Study Questions: Set No. 10
Introduction to C
Thursday November 28, 2013

Covering:

Chapter 17 and 22

1. Having to check the return value of `malloc` (or any other memory allocation function) each time we call it can be an annoyance. Write a function named `my_malloc` that serves as a ***wrapper*** for `malloc`. When we call `my_malloc` and ask it to allocate `n` bytes, it in turn calls `malloc`, tests to make sure that `malloc` doesn't return a null pointer, and then returns the pointer from `malloc`. Have `my_malloc` print an error message and terminate the program if `malloc` returns a null pointer.

2. Write a function named `duplicate` that uses dynamic storage allocation to create a copy of a string. For example, the call
   ```
   p = duplicate(str);
   ```
   would allocate space for a string of the same length as `str`, copy the contents of `str` into the new string, and return a pointer to it. Have `duplicate` return a null pointer if the memory allocation fails.

3. Write the following function:
   ```
   int *create_array(int n, int initial_value);
   ```
   The function should return a pointer to a dynamically allocated `int` array with `n` members, each of which is initialized to `initial_value`. The return value should be `NULL` if the array can't be allocated.

4. Suppose that the following declarations are in effect:
   ```
   struct point
   { int x, y;
   };

   struct rectangle
   { struct point upper_left, lower_right;
   };

   struct rectangle *p;
   ```
   Assume that we want `p` to point to a `rectangle` structure whose upper left corner is at (10, 25) and whose lower right corner is at (20, 15). Write a series of statements that allocate such a structure and initialize it as indicated.

5. Suppose that `f` and `p` are declared as follows:
   ```
   struct
   { union
     { char a, b;
       int c;
     } d;
     int e[5];
   } f, *p = &f;
   ```
   Which of the following statements are legal?
   ```
   p->b = ' '
   p->e[3] = 10;
   (*p).d.a = '*';
   p->d->c = 20;
   ```

6. The following loop is supposed to delete all nodes from a linked list and release the memory that they occupy. Unfortunately, the loop is incorrect. Explain what's wrong with it and show how to fix the bug.

```
for (p = first; p != NULL; p = p->next)
  free (p);
```

7. True or false: If `x` is a structure and `a` is a member of that structure, then `(&x)->a` is the same as `x.a`. Justify your answer.

8. Write the following function:

```
int count_occurrences(struct node *list, int n);
```

The `list` parameter points to a linked list; the function should return the number of times that `n` appears in this list. Assume that the `node` structure is as follow:

```
struct node
{
  int value;          /* data stored in the node  */
  struct node *next;  /* pointer to the next node */
};
```

9. Write the following function:

```
struct node *find_last(struct node *list, int n);
```

The `list` parameter points to a linked list. The function should return a pointer to the *last* node that contains `n`. It should return `NULL` if `n` does not appear in the list. Assume that the `node` structure is as follow:

```
struct node
{
  int value;          /* data stored in the node  */
  struct node *next;  /* pointer to the next node */
};
```

10. The following function is supposed to insert a new node into its proper place in an ordered list, returning a pointer to the first node in the modified list. Unfortunately, the function does not work correctly in all cases. Explain what is wrong with it and show how to fix it. . Assume that the `node` structure is as follow:

```
struct node
{
  int value;          /* data stored in the node  */
  struct node *next;  /* pointer to the next node */
};

struct node *insert_into_ordered_list(struct node *list,
                                      struct node *new_node)
{
  struct node *cur = list, * prev = NULL;

  while(cur->value <= new_node->value)
  {
     prev = cur;
     cur = cur->next;
  }

  prev->next = new_node;
  new_node->next = cur;
  return list;
}
```

11. Write the following function:

```
void delete_from_list(struct node **list, int n);
```

The `list` parameter is a pointer to a point to the first node in a linked list. The function should delete the *first* node that contains `n`, if any. The function must modify its argument to point to the list after the desired node has been deleted. Assume that the `node` structure is as follow:

```
struct node
{
  int value;          /* data stored in the node  */
  struct node *next;  /* pointer to the next node */
};
```

12. 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);
```

13. Which one of the following calls is *not* a valid way of reading one character from the standard input stream?

```
(a) getch()
(b) getchar()
(c) getc(stdin)
(d) fgetc(stdin)
```

14. Consider the following loop:

```
while((ch = getc(source_fp)) != EOF)
  putc(ch, dest_fp);
```

Suppose that we neglected to put parentheses around `ch = getc (source_fp)`

```
while(ch = getc(source_fp) != EOF)
  putc(ch, dest_fp);
```

Would the program compile without an error?
If so, what would the program do when it runs?

15. The following function is supposed to print the number of periods in a file. Unfortunately, it did not print the correct number of periods. Find the error in the function and show how to fix it.

