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>

int main(void) {
  int n;
  printf("Enter a number: ");
  scanf("%d", &n);

  printf("Your number %d in binary is ", n);

  for (int i = 31; i >= 0; i--) {
     int bit = (n >> i) & 1;   // isolate bit i
     printf("%d", 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>
#include <stdlib.h>

int main(int argc, char *argv[]) {
   unsigned long long decimalValue;
   int hexDigits = 8;  // default to 32-bit (8 hex digits)
```

```
    if (argc == 1) {
        // No command-line arguments: ask user for input
        printf("Enter decimal value: ");
        scanf("%llu", &decimalValue);

        printf("Enter number of hex digits (8 or 16): ");
        scanf("%d", &hexDigits);
    } else if (argc >= 2) {
        // At least one argument: read decimal value from command line
        decimalValue = strtoull(argv[1], NULL, 10);

        // If hex digits argument exists, read it; otherwise default to 8
        if (argc >= 3)
            hexDigits = atoi(argv[2]);
    }

    // Validate hex digit input
    if (hexDigits != 8 && hexDigits != 16) {
        printf("Error: Hex digits must be 8 or 16.\n");
        return 1;
    }

    // Print in zero-padded hex (uppercase)
    printf("0x%0*llX\n", hexDigits, decimalValue);

    return 0;
}
```

3. Write a C program timesTables.cto 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("Usage: %s <N>\n", argv[0]);
        return 1;
    }

    int N = atoi(argv[1]);

    if (N < 2) {
        printf("Error: N must be >= 2\n");
```

```
        return 1;
    }

    // Print header row
    printf("    "); // spacing before header row
    for (int i = 2; i <= N; i++) {
        printf("%4d", i);
    }
    printf("\n");

    // Print table rows
    for (int i = 2; i <= N; i++) {
        printf("%4d", i); // row label
        for (int j = 2; j <= N; j++) {
            printf("%4d", 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(void) {
  printf("This program will time how long you can hold your breath.\n");
  printf("Take a deep breath, then press Enter to begin.\n");
  printf("When you absolutely have to exhale, press Enter again.\n\n");
  printf("Press Enter to start...");
  getchar(); // Wait for first Enter

  time_t start = time(NULL); // start timing

  printf("Timing... Press Enter when you must exhale.\n");
  getchar(); // Wait for second Enter
```

```
  time_t end = time(NULL); // end timing

  double elapsed = difftime(end, start); // seconds difference

  int minutes = (int)elapsed / 60;
  int seconds = (int)elapsed % 60;

  printf("\nYou 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>  // for isspace()

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

   FILE *file = fopen(argv[1], "r");
   if (file == NULL) {
      perror("Error opening file");
      return 1;
   }

   int c;
   int in_word = 0;   // flag to indicate if currently inside a word
   long word_count = 0;

   while ((c = fgetc(file)) != EOF) {
      if (isspace(c)) {
         // whitespace includes spaces, tabs, newlines, etc.
```

```
            in_word = 0;
        } else if (!in_word) {
            in_word = 1;
            word_count++;
        }
    }

    if (ferror(file)) {
        perror("Error reading file");
        fclose(file);
        return 1;
    }

    fclose(file);

    printf("Word count: %ld\n", word_count);
    return 0;
}
```