C Programming: dynamic data structures, patching binary files, managing debugging output

Task 0: Using valgrind output to discover bugs and memory leaks

Programs inevitably contain bugs, at least when they are still being developed. Interactive debugging using `valgrind(1)` helps locate and eliminate bugs. `valgrind` helps discovering illegal memory access even when no segmentation fault occurs (e.g. when reading the n+1 place of an array of size n). `valgrind` is extremely useful for discovering and fixing memory leaks. It will tell you which memory allocation was not freed.

To run valgrind write: `valgrind --leak-check=full [program-name] [program parameters]`

The source code of a buggy program, bubblesort.c, is provided. The program should sort numbers specified on the command line and print the sorted numbers, like this:

```
$ bubblesort 3 4 2 1
Original array: 3 4 2 1
Sorted array: 1 2 3 4
```

However, an illegal memory access causes a segmentation fault. In addition, the program has a few memory leaks. Use valgrind to find the errors and memory leaks and fix them.

Task 1: Counting Students With the Same Last Name

In this task we will take a list of students' names. For each last name in the list we will count and print the number of students with that same last name.
You can assume all lines in the file are at most 254 characters long..

Task 1a - Implement a linked list structure with a counting feature

You can use some of the code you wrote for Lab 2 in your implementation. To implement a container that holds and counts unique last names, the following data type should be used:

```
1. typedef struct entry entry;
2.
3. struct entry {
4.  entry *next;
5.  char *lastname;
6.  int count;
7. };
```

Since the number and identity of the last names is unknown initially, you should use a linked list based on the above type of entry to maintain a list of last names observed in the input and the number of times each one of them was encountered.
Please note that you will need to allocate entries and buffers to store the last names dynamically, using `malloc(3)`.
You should be implement the following functions and use them:

```
                                6 lines ...
1. entry* register_entry(entry* list, char * last_name); /* If there is an entry in the list with last_name -
    add 1 to the
2.        counter and return list. Otherwise, add a new entry to the end of the list and return list.
3.        If list is null - create a new entry and return a pointer to the entry. */
4. void foreach(entry* list, void (*handler)(entry * node)); /*For each entry in the list apply the function handler (that is given as a parameter).*/
5. void print_entry(entry * node); /*Print one entry to the standard output (e.g. "Cohen -
    2")*/
6. int list_length(entry* list); /* Return the list length */
7. void free_list(entry* list); /* Free the memory allocated by the list */
```

Testing your linked-list code

Before moving to the next section, you should test your linked list implementation.
Your test should check the functionality of the function as well as memory leaks (using valgrind).
An example for a test:

```
1. void testLL(){
2.   entry * list = register_entry(NULL, "last_name1");
3.   list = register_entry(list , "last_name1");
4.   list = register_entry(list , "last_name2");
5.   printf("list contains %d entries:\n", list_length(list));
6.   foreach(list, print_entry);
7.   free_list(list);
8. }
```

The output of the function should be:

```
list contains 2 entries:
last_name1 - 2
last_name2 - 1
```

valgrind should show 0 memory leaks / errors.

Task 1b - Count Last names

Using task 1a code, write a program that prints unique name counts. The program receives a filename as a parameter. Each line in the file is of the form "first-name last-name" (without the quotes).
Find all the last names for a given file and count the number of times a certain last name appears in the file.
For example, the following commands:

echo "Israel Israeli" > a.txt
echo "Israel Cohen" >> a.txt
echo "Israel Cohen" >> a.txt
lastnames a.txt

Should print
There are 2 different last names:
Israeli - 1
Cohen - 2

Instructions: Use fgets(3) to read lines from the file and and sscanf(3) to split first and last names

Task 2: Cracking a binary file

The purpose of this task is to get to know the `hexedit(1)` tool.

On a Wednesday morning an anonymous Google engineer decided to write the program "sum"
The program prints:
One + Six = 7

1. a. Download the file sum (using right click, save as).
   b. Set the file permissions (in order to make it executable) using
      `chmod u+x sum`
   c. Patch the file using the hexedit(1) utility, so it would instead print: `One + Two = 3`.
      Make sure it prints correctly.
2. Now change the program so it would print: `Four + Two = 6`
   What happened and why?

Deliverables:

Task 1 needs to be completed during the lab. Task 2, may be done in a completion lab.

Page last modified on 23 April 2012, 22:42 by **achiya**
powered by **CourseWiki 2.4.2** , **gzip** supported
generated in 0.00106982 sec.