a. Write a C program, `dec2bin.c` to convert a base-10 number to its 32-bit binary value equivalent. You may take the base-10 number in from the command line, or you may prompt the user for the number and read in her response [your option]. Your output should be a string of binary digits which correspond to the base-10 value. For example, running the program with `dec2bin 65535` [or just `dec2bin` if asking the user] should produce the output string `00000000000000001111111111111111`. Use unsigned integers.

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

void printBinary(unsigned int num) {
    char binary[33];
    binary[32] = '\0';
    for (int i = 0; i < 32; i++) {
        binary[i] = '0';
    }
    for (int i = 31; i >= 0; i--) {
        binary[i] = (num & 1) + '0';
        num >>= 1;
    }
    printf("%s", binary);
}

int main(int argc, char *argv[]) {
    unsigned int number;
    if (argc > 0) {
        number = (unsigned int)strtoul(argv[1], NULL, 10);
        printBinary(number);
    } else if (argc == 1) {
        printf("Argument not given. ");
    }
    return 0;
}
```

b. Modify your C program from problem #1 to to make the program `dec2hex.c` which will output the 32-bit or 64-bit [8-digit or 16-digit] hexidecimal equivalent of its input. For this modification, you must also handle an optional command line argument which indicates the number of digits the output hex value will contain, either 8 or 16. This will be the second argument on the line and if it is omitted the program will default to 32. For example, running the program with `dec2hex 65535 8` should produce the output string `0x0000FFFF`,

and `dec2hex 65535 16` should result in the output string `0x000000000000FFFF`. If asking the user for input, both values should be asked for. Use unsigned integers.

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

void printHex(unsigned long num, int digits) {
    printf("0x");
    if (digits == 8) {
        printf("%08X", (unsigned int)num);
    } else {
        printf("%016lX", num);
    }
}
int main(int argc, char *argv[]) {
    if (argc == 1) {
        printf("Argument not given. ");
    } else {
        unsigned long number;
        int digits = 8;
        number = strtoul(argv[1], NULL, 10);
        if (argc > 2) {
            digits = atoi(argv[2]);
        }
        printHex(number, digits);
    }
    return 0;
}
```

c. Write a C program `timesTables.c` to output the times tables from 2 to `N`, where `N` is a user-defined number take from the command line. Output the values in a nice table, using a format specifier that will allow for enough space for the results to be neatly aligned in columns.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    int N = atoi(argv[1]);;

    printf("%5c", 0);
    for (int i = 2; i <= N; i++) {
        printf("%4d", i);
    }
    printf("\n");
    printf("%5c", 0);
    for (int i = 0; i < N-1; i++) {
```

```
        printf("====");
    }
    printf("\n");
    for (int i = 2; i <= N; i++) {
        printf("%2d |", i);
        for (int j = 2; j <= N; j++) {
            printf("%4d", i * j);
        }
        printf("\n");
    }
    return 0;
}
```

d. Write a C program `holdit.c` that times you as you hold your breath. The program must put out a short message that has instructions on what to do, which should read something like, "This program will time how long you can hold your breath. Take a deep breath, then press the 'Enter' key. When you absolutely have to exhale, press the enter key again. The duration will be displayed in minutes and seconds."
   You will need to research the way the time functions work in C.

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

int main() {
    time_t startTime, endTime;
    int seconds;
    int minutes;

    printf("This program will time how long you can hold your breath.\nTake a deep breath, then press the 'Enter' key.\nWhen you absolutely have to exhale, press the enter key again.\nThe duration will be displayed in minutes and seconds.\n");


    // Waits for new line (enter)
    getchar();


    time(&startTime);

    getchar();

    time(&endTime);

    double totalSeconds = difftime(endTime, startTime);

    minutes = (int)totalSeconds / 60;
    seconds = (int)totalSeconds % 60;
```

```
    printf("\nYou held your breath for %d minutes and %d seconds.\n",
          minutes, seconds);
    return 0;
}
```

e. Write a C program `wordcount.c` that counts the number of words in a file of text. Your program should take a file name as a command line argument. As you read the file contents, keep a count of the number of words which are separated by "whitespace". [Research what is meant by "whitespace" in the C environment.] When the file has been completely read, close the file and write out the number of words. Be sure you handle error conditions like files that don't exist or errors while reading the file. You should also be able to handle files that are in different directories from where your program resides.

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

int main(int argc, char *argv[]) {
    int wordCount = 0;
    int isReadingWord = 0;
    int currChar;

    if( argc != 2 ) {
        puts( "Exactly two command line arguments needed" );
        return 1;
     } else {
        FILE* in = fopen( argv[1], "r" );
        if( !in ) {
            printf( "File %s does not exist\n", argv[1] );
            return 2;
        }
       while ((currChar = fgetc(in)) != EOF) {
            if (isspace(currChar)) {
                if (isReadingWord) {
                    isReadingWord = 0;
                }
            } else {
                // New word
                if (!isReadingWord) {
                    wordCount++;
                    isReadingWord = 1;
                }
            }
       }
```

```
        fclose(in);
    }
    printf("%d", wordCount);
    return 0;
}
```