?

Program 85: Alligators and Ducks

Suppose there are n ducks floating on the pond in a circle. The pond is also home for an alligator with a fondness for ducks. Beginning at the 1st position, the alligator counts around the circle and eats every mth duck (the circle closing as ducks are eaten). For example, the eaten order when n=8, m = 4 is

5 1 6 3 2 4 8 7 as shown:



The fifth duck is the first on the menu, the first duck is second, etc. Write a program which prints out the order of disappearance of the ducks given n (number of ducks) and m (inteval to the next duck to be eaten) to the screen. At least one blank must separate each number. The values of m and n are to be entered from the keyboard via standard input. You should use malloc to create the space for the m ducks.

Program 86: Shell Sort

A file contains at most 100,000 integer numbers. The name of the file is given on the command line as argv[1]. Sort these numbers into ascending order using the Shell Sort strategy discussed in class. You should remember that selection sort is an order n-squared algorithm and it will generally outperform insertion sort. After you have sorted the numbers, your program should output the first 5 numbers. It should also output the last five numbers too.

Program 87: Ackermann's Function

Go to http://mathworld.wolfram.com/AckermannFunction.html and read about Ackermann's function. Here is the function:

```
A(x,y) =     y + 1,            if  x is 0
             A(x-1,1)          if  y is 0
             A(x-1,A(x,y-1))   otherwise
```

Your job is to pass in a hardcopy solution to Ackerman's Function for small positive values for x and y. Attach to your code a nice looking table. The rows of the table represent x input values and the columns of the table represent y input values. Lable your table.

Fill in each cell of the table with two values. The first value is the value of the Ackermann function for a given x, y pair. The second value is the user time it took to compute the function using the time utility. Use the user time value. Make yoru x and y values have unsigned long type. Let your patience and/or machine size to represent an unsigned long determine how big of a table you will fill.

Start with values for x = 0, 1, 2, 3, …
Start with values of y = 0, 1, 2, 3, …

Note: There is no electronic submission for this problem, just a hardcopy submission.

Program 88: BigFib

You are to implement a program that will compute to three decimal places the frequency of a given decimal digit (0 through 9) in the decimal representation of a Fibonacci number of given index n. Recall that we start our sequence with n=1.

An example interaction would be:

Input the Fibonacci index: 100
Input the digit whose frequency is desired: 3

The digit 3 occurred with frequency: 0.095

You must be able to deal with large Fibonacci numbers. Your algorithm must be able to handle within no more than 30 seconds of CPU time (user time), numbers of the order Fibonacci(1500).

Program 89: Recursive Digit Sum

Redo the digit sum problem all over again, except this time make sure that it is a recursive solution.

Program 90: Recursive Persistence

Redo the persistence problem all over again, except this time make sure that it is a recursive solution.

Program 91: Independence Day
(We gave this question on a High School Programming Contest on November 4th, 1994)

In 1994, July 4th fell on a Monday. Write a program that allows the user to repeated enter in a year from the keyboard until EOF is reaching (control-D) and then the program will output what day of the week July 4th falls on.

Example input:

1994
1809

1963
2096

Expected Output:

Monday
Tuesday
Thursday
Wednesday

Note: You should take into account leap years. A leap year is defined as any year such that it is divisible by 4 but not by 100 unless divisible by 400. For example, 2000 is a leap year, but 1900 is not.

You may assume that the year will be in the range 1800 to 2150 inclusive.

Program 92: Frequency Distribution of Array Elements

Write a C program that reads in 20 integers into an array. Each of the integers is between -10 and 10 inclusive. It then writes to the screen on separate lines each distinct integer along with the number of times that it occurs. If a particular integer does not occur then do not write its frequency. The lines of integers should appear in descending order. For example, if you had only the following seven integers:

-7 3 3 -7 5 5 3

as data in the array, your program would output:

5 occurs 2 times
3 occurs 3 times
-7 occurs 2 times

Your program must first fill your array of with 20 integers. Next, you should process the array and output the results in the format shown above. Do not modify or rearrange the elements in the integer array once they are stored. An executable image showing you the right answer is given in

~canning/pubic/101/freq/freqdist

Program 93: Igpay Atinlay
Taken from the 1989 University of Lowell High School Invitational Programming Contest.

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:

Rule 1) 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 lankbay

Rule 2) If a word starts with a vowel, then simply append way to the word. Example: apple becomes appleway and order becomes orderway.

