Unit 3 Sample Problems - C Programming III

In this sample problem set, we will practice advanced concepts in the C programming language.

- Length: 1:15 minutes with discussion.
- Questions: Q1, Q3-Q6

The Preprocessor

1. [Lisonbee] Below is some partially implemented C code. Using it as a template, create macros based on the instructions given in the comments.

   Ans: [Lisonbee]

```c
// Create a macros AMOUNT_1 and AMOUNT_2, and assign them any int value.
#define AMOUNT_1 123
#define AMOUNT_2 234

// Create a macro called DIFFERENCE and assign it's value to be the
// result of AMOUNT_1 minus AMOUNT_2.
#define DIFFERENCE (AMOUNT_1 - AMOUNT_2)

// Determine if AMOUNT_1 is greater than AMOUNT_2 using DIFFERENCE (if
// DIFFERENCE is less than 0, then AMOUNT_2 is larger, vice versa).
#if ( DIFFERENCE < 0 )

    //If this condition is true, define a macro called OUTPUT and assign
    //it a value of 1.
    #define OUTPUT 1

#else

    //If this condition is false, define a macro called OUTPUT and assign
    //it a value of 0.
    #define OUTPUT 0

#endif
```

2. [Acuña] Write the final code that would be generated after the preprocessor has substituted the macros in the following code: [1 points]

```c
#define AREA_OF_TRIANGLE(b, h)  (b * h) / 2.0
int base = 3;
int height = 5;
float area = AREA_OF_TRIANGLE(base, height);

//Ans: [Acuna]

int base = 3;
int height = 5;
float area = (base * height) / 2;
```

Object-Oriented Programming

3. [Lisonbee] Implement a string ADT using a struct and six functions. At its most simple, a string is an array of characters that is terminated by a null terminator. Although having a dedicated string type is generally an OOP concept, we can still implement it in C. The struct should use variables to represent an internal data structure, and the length of the string. Create six functions: string_create, string_destroy, string_append, string_substring, string_length, and string_display. Descriptions of each are shown on the next page. NOTE: you may only use stdlib.h and stdio.h for this problem; you should include them in your header file.

   Your ADT will be structured with three files: main.c, string.h, and string.c. The first file (main.c) is provided below, while you will need to implement string.h/c. The main.c file contains testing code that uses the string you will be implementing - it indirectly shows the syntax for the functions.

   main.c

```c
#include "string.h"

int main() {
    string str, sub_str;
    str = string_create("this is a string");

    printf("String contents: ");
    string_display(str);
    printf("Length: %d\n", string_length(str));

    string_append(str, ", I think...");
    printf("String contents: ");
    string_display(str);
    printf("Length: %d\n", string_length(str));

    sub_str = string_substring(str, 18, 25);
    printf("String contents: ");
    string_display(sub_str);
    printf("Length: %d\n", string_length(sub_str));

    string_destroy(str);
        string_destroy(sub_str);
    return 0;
}
```

   Output:

```
String contents: this is a string
Length: 16
String contents: this is a string, I think...
Length: 28
String contents: I think
Length: 7
```

Ans: [Lisonbee]

string.h

```
#ifndef STRING_H
#define STRING_H

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

// STRUCT DECLARATION
typedef struct string* string;

// FUNCTION DECLARATIONS
string string_create(char* str);
void string_destroy(string str);
void string_append(string str, char* app);
string string_substring(string str, int start, int end);
int string_length(string str);
void string_display(string str);

#endif
```

