1. 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>

int main(int argc, char *argv[]) {
    unsigned int num;
    int i;

    if (argc == 2) {
        /* get number from command line */
        num = (unsigned int)atoi(argv[1]);
    } else {
        /* ask the user for a number */
        printf("Enter a base-10 number: ");
        scanf("%u", &num);
    }

    printf("32-bit binary: ");
    for (i = 31; i >= 0; i--) {
        unsigned int bit = (num >> i) & 1;
        printf("%u", bit);
    }
    printf("\n");

    return 0;
}
```

2. Write a C program like 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-bit. For example, running the program with `dec2hex 65535 8` should produce the output string `0x0000FFFF`, andrunning with `dec2hex 65535 16` should result in the output string `0x000000000000FFFF`. If asking the user for input, both values should be asked for using separate prompts. If getting the values from the command line, accept a space-delimited list of two numbers. Use unsigned integers for this problem. [HINT: check out the `printf()` output format specification — you'll find some help there to make things easier.]

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    unsigned long long num = 0;
    unsigned int digits = 8;

    if (argc >= 2) {
        if (sscanf(argv[1], "%llu", &num) != 1) {
            printf("Invalid number.\n");
            return 1;
        }

        if (argc >= 3) {
            if (sscanf(argv[2], "%u", &digits) != 1) {
                printf("Invalid digit count. Using 8.\n");
                digits = 8;
            }
        }

    } else {
        printf("Enter a base-10 number: ");
        if (scanf("%llu", &num) != 1) {
            printf("Invalid number.\n");
            return 1;
        }

        printf("Enter number of hex digits (8 or 16): ");
        if (scanf("%u", &digits) != 1) {
```

```
            printf("Invalid digit count. Using 8.\n");
            digits = 8;
        }
    }

    if (digits != 8 && digits != 16) {
        printf("Invalid digit count (must be 8 or 16). Using 8.\n");
        digits = 8;
    }

    if (digits == 16) {
        printf("0x%016llX\n", num);
    } else {  /* digits == 8 */
        unsigned long long masked = num & 0xFFFFFFFFULL;
        printf("0x%08llX\n", masked);
    }

    return 0;
}
```

3. 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[]) {
    if (argc != 2) {
        printf("Please enter 1 number");
        return 1;
    }
    int N = atoi(argv[1]);
    if (N < 2) {
        printf("Please enter a number greater than or equal to 2.\n");
        return 1;
    }
printf("%3c","x");
    for (int i = 1; i <= N; i++) {
        printf("%6d", i);
    }
    printf("\n\n");
    for (int i = 1; i <= 12; i++) {
```

```
        printf("%3d :", i);
        for (int j = 1; j <= N; j++) {
            printf("%6d", i * j);
        }
        printf("\n");
    }
    return 0;
}
```

4. 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 <time.h>

int main() {

    printf("This program will time how long you can hold your breath.\n");

    printf("Take a deep breath, then press the 'Enter' key.\n");

    printf("When you absolutely have to exhale, press the 'Enter' key again.\n");

    printf("The duration will be displayed in minutes and seconds.\n\n");

    printf("Press 'Enter' to start...");

    getchar();

    printf("Start time \n");

    time_t start_time = time(NULL);

    printf("Press 'Enter' when you need to exhale.\n");

    getchar();

    printf("Stop time \n");

    time_t end_time = time(NULL);
```

```
    double duration = difftime(end_time, start_time);
    int minutes = (int)duration / 60;
    int seconds = (int)duration % 60;
    printf("You held your breath for %d minute(s) and %d second(s).\n", minutes, seconds);
    return 0;
}
```

5. 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 encountered 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[]) {
        FILE *file;
        int ch;
        int word_count = 0;
        int in_word = 0;

        if (argc != 2) {
                fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
                return 1;
        }
```

```
file = fopen(argv[1], "r");
if (file == NULL) {
        fprintf(stderr, "Error: Could not open file '%s'\n", argv[1]);
        perror("Details");
        return 1;
}

while ((ch = fgetc(file)) != EOF) {
        if (isspace(ch)) {
                if (in_word) {
                        word_count++;
                        in_word = 0;
                }
        } else {
                in_word = 1;
        }
}

if (in_word) {
        word_count++;
}

if (ferror(file)) {
        fprintf(stderr, "Error: Problem reading file '%s'\n", argv[1]);
        fclose(file);
```

```
            return 1;
        }

        if (fclose(file) != 0) {
            fprintf(stderr, "Error: Problem closing file '%s'\n", argv[1]);
            return 1;
        }

        printf("Number of words in '%s': %d\n", argv[1], word_count);

        return 0;
}
```