The letter y should be considered a consonant.

The input format of text is a series of lines from standard input. 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
…..

If it helps, no word will contain more than 10 characters. No line will contain more than 80 characters. The data will not contain any punctuation or capital letters.

Program 94: Simply Two Dimensional

The purpose of this exercise is to make sure that you can input numbers into a two dimensional array and demonstrate that you can process the data in a fairly simple manner. Input for this program is a two-dimensional array of floating point data located in a file named textfile94. The input array will contain 3 rows of data with each row containing 5 columns of data. Thus, if the input values are:

4.33	5.33	1.11	99.00	100.00
1.0	33.3	12.5	1.1	-1000.00
22.1	11.9	2.4	8.3	8.9

Then your program should output:

The average values for the three rows are:	41.95	-190.42	10.32		
The average values for the five columns are:	9.14	16.84	5.33	36.13	-297.7

Program 95: An Array of Structs

This program is not conceptually hard.

The purpose of this program is to give you a bit of practice manipulating structs. To accomplish this you need to read the appropriate sections from your book concerning arrays and structs. If you have a question, then please feel free to ask. One skill you are trying to master is the independent reading of technical material(the textbook) and then applying what you have read.

A file called textfile95 has an unknown number of NAME and AGE pairs. The maximum amount of pairs that could be in the file is 100. Each pair is found on a separate line. For example,

Shannon 44
Gauss 3
Achimedes 9
Russell 77

Using an array of structs, please read each line of data into your program, sort the data by NAME (ascending order) and print out the name and the associated AGE. At the end of the sorted list, please print out the average age of the entire group of names. For the above example data your answer would be:

Archimedes 9
Gauss 33
Russell 77
Shannon 44

The average age is 40.75 years.

Program 96: Finite Space on the Heap

Although it may sound strange, I want you to write a program that is going to crash. Please write a program that loops indefinitely and each time through the loop I want you to malloc up space for 1000 integers. The program does nothing interesting, but I want you to get a sense that malloc reserves space dynamically and you only have a finite amount of space. The general idea is something like this:

```
while( 1 ){ p = (int *) malloc( 1000 * sizeof( int ) ) ; }
```

Program 97: Recognizing a Palindrome

Write a C function with prototype declaration: `int is_palindrome( char *string )`

that returns 1 if the argument string is a palindrome and 0 if not. A palidrome string is the same character string when read backwards.

Then write a C program to test the C function `is_palidrome().` Read strings from the keyboard in response to a prompt. Identify whether or not the string is a palindrome by writing the appropriate message to the screen. The program is terminated by EOF (control-d).

Program 98: Print Thyself

I want you to create a C program that can print itself without opening a file.

If need be, you are permitted to copy a solution from the web. If you do so, please identify the link where you got your solution.

In either case, staple to your hardcopy solution a written explanation of the code. Your statement should be sufficiently clear to convince me that you understand how the code works. Your written work should be presentable. If your penmanship is poor, then please type your explanation.

There is no need to provide an electronic submission for this program. A hard copy with explanation will do.

Program 99: Logical Sorting of Strings

A file contains 20 words. One word per line. Each word has no more than 10 characters. Read all 20 words into an array. You may use the definition:

char words[20][11].

Note that each row should contain up to 11 cells since the null byte must be stored too.

Now create another array containing 20 pointers to characters. Something like:

```
char  *word_ptr[20]
```

The array of words can be sorted without exchanging the individual elements of the words array. This is done by creating an array of pointers that initially point to the corresponding array of words, element by element. (initially, word_ptr[i] = &words[i]).

A sort function takes as arguments the pointer array and a size variable. The size variable in this case will be 20. The sort function will rearrange the pointers (not the words) such that the data is sorted if accessed indirectly by following the rearranged pointers.

Your program should first print out the words in sorted order. One per line.

Then your program should print out the words in their natural stored order.

Program 100: Graham Scan for the Convex Hull

Program 101: Simple Encryption

Your program is to open and read a textfile character by character via the getc library function and convert all alphabetic characters to lower case and then left rotate the alphabetic character by 13. For example,

{A, a} → n
{B, b} → o
{M, m} → z
{N, n} → a

etc…

The name of the input file is given to the program on the command line as argv[1]. Your program should write the encrypted characters to the standard output device (screen) via the library routine putchar. All non-alphabetic characters which are read in are to be written to the output unchanged.