York University
Electrical Engineering and Computer Science

EECS2031: Software Tools
SU2016
Assignment #11

Chapter 22: Exercises

3. Find the error in the following program fragment and show how to fix it.

```
FILE *fp;

if (fp = fopen(filename, "r")) {
  read characters until end-of-file
}
fclose(fp);
```

The argument to fclose must be a file pointer obtained from a call of fopen. The program fragment calls fclose regardless of whether the call of fopen succeeded. The call of fclose should be moved inside the if statement:

```
FILE *fp;

if (fp = fopen(filename, "r")) {
  /* read characters until end-of-file */
  fclose(fp);
}
```

4. Show how each of the following numbers will look if displayed by `printf` with `%#012.5g` as the conversion specification:
   (a) 83.7361
   (b) 29748.6607
   (c) 1054932234.0
   (d) 0.0000235218

(a) `00000083.736`
(b) `00000029749.`
(c) `001.0549e+09`
(d) `002.3522e-05`

15. Write calls of `fseek` that perform the following file-positioning operations on a binary file whose data is arranged in 64-byte "records." Use `fp` as the file pointer in each case.
   (a) Move to the beginning of record `n`. (Assume that the first record in the file is record 0.)
   (b) Move to the beginning of the last record in the file.
   (c) Move forward one record.
   (d) Move backward two records.

(a) `fseek(fp, n * 64L, SEEK_SET);`
(b) `fseek(fp, -64L, SEEK_END);`
(c) `fseek(fp, 64L, SEEK_CUR);`
(d) `fseek(fp, -128L, SEEK_CUR);`

Chapter 22: Programming Projects

2. Write a program that converts all letters in a file to upper case. (Characters other than letters shouldn't be changed.) The program should obtain the file name from the command line and write its output to `stdout`.

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

int main(int argc, char *argv[])
{
  FILE *fp;
  int ch;

  if (argc != 2) {
    fprintf(stderr, "usage: toupper file\n");
    exit(EXIT_FAILURE);
  }

  if ((fp = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Can't open %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  while ((ch = getc(fp)) != EOF)
    putchar(toupper(ch));

  fclose(fp);
  return 0;
}
```

3. Write a program named `fcat` that "concatenates" any number of files by writing them to standard output, one after the other, with no break between files. For example, the following command will display the files `f1.c`, `f2.c`, and `f3.c` on the screen:

   ```
   fcat f1.c f2.c f3.c
   ```

   `fcat` should issue an error message if any file can't be opened. *Hint:* Since it has no more than one file open at a time, `fcat` needs only a single file pointer variable. Once it's finished with a file, `fcat` can use the same variable when it opens the next file.

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

int main(int argc, char *argv[])
{
  FILE *fp;
  int ch, i;

  if (argc < 2) {
    fprintf(stderr, "usage: fcat filename [filename ...]\n");
    exit(EXIT_FAILURE);
  }

  for (i = 1; i < argc; i++) {
    if ((fp = fopen(argv[i], "r")) == NULL) {
      fprintf(stderr, "Can't open %s\n", argv[i]);
      exit(EXIT_FAILURE);
    }
    while ((ch = getc(fp)) != EOF)
      putchar(ch);
    fclose(fp);
  }

  return 0;
}
```

9. Write a program that merges two files containing part records stored by the `inventory.c` program (see Programming Project 8). Assume that the records in each file are sorted by part number, and that we want the resulting file to be sorted as well. If both files have a part with the same number, combine the quantities stored in the records. (As a consistency check, have the program compare the part names and print an error message if they don't match.) Have the program obtain the names of the input files and the merged file from the command line.

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

#define NAME_LEN 25

struct part {
  int number;
  char name[NAME_LEN + 1];
  int on_hand;
};

int main(int argc, char *argv[])
{
  FILE *in_fp1, *in_fp2, *out_fp;
  int num_read1, num_read2;
  struct part part1, part2;

  if (argc != 4) {
    fprintf(stderr, "usage: merge infile1 infile2 outfile\n");
    exit(EXIT_FAILURE);
```

```
  }

  if ((in_fp1 = fopen(argv[1], "rb")) == NULL) {
    fprintf(stderr, "Can't open %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  if ((in_fp2 = fopen(argv[2], "rb")) == NULL) {
    fprintf(stderr, "Can't open %s\n", argv[2]);
    exit(EXIT_FAILURE);
  }

  if ((out_fp = fopen(argv[3], "wb")) == NULL) {
    fprintf(stderr, "Can't open %s\n", argv[3]);
    exit(EXIT_FAILURE);
  }

  num_read1 = fread(&part1, sizeof(struct part), 1, in_fp1);
  num_read2 = fread(&part2, sizeof(struct part), 1, in_fp2);
  while (num_read1 == 1 && num_read2 == 1)
    /* successfully read records from both files */
    if (part1.number < part2.number) {
      fwrite(&part1, sizeof(struct part), 1, out_fp);
      num_read1 = fread(&part1, sizeof(struct part), 1, in_fp1);
    } else if (part1.number > part2.number) {
      fwrite(&part2, sizeof(struct part), 1, out_fp);
      num_read2 = fread(&part2, sizeof(struct part), 1, in_fp2);
    } else {
      /* part numbers are equal */
      if (strcmp(part1.name, part2.name) != 0)
        fprintf(stderr,
                "Names don't match for part %d; using the name %s\n",
                part1.number, part1.name);
      part1.on_hand += part2.on_hand;
      fwrite(&part1, sizeof(struct part), 1, out_fp);
      num_read1 = fread(&part1, sizeof(struct part), 1, in_fp1);
      num_read2 = fread(&part2, sizeof(struct part), 1, in_fp2);
    }

  /* copy rest of file1 to output file */
  while (num_read1 == 1) {
    fwrite(&part1, sizeof(struct part), 1, out_fp);
    num_read1 = fread(&part1, sizeof(struct part), 1, in_fp1);
  }

  /* copy rest of file2 to output file */
  while (num_read2 == 1) {
    fwrite(&part2, sizeof(struct part), 1, out_fp);
    num_read2 = fread(&part2, sizeof(struct part), 1, in_fp2);
  }

  fclose(in_fp1);
  fclose(in_fp2);
  fclose(out_fp);
  return 0;
}
```