string.c

```
#include "string.h"

struct string {
    char* contents;
    int length;
};

//helper function
int length(char* str) {
    int count = 0;
    while (*(str++) != 0)
        count++;
    return count;
}

string string_create(char* str) {
    string s = malloc(sizeof(string));
    s->length = length(str);
        char* s = malloc(s->length);

        //copy char array
        int i;
        for (i = 0; i < s->length; i++) {
                s[i] = str[i];
        }

    return s;
}
```

```
void string_destroy(string str) {
    free(str->contents);
    free(str);
}

void string_append(string str, char* app) {
    char* buff = malloc(str->length + length(app) + 1);
    int i;
    for (i = 0; i < str->length; i++)
        buff[i] = str->contents[i];
    for (i = 0; i < length(app); i++)
        buff[i + str->length] = app[i];
    buff[i + str->length] = '\0';
    free(str->contents);
    str->contents = buff;
    str->length = length(buff);
}

string string_substring(string str, int start, int end) {
    int size = end - start;
    char* new_str = malloc(size);
    int i;
    for (i = 0; i < size; i++)
        new_str[i] = str->contents[start++];
    new_str[i] = '\0';
    string result = malloc(sizeof(string));
    result->contents = new_str;
    result->length = length(new_str);
    return result;
}

int string_length(string str) {
    return str->length;
}

void string_display(string str) {
    printf("%s\n", str->contents);
}
```

Handling Binary Files

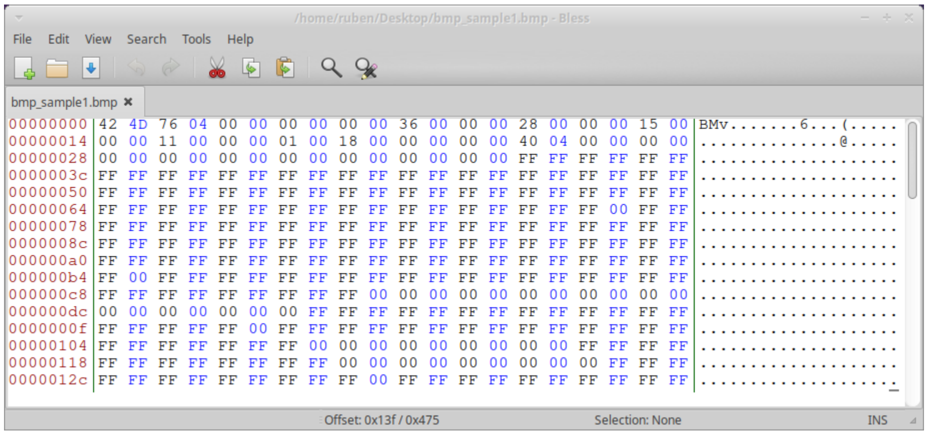
4. [Acuña] Consider the following file storage scenarios:

   (a) A word processing file that stores text, formatting information, tables, and images.
   (b) A configuration file that is meant to be edited by advance users.
   (c) An executable program file that displays a class schedule.

   Which of these should be implemented using a plain text file or a binary file format? Explain.

   Ans: [Acuña] a) Binary - it contains number heavy data like images. b) text - users should be able to edit it without needing tools. c) binary - a program only has meaning when it is binary, text doesn't make sense.

5. [Acuña] Shown below is a valid BMP file that has been opened in a hex editor. Based on the data visible, answer the following questions. Indicate which number base you use for each question. (The complete specification is shown in the appendix.) Note that this screen shot comes from an Intel architecture system where numbers are stored with little-endian byte ordering


/home/ruben/Desktop/bmp_sample1.bmp - Bless
File Edit View Search Tools Help
bmp_sample1.bmp

```
00000000 42 4D 76 04 00 00 00 00 00 00 36 00 00 00 28 00 00 00 15 00 BMv.......6...(.....
00000014 00 00 11 00 00 00 01 00 18 00 00 00 00 00 40 04 00 00 00 00 ..............@.....
00000028 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF FF FF FF ....................
0000003c FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....................
00000050 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....................
00000064 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 FF FF ....................
00000078 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....................
0000008c FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....................
000000a0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....................
000000b4 FF 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....................
000000c8 FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 ....................
000000dc 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF ....................
0000000f FF FF FF FF FF 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....................
00000104 FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 FF FF FF FF ....................
00000118 FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 FF FF FF ....................
0000012c FF FF FF FF FF FF FF FF FF 00 FF FF FF FF FF FF FF FF FF FF ....................
```

