Code Examples from Lectures on 11-7_reading_and_writing_files

streamDemo.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/11-7_reading_and_writing_files/code/streamDemo.c)

An example of sending error messages to stderr and exiting with an error exit code ./streamDemo > output would print error stream out on screen, but redirect stdout to the file

Not examinable:

To check your error exit code you can type echo $?

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

int main(void) {
    int num;

    if(scanf("%d",&num) != 1){
        fprintf(stderr,"Error: Enter an int\n");
        return EXIT_FAILURE;
    }

    printf("Yay %d!\n",num);
    return EXIT_SUCCESS;
}
```

readFile.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/11-7_reading_and_writing_files/code/readFile.c)

Simple example of opening and reading the a file, line by line

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

#define INPUT_SIZE 1024

//argv is array of strings
int main(int argc, char *argv[]) {
    char input[INPUT_SIZE];

    FILE * inputStream;

    inputStream = fopen("food.txt", "r");

    if (inputStream == NULL) {
        // If fopen failed, this function, perror
        // prints out the error message from the
        // error code that fopen generated
        perror(argv[0]);
        //Or instead of perror you could use your own message
        //fprintf(stderr,"Error could not open food.txt\n");
        return EXIT_FAILURE;
    }

    while (fgets(input,INPUT_SIZE,inputStream) != NULL){
         printf("%s",input);

    }
    fclose(inputStream);
    return EXIT_SUCCESS;
}
```

writeFile.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/11-7_reading_and_writing_files/code/writeFile.c)

Simple example of file creation creates file "food.txt" containing 1 line ("mashed potato")

If there is already a file called "food.txt" it will be overwritten

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

//Try with "a"

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

    //Try "a" instead of "w" to append to the file
    //instead of writing over the original file
    //if it already exists
    //outputStream = fopen(food.txt","a");
    outputStream = fopen("anotherFile.txt", "w");

    if (outputStream == NULL) {
        perror(argv[0]);
        return EXIT_FAILURE;
    }

    fprintf(outputStream, "mashed potato\n");
    fclose(outputStream);
    return EXIT_SUCCESS;
}
```

copy.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/11-7_reading_and_writing_files/code/copy.c)

Exercise: We want to write a program that is like cp and can be run like: cp f1 f2 ./copy f1 f2

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

#define INPUT_SIZE 1024


int main(int argc, char *argv[]) {
    char input[INPUT_SIZE];
    FILE * in;
    FILE * out;

    if ( argc < 3 ){
        fprintf(stderr,"Incorrect usage\n");
        return EXIT_FAILURE;
    }

    in = fopen(argv[1], "r");
    if(in == NULL){
        perror(argv[0]);
        return EXIT_FAILURE;
    }

    out = fopen(argv[2], "w");
    if(out == NULL){
        perror(argv[0]);
        return EXIT_FAILURE;
    }

    //You could do it line by line
    //while (fgets(input,INPUT_SIZE,in) != NULL){
    //     fprintf(out,"%s",input);

    //}

    //Or character by character
    int c = fgetc(in);
    while( c != EOF ){
        fputc(c,out);
        c = fgetc(in);
    }
    fclose(in);
    fclose(out);
    return 0;
}
```

grep.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/11-7_reading_and_writing_files/code/grep.c)

We want a program that can be used like grep to search for a pattern in a file ./grep ch f.txt

Note this code assumes matched lines contain < MAX_LINE characters.

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

#define MAX_LINE 65536

int main(int argc, char *argv[]) {
    FILE *in;
    char inputText[MAX_LINE];

    in = fopen(argv[2],"r");
    if(in == NULL){
        perror(argv[0]);
        return EXIT_FAILURE;
    }
    //If we wanted to find lines starting with the search pattern we could do
    //strncmp  if(strncmp(inputText,argv[1],strlen(argv[1])) == 0){
    while(fgets(inputText,MAX_LINE,in) != NULL){
         if(strstr(inputText,argv[1]) != NULL){
             printf("%s",inputText);
         }

    }

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

bmpDemo.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/11-7_reading_and_writing_files/code/bmpDemo.c)

An advanced example of generating a bmp

To compile, run and view the finished product type

dcc -o bmpDemo bmpDemo.c

./bmpDemo

display bmpDemo.bmp

An example of creating a bitmap file with code.

We generate the bytes using a while loop.

The bitmap is 512 x 512.

