



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Fall Term 2018



SYSTEMS PROGRAMMING AND COMPUTER ARCHITECTURE

Assignment 3: More C [1]

Assigned on: **3rd Oct 2018**

Due by: **16th Oct 2018, 23:59**

This assignment contains more C programming. Skeletons and source files related to this assignment (callback.c, stateoutflow0708.txt, wc.c) can be downloaded from the course web-site next to the assignment link.

1 First whitespace-separated word

Write a C program that has a function that:

- accepts a string as a parameter
- returns the first whitespace-separated word in the string (as a newly allocated string) and the size of that word

Example: “lorem ipsum” \rightarrow (“lorem”, 5)

2 Complex numbers

Implement a Complex number module:

- consisting of the following source files: complex.c, complex.h,
- includes a struct called `complex` to define a complex number $a+bi$, where a and b are doubles,
- includes functions to: add, subtract, multiply, and divide complex numbers,
- implement a test driver in test_complex.c, which contains `main()`.

3 Dynamic memory allocation

Having finished the implementation of Question 2, consider:

```

struct complex_set {
    int num_points_in_set;
    struct complex *points; // an array of struct complex
};

struct complex_set *alloc_set(struct complex c_arr[], int size);
void free_set(struct complex_set *set);

```

Now, implement `alloc_set()` and `free_set()`:

- `alloc_set()` needs to use `malloc` twice: once to allocate a new `complex_set`, and once to allocate the points field inside it,
- `free_set()` needs to use `free` twice.

Keep in mind, that `malloc` may return a null pointer on failure. Your implementation of `alloc_set` should do the same. You may want to validate your arguments (check for NULL pointers).

4 Binary search tree

Implement and test a binary search tree¹ in C:

- implement key `insert()` and `lookup()` functions
- implement it as a C module: `bst.c`, `bst.h`
- implement `test_bst.c` (contains `main()`, tests out your BST)
- don't worry about making it balanced
- as a bonus implement a key `delete()` function

5 wc

`wc` is a Unix utility that displays the count of characters, words and lines present in a file. If no file is specified it reads from the standard input. If more than one file name is specified it displays the counts for each file along with the filename. In this problem, we will be implementing `wc`. One of the ways to build a complex program is to develop it iteratively, solving one problem at a time and testing it thoroughly. For this problem, start with the shell in `wc.c` and then iteratively add the missing components.

6 File I/O

In this problem, we will be reading in formatted data and generating a report. One of the common formats for interchanging formatted data is tab delimited (`\t`) where each line corresponds to a single record. The individual fields of the record are separated by tabs. For this problem, the file `stateoutflow0708.txt` contains emigration data of people from individual states. The first row of the file contains the column headings. There are eight self explanatory fields. Your task is to read

¹http://en.wikipedia.org/wiki/Binary_search_tree

the file using `fscanf` and generate a report outlining the migration of people from Massachusetts (state code: 25) to all the other states. Use the field `Aggr AGI` to report the numbers. Also, at the end, display a total and verify it is consistent with the one shown below. An example report should look like the following:

STATE	TOTAL

"NEW_YORK"	484418
"FLORIDA"	590800
.....	

TOTAL	4609483

Make sure that the fields are aligned.

7 Function pointer

In this problem, we will use and create function that utilizes function pointers. The file `callback.c` contains an array of records consisting of a fictitious class of celebrities. Each record consists of the firstname, lastname and age of the student. Write code to do the following:

- Sort the records based on first name. To achieve this, you will be using the `qsort()` function provided by the standard library:

```
void qsort(void *base, size_t num, size_t width,
           int (*comparator)(const void *, const void *))
```

The function takes a pointer `base` to the start of the array, the number of elements `num` and size of each element `width`. In addition it takes a function pointer `comparator` that takes two arguments. The function pointed to by `comparator` is used to compare two elements within the array. Similar to `strcmp()`, it is required to return a negative quantity, zero or a positive quantity depending on whether the element pointed to by the first argument is “less” than, equal to or “greater” than the element pointed to by the second argument. You are required to write the appropriate callback function.

- Now sort the records based on last name. Write the appropriate callback function.
- The function `void apply (...)` iterates through the elements of the array calling a function for each element of the array. Write a function `isolder()` that prints the record if the age of the student is greater than 20 and does nothing otherwise.
- BONUS: To get a better grasp of C language, implement your own `mysort` function using any sorting algorithm you learned in previous semesters and replace `qsort` with `mysort`.

```
void mysort(void *base, size_t num, size_t width,
            int (*comparator)(const void *, const void *))
```

Hand In Instructions

Upload your source code to a subfolder **assignment3** of your SVN folder. Refer to Assignment 1 for instructions on using SVN.