Offset: 0x13f / 0x475   Selection: None   INS

   (a) How large is the file?
   Ans: [Acuna]
   0x0476 = 1142d bytes
   (b) What is the width and height of this image?
   Ans: [Acuna]
   width = 0x15 = 21d pixels, height = 0x11 = 17d pixels
   (c) How many bits per pixel are used?
   Ans: [Acuna]
   0x18=24d bits = 3d bytes
   (d) Using b and c, how many bytes are required for each row? How many are for padding?
   Ans: [Acuna]
   Rows require 21*3+1 bytes, with +1 as padding since (21*3)%4=3. File is 64*17+54=1142 bytes.

6. [Acuña] In general, a stream of bits (a file, a download, etc) can be interpreted by any file specification. In terms of the information shown above, what suggests the correct format to use?

   Ans: [Acuña]

   The most obvious give away is the file extension: .bmp. It is conventional for the file extension to indicate the file type. Less obvious is that the first two bytes of the file contain BM, which is the 2 byte fixed signature for BMPs. Note that this is a necessary rather than sufficient condition - if a file does not begin with BM, it is not a BMP, but if it does, we are not guaranteed that it is a BMP (since other file formats could start the same way).

7. [Lisonbee] Consider a custom file format for storing experimental data sequentially (binary). Knowing only what's given below about this hypothetical file format, implement a function that can read this format (using the fread function) and return the results from a given trial of an experiment.

   - File Header (8 bytes)
     - File size (64 bits)
   - Experimental Header (5 bytes)
     - Number of experiments (16 bits)
     - Number of trials per experiment (8 bits)
     - Size of trial data (16 bits)
   - Experimental Data (2D array based on number of experiments and trials per experiment)
     - Each row begins with the experiment ID (16 bytes), followed by the data from each successive trial

Useful Types	**Description**
uint8_t	8 bit unsigned integer
uint16_t	16 bit unsigned integer
uint64_t	64 bit unsigned integer

Table 1: Useful types.

```c
uint16_t get_results(FILE* file_in, uint16_t experiment_ID, uint16_t trial) {
    uint16_t num_exp, num_trials, size, result;

    //read file header
    fread(NULL, sizeof(uint64_t), 1, file_in);

    //read experimental header
    fread(&num_exp, sizeof(uint16_t), 1, file_in);
    fread(&num_trials, sizeof(uint8_t), 1, file_in);
    fread(&size, sizeof(uint16_t), 1, file_in);

    //read experimental data
    int i, j;
    for(i = 0; i < num_exp; i++) {
        fread(&result, sizeof(uint16_t), 1, file_in);
        if(result == experiment_ID) {
            fseek(file_in, size * trial, SEEK_CUR);
            fread(&result, sizeof(uint16_t), 1, file_in);
            return result;
        }
        fseek(file_in, num_trials * size, SEEK_CUR);
    }

    return 0; //if the data isn't found
}
```