The bitmap file has a header, with data that includes the size of the file and other things we do not need to worry about.

After the header, there are data bytes that represent the pixels.

Each pixel is represented by 3 bytes.

One for blue, one for green and one for red.

The first 3 bytes in the pixel data represent the bottom left corner of the file.

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

#define BYTES_PER_PIXEL 3
#define BITS_PER_PIXEL (BYTES_PER_PIXEL*8)
#define NUMBER_PLANES 1
#define PIX_PER_METRE 2835
#define MAGIC_NUMBER 0x4d42
#define NO_COMPRESSION 0
#define OFFSET 54
#define DIB_HEADER_SIZE 40
#define NUM_COLORS 0

#define SIZE 512
#define BMP_FILE "bmpDemo.bmp"

void writeHeader (FILE *file);
void generateData(FILE *file);
void writePixel(unsigned char b, unsigned char r, unsigned char g, FILE * f);

int main (void) {

   FILE *outputFile;


   outputFile = fopen(BMP_FILE, "wb");

   if ( outputFile == NULL ){
       fprintf(stderr,"Could not open file\n");
   }


   writeHeader(outputFile);

   generateData(outputFile);


   fclose(outputFile);

   return EXIT_SUCCESS;
}


// Change this function to create different images!
void generateData(FILE * f){
   unsigned char blue = 255;
   unsigned char green = 183;
   unsigned char red = 255; //OxFF

   int row = 0;
   int col = 0;
   while (row < SIZE) {
      col = 0;
      while(col < SIZE){
          if(col + row == SIZE -1){
               blue = 0;
               green = 0;
               red = 0;
          }else if( row < SIZE/2){
               blue = 0;
               green = 255;
               red = 0;
          } else {
               blue = 255;
               green = 183;
               red = 255;
          }
          writePixel(blue,green,red,f);

          col++;
      }
      row++;
   }
}

// The colour of each pixel is determined by its
// blue, green and red values.
// So each pixel needs 3 bytes of data written to the file
void writePixel(unsigned char b, unsigned char g, unsigned char r, FILE * f){
          fwrite (&b, sizeof (unsigned char), 1, f);
          fwrite (&g, sizeof (unsigned char), 1, f);
          fwrite (&r, sizeof (unsigned char), 1, f);
}
```

```
// Do not worry if you do not understand this code and what all the
// values are for. It is just writing a header in the bmp and follows
// the standard format For more info if you are interested see
// https://en.wikipedia.org/wiki/BMP_file_format

void writeHeader (FILE *file) {


   unsigned short magicNumber = MAGIC_NUMBER;
   fwrite (&magicNumber, sizeof magicNumber, 1, file);

   unsigned int fileSize = OFFSET + (SIZE * SIZE * BYTES_PER_PIXEL);
   fwrite (&fileSize, sizeof fileSize, 1, file);

   unsigned int reserved = 0;
   fwrite (&reserved, sizeof reserved, 1, file);

   unsigned int offset = OFFSET;
   fwrite (&offset, sizeof offset, 1, file);

   unsigned int dibHeaderSize = DIB_HEADER_SIZE;
   fwrite (&dibHeaderSize, sizeof dibHeaderSize, 1, file);

   unsigned int width = SIZE;
   fwrite (&width, sizeof width, 1, file);

   unsigned int height = SIZE;
   fwrite (&height, sizeof height, 1, file);

   unsigned short planes = NUMBER_PLANES;
   fwrite (&planes, sizeof planes, 1, file);

   unsigned short bitsPerPixel = BITS_PER_PIXEL;
   fwrite (&bitsPerPixel, sizeof bitsPerPixel, 1, file);

   unsigned int compression = NO_COMPRESSION;
   fwrite (&compression, sizeof compression, 1, file);

   unsigned int imageSize = (SIZE * SIZE * BYTES_PER_PIXEL);
   fwrite (&imageSize, sizeof imageSize, 1, file);

   unsigned int hResolution = PIX_PER_METRE;
   fwrite (&hResolution, sizeof hResolution, 1, file);

   unsigned int vResolution = PIX_PER_METRE;
   fwrite (&vResolution, sizeof vResolution, 1, file);

   unsigned int numColors = NUM_COLORS;
   fwrite (&numColors, sizeof numColors, 1, file);

   unsigned int importantColors = NUM_COLORS;
   fwrite (&importantColors, sizeof importantColors, 1, file);

}
```