References

- [1] Daniel Weller, Sharat Chikkerur, *6.087 Practical Programming in C*, January IAP 2010. (Massachusetts Institute of Technology: MIT OpenCourseWare), <http://ocw.mit.edu> (Accessed Nov 1, 2010). License: Creative Commons BY-NC-SA

Appendix: wc.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char* argv[])
5  {
6      FILE *fp = NULL;
7      int nfiles = --argc; /*ignore the name of the program itself*/
8      int argidx = 1; /*ignore the name of the program itself*/
9      char *currfile = "";
10     char c;
11     /*count of words,lines,characters*/
12     unsigned long nw = 0, nl = 0, nc = 0;
13
14     if (nfiles == 0)
15     {
16         fp = stdin; /*standard input*/
17         nfiles++;
18     }
19     else /*set to first*/
20     {
21         currfile = argv[argidx++];
22         fp = fopen(currfile, "r");
23     }
24     while (nfiles > 0) /*files left >0*/
25     {
26         if (fp == NULL)
27         {
28             fprintf(stderr, "Unable to open input\n");
29             exit(-1);
30         }
31         nc = nw = nl = 0;
32         while ((c = getc(fp)) != EOF)
33         {
34             /*TODO:FILL HERE
35             process the file using getc(fp)
36             */
37         }
38         /*print totals*/
39         printf("%ld %ld %ld %s\n", nl, nw, nc, currfile);
40         /*next file if exists*/
41         nfiles--;
42         if (nfiles > 0)
43         {
44             currfile = argv[argidx++];
45             fp = fopen(currfile, "r");
46         }
47     }
48     return 0;
49 }
```

Appendix: callback.c

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  struct student {
6      char fname[100];
7      char lname[100];
8      int year;
9      int age;
10 };
11
12 struct student class[] = {
13     {"Sean", "Penn", 2, 21},
14     {"Sean", "Connery", 4, 25},
15     {"Angelina", "Jolie", 3, 22},
16     {"Meryl", "Streep", 4, 29},
17     {"Robin", "Williams", 3, 32},
18     {"Bill", "Gates", 3, 17},
19     {"Jodie", "Foster", 4, 25},
20     {"John", "Travolta", 1, 17},
21     {"Isaac", "Newton", 2, 19},
22     {"Sarah", "Palin", 2, 19}
23 };
24
25 /*
26  @function compare_first_name
27  @desc      compares first name of two records.
28  */
29 int compare_first_name(const void *a, const void *b)
30 {
31     // TODO
32     return 1; /*place holder for now*/
33 }
34
35 /*
36  @function compare_last_name
37  @desc      compares last name of two records.
38  */
39 int compare_last_name(const void *a, const void *b)
40 {
41     // TODO
42     return 1; /*place holder for now*/
43 }
44
45 /*!
46  @function apply
47  @desc      applies
48  */
49 void apply(struct student *sarr, int nrec, void(*fp)(void *prec, void *arg),
50           void *arg)
51 {
```

```

52     int i = 0;
53     for (i = 0; i < nrec; i++)
54     {
55         /*callback*/
56         // TODO
57     }
58 }
59
60 /*
61  @function printrec
62  @desc      prints student record
63  */
64 void printrec(void *prec, void *arg)
65 {
66     struct student *pstud = (struct student *) prec;
67     printf("%-20s %-20s %2d %2d\n", pstud->fname, pstud->lname, pstud->year,
68           pstud->age);
69 }
70
71 /*
72  @function isolder
73  @desc      prints student record only if the student is older than *((int*)arg)
74  NOTE: use the same format as printrec
75  */
76 void isolder(void *prec, void *arg)
77 {
78     // TODO
79 }
80
81 int main()
82 {
83     int nstudents = sizeof(class) / sizeof(struct student);
84     int age;
85
86     puts("Raw records:");
87     puts("-----");
88     apply(class, nstudents, printrec, NULL);
89
90     /*sort based on first name*/
91     puts("Sorted by first name:");
92     puts("-----");
93     qsort(class, nstudents, sizeof(struct student), compare_first_name);
94     apply(class, nstudents, printrec, NULL);
95
96     /*sort based on last name*/
97     puts("Sorted by last name:");
98     puts("-----");
99     qsort(class, nstudents, sizeof(struct student), compare_last_name);
100    apply(class, nstudents, printrec, NULL);
101
102    /*print people older than 20*/
103    puts("People older than 20:");
104    puts("-----");
105    age = 20;

```

```
106     apply(class, nstudents, isolder, &age);
107     return 0;
108 }
```