Appendix

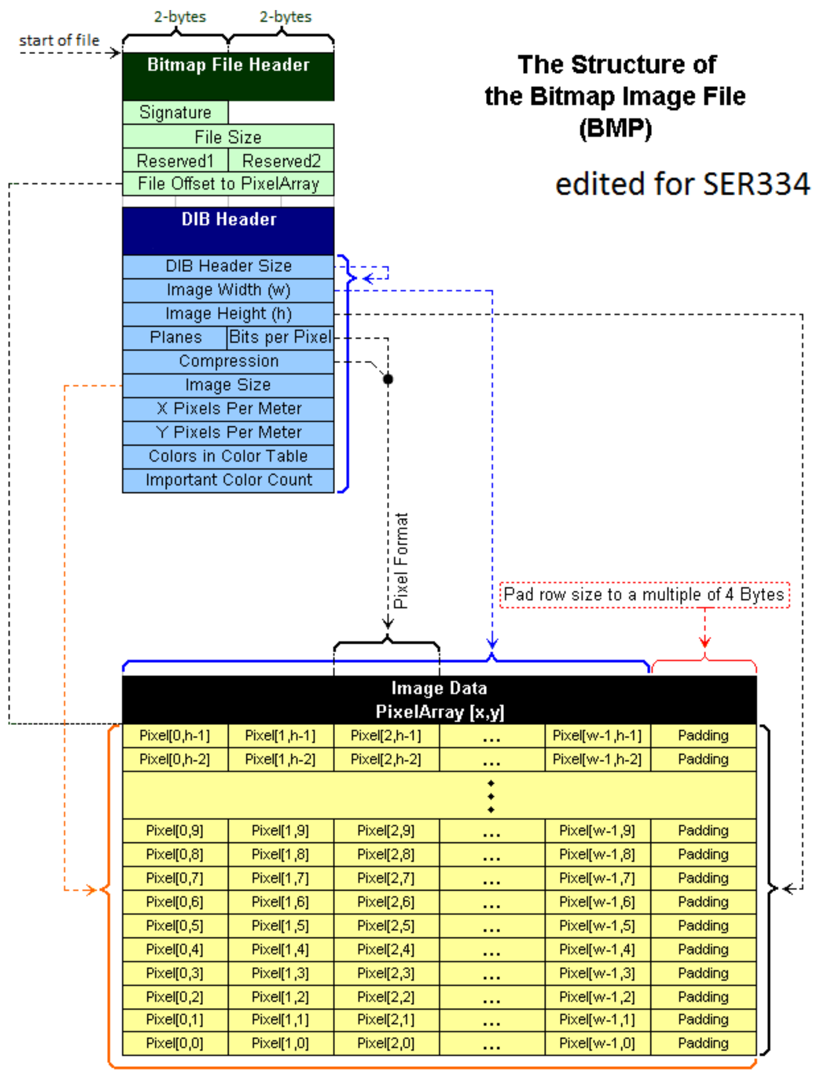

2-bytes
2-bytes
start of file
Bitmap File Header
Signature
File Size
Reserved1
Reserved2
File Offset to PixelArray
DIB Header
DIB Header Size
Image Width (w)
Image Height (h)
Planes
Bits per Pixel
Compression
Image Size
X Pixels Per Meter
Y Pixels Per Meter
Colors in Color Table
Important Color Count
The Structure of the Bitmap Image File (BMP)
edited for SER334
Pixel Format
Pad row size to a multiple of 4 Bytes
Image Data PixelArray [x,y]
Pixel[0,h-1] Pixel[1,h-1] Pixel[2,h-1] ... Pixel[w-1,h-1] Padding
Pixel[0,h-2] Pixel[1,h-2] Pixel[2,h-2] ... Pixel[w-1,h-2] Padding
Pixel[0,9] Pixel[1,9] Pixel[2,9] ... Pixel[w-1,9] Padding
Pixel[0,8] Pixel[1,8] Pixel[2,8] ... Pixel[w-1,8] Padding
Pixel[0,7] Pixel[1,7] Pixel[2,7] ... Pixel[w-1,7] Padding
Pixel[0,6] Pixel[1,6] Pixel[2,6] ... Pixel[w-1,6] Padding
Pixel[0,5] Pixel[1,5] Pixel[2,5] ... Pixel[w-1,5] Padding
Pixel[0,4] Pixel[1,4] Pixel[2,4] ... Pixel[w-1,4] Padding
Pixel[0,3] Pixel[1,3] Pixel[2,3] ... Pixel[w-1,3] Padding
Pixel[0,2] Pixel[1,2] Pixel[2,2] ... Pixel[w-1,2] Padding
Pixel[0,1] Pixel[1,1] Pixel[2,1] ... Pixel[w-1,1] Padding
Pixel[0,0] Pixel[1,0] Pixel[2,0] ... Pixel[w-1,0] Padding

Figure 1: BMP file format specification. (Modified from Wikipedia.)