```
int count_periods(const char *filename)
{
  FILE *fp;
  int n = 0;
  if((fp = fopen(filename, "r")) != NULL)
  {
    while(fgetc(fp) != EOF)
      if(fgetc(fp) == '.')
        n++;
    fclose (fp);
  }
  return n;
}
```

16. Write the following function:

```
int line_length(const char *filename, int n);
```

The function should return the length of line `n` in the text file whose name is `filename` (assuming that the first line in the file is line 1). If the line doesn't exist, the function should return 0.

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

18. Write a program named `can.open.c` that determines if files exist and can be opened for reading. Have the program obtain the file names from the command line. User may put any number of file names on the command line.

19. 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 `fi.c, f2.c,` and `f3.c` on the screen:
    ```
    fcat fl.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.

20. Write a program that counts the number of characters in a text file. Have the program obtain the file name from the command line.

21. Write a program that counts the number of words in a text file. (A "word" is any sequence of non-white-space characters). Have the program obtain the file name from the command line.

22. Write a program that counts the number of lines in a text file. Have the program obtain the file name from the command line.

23. Write a program that displays the bytes in a file as a series of hexadecimal codes, printed 20 per line. Have the program obtain the file name from the command line. Be sure to open the file in "`rb`" mode.

24. Write a program that reads a date from the command line and displays it in the following form:
    ```
    December 17, 2011
    ```
    Allow the user to enter the date as either `12-17-2011` or `12/17/2011`
    You may assume that there are no spaces in the date provided by the user. Print an error message to `stderr` stream if the date does not have one of the specified forms.
    Hints: Use `sscanf` to extract the month, day, and year from the command line argument.

25. Write a program that makes a copy of a file. Your program must use `fread` and `fwrite` to copy the file in blocks of 512 bytes. Of course, the last block may contain fewer than 512 bytes. Have the program obtain the name of the original file and the new file from the command line when the program is executed.

26. Write a program that converts a Windows text file to a UNIX text file. Have the program obtain the names of both files from the command line. *Hint:* Open the input file in "`rb`" mode and the output file in "`wb`" mode.

27. Write a program that converts a UNIX text file to a Windows text file. Have the program obtain the names of both files from the command line. *Hint:* Open the input file in "`rb`" mode and the output file in "`wb`" mode.

28. Write a program that reads a file containing a `C` code. Your program should remove any `C` comment from the code and write it to another file. The `C` comments can be in a form of `/* .... */` or `//`. Note that, if `/* .... */` or `//` appeared inside a string literal, it will not be treat as comment, but it will be treated as a part of the string literal, and hence it should not be removed. Have the program obtain the name of the input file and the output file from the command line when the program is executed.

29. Of the many techniques for compressing the contents of a file, one of the simplest and fastest is known as *run-length encoding*. This technique compresses a file by replacing sequences of identical bytes by a pair of bytes: a repetition count followed by a byte to be repeated. For example, suppose that the file to be compressed begins with the following sequence of bytes (shown in hexadecimal):
`46 6F 6F 20 62 61 72 21 21 21 20 20 20 20 20.`
The compressed file will contain the following bytes:
`01 46 02 6F 01 20 01 62 01 61 01 72 03 21 05 20.`
Run-length encoding works well if the original file contains many long sequences of identical bytes. In the worst case (a file with no repeated bytes), run-length encoding can actually double the length of the file.
(a) Write a program named `compress_file` that uses run-length encoding to compress a file. To run `compress_file`, we would use a command of the form "`compress_file` *original-file"*. `compress_file` will write the compressed version of *original-file* to *original-file.*`rl`. For example, the command
`compress_file foo.txt`
will cause `compress_file` to write a compressed version of `foo.txt` to a file named `foo.txt.rle`
(b) Write a program named `uncompress_file` that reverses the compression performed by the `compress_file` program. The `uncompress_file` command will have the form "`uncompress_file` *compressed-file"*. *compressed-file* should have the extension `.rle`. For example, the command `uncompress_file foo.txt.rle` will cause `uncompress_file` to open the file `foo.txt.rle` and write an uncompressed version of its contents to `foo.txt`. `uncompress_file` should display an error message if its command-line argument doesn't end with the `.rle` extension.

30. Write a program that merges two files containing *sorted* integer numbers. The merged output file should be sorted as well. Have the program obtain the name of the two original files and the merged file from the command line when the program is executed.

31. Write a program that reads a file containing integer numbers (in *text format*) separated by one or more whitespace characters and write them to another file in a *binary format*. Have the program obtain the name of the input file and the output file from the command line when the program is executed.

32. Write a program that reads a file containing integer numbers (in *binary format*) and write them to another file in a *text format,* separated by single *newline* character. Have the program obtain the name of the input file and the output file from the command line when the program is executed.