Fall Term 2022

SYSTEMS PROGRAMMING AND COMPUTER ARCHITECTURE
Assignment 2: C Programming

Assigned on: **5th October 2022**
Due by: **11th October 2022 23:59**

Introduction to C Programming

This exercise consists of C programming problems.

1 Reverse an array

Write a C program that has a function that:

- accepts an array of 32-bit unsigned integers and a length
- reverses the elements of the array in place
- returns void (nothing)

Example for an array of length 4: $\boxed{5}\ \boxed{2}\ \boxed{7}\ \boxed{5} \rightarrow \boxed{5}\ \boxed{7}\ \boxed{2}\ \boxed{5}$

2 Box-and-arrow diagram

Use a box-and-arrow diagram for the following program to explain what it prints out:

```c
#include <stdio.h>

int foo(int *bar, int **baz)
{
  *bar = 5;
  *(bar+1) = 6;
  *baz = bar+2;
  return *((*baz)+1);
}

int main(int argc, char **argv)
{
  int arr[4] = {1, 2, 3, 4};
  int *ptr;
```

```
  arr[0] = foo(&(arr[0]), &ptr);
  printf("%d %d %d %d %d\n", arr[0], arr[1], arr[2], arr[3], *ptr);
  return 0;
}
```

3 Little vs. big endian

Write a C program that prints out whether the computer it is running on is little endian or big endian. (hint: pointer and casts)

4 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" → ("lorem", 5)

5 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 $a+b$i, 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()`.

6 Dynamic memory allocation

Having finished the implementation of Question 5, 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).

7 Binary search tree

Implement and test a binary search tree[1] 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

8 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.

9 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 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
```

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

Make sure that the fields are aligned.

10 Function pointers basics

Write a C program that has a function that:

- accepts a function pointer (pointing to a function with an integer return type and a single integer argument) and an additional array of integers and length of the array as arguments
- invokes the pointed-to function with each of the elements in the array as an argument
- overrides the current array element with the return value of the called function

Example: The function `comp` provided as a function pointer along with the array [-1] [3] [-27] should yield [0] [1] [0].

```
int comp(int a)
{
    if (a <= 0) return 0;
    else return 1;
}
```

11 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 *))
  ```

Compiling C Programs

The default C compiler on Ubuntu is *gcc* (as in GNU Compiler Collection). To compile a single C file into an executable and execute it, the following two commands can be used.

```
unix> gcc main.c -o main
unix> ./main
```

C is a not a memory safe language, and it is quite easy for the aspiring C hacker to introduce *undefined behavior*. For example, when an array out-of-bounds access is performed. But undefined behavior doesn't mean your program will crash. In fact, it will work most of the time (we highly recommend experimenting as it will improve your understanding). If you are not experimenting and your goal is to write correct (or at least free from undefined behavior) C, the compiler can actually help you. There are two approaches, the first one is additional compile time checks (warnings). They will slightly increase the cost of compiling, but do not change the binary generated (and therefore also not the performance). Hence it's generally a good idea to enable warnings. You can enable warnings individually, or use `-Wall` to activate all of them.

```
unix> gcc main.c -Wall -o main
unix> ./main
```

The second approach alters the program to perform additional runtime checks. They will impact the performance of your program, some even quite drastically. Hence you do not want them for release builds or benchmarking. However, for learning C and making sure your program is correct, they are a valuable resource. The following command activates a set of sanitizers that are suitable for this (and the next) assignment.

```
unix> gcc -Wall -fsanitize=address -fsanitize=undefined \
      -fstack-protector-all main.c -o main
unix> ./main
```

Unlike warnings, these sanitizers will output error messages only when the program runs and executes a code path that invokes the undefined behavior. If your program is correct, it will behave like without sanitizers. If there is a problem it will produce a lengthy description of the problem. Note that these sanitizers will never prove the absence of undefined behavior. They also might produce false warnings, especially if you write fancy or low-level code (which is unnecessary in this assignment). So interpret the results with a grain of salt.

We provide a gcc wrapper script on the course homepage that compiles your program with the suggested flags. Make sure the script is in the same folder as your source files. You have to make the script executable with `chmod +x safe_gcc`.

```
unix> ./safe_gcc main.c -o main
unix> ./main
```

Please use this script to compile your solutions. Unlike your TA, it will provide instant feedback on your code.

Hand In Instructions

Question 2 is a pen-and-paper exercise. Hand it in to your assistant during the exercise session or upload your written or scanned solution. For the rest of the problems, upload your source files to a subfolder named **assignment2** in your git repository. Refer to Assignment 1 for instructions on using git.

Appendix: wc.c

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

int main(int argc, char* argv[])
{
    FILE *fp = NULL;
    int nfiles = --argc; /*ignore the name of the program itself*/
    int argidx = 1; /*ignore the name of the program itself*/
    char *currfile = "";
    char c;
    /*count of words,lines,characters*/
    unsigned long nw = 0, nl = 0, nc = 0;

    if (nfiles == 0)
    {
        fp = stdin; /*standard input*/
        nfiles++;
    }
    else /*set to first*/
    {
        currfile = argv[argidx++];
        fp = fopen(currfile, "r");
    }
    while (nfiles > 0) /*files left >0*/
    {
        if (fp == NULL)
        {
            fprintf(stderr, "Unable to open input\n");
            exit(-1);
        }
        nc = nw = nl = 0;
        while ((c = getc(fp)) != EOF)
        {
            /*TODO:FILL HERE
             process the file using getc(fp)
             */
        }
        /*print totals*/
        printf("%ld %ld %ld %s\n", nl, nw, nc, currfile);
        /*next file if exists*/
        nfiles--;
        if (nfiles > 0)
        {
            currfile = argv[argidx++];
            fp = fopen(currfile, "r");
        }
    }
    return 0;
}
```

Appendix: callback.c

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

struct student {
    char fname[100];
    char lname[100];
    int year;
    int age;
};

struct student class[] = {
    {"Sean","Penn",2,21},
    {"Sean","Connery",4,25},
    {"Angelina","Jolie",3,22},
    {"Meryl","Streep",4,29},
    {"Robin","Williams",3,32},
    {"Bill","Gates",3,17},
    {"Jodie","Foster",4,25},
    {"John","Travolta",1,17},
    {"Isaac","Newton",2,19},
    {"Sarah","Palin",2,19}
};

/*
 @function compare_first_name
 @desc     compares first name of two records.
 */
int compare_first_name(const void *a, const void *b)
{
    // TODO
    return 1; /*place holder for now*/
}

/*
 @function compare_last_name
 @desc     compares last name of two records.
 */
int compare_last_name(const void *a, const void *b)
{
    // TODO
    return 1; /*place holder for now*/
}

/*!
 @function apply
 @desc     applies
 */
void apply(struct student *sarr, int nrec, void(*fp)(void *prec, void *arg),
       void *arg)
{
```

```
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     }
```