COMP319 Algorithms 1, Spring 2022
Homework Programming Assignment 3 (HW3)
4 problems, total 100pts

Instructor: Gil-Jin Jang   Email: gjang@knu.ac.kr
School of Electronics Engineering, Kyungpook National University

Objectives

1. Implementation of quick and heap sort algorithms on large-sized `container` objects
2. Efficient implementation of median search algorithm using partitioning and heaps

Common Requirements (same as HW2)

ID and name write your ID and name in a comment block at the beginning of the code.

Citation sorting algorithm implementations are already given in the lecture slide. Moreover, there are huge number of sort algorithm implementations that are available on the Internet. Therefore, if you make use of them, there is a high chance that your code could be almost identical to that of other students. Even if you do not refer to any other source, your code may eventually become very similar to each other because you have to implement the same algorithms. To avoid any unwanted COPY, if you have referred to any material, write the web address, name of the textbook, lecture number and page number, etc., as comments near the relevant code lines. There is no reduction in scores by using publicly available codes.

Character code use standard ASCII characters only. Special characters or language-specific symbols can take more than 1 byte, and may cause compilation errors. There is no explicit penalty for using special characters, but students should be responsible for the compilation errors due to those.

Execution time the execution time of your code can be measured by using a built-in function `clock()` defined in `time.h`. In the given template file, two static functions, `reset_timer()` and `elapsed_time_in_sec()` are given, and you can measure the time of your code by placing `reset_timer()` in the beginning and `elapsed_time_in_sec()` at the point that you want to measure the time from when the `reset_timer()` is called.

```
reset_timer();    // reset the start time
....              // statements to measure time
t = elapsed_time_in_sec();
            // time in seconds from when reset_timer() was called
```

Memory usage the amount of dynamic memory use can be measured by the provided `malloc_c` and `strdup_c` functions. The number of bytes allocated by these functions are accumulated to a static variable `used_memory` so that we should figure out how much memory is required for your algorithm given a specific input. See the provided template code for detailed implementation.

Redefined memory/string functions all the dynamic-sized arrays should be allocated by `void *malloc_c(size_t)`, and string duplication should be done by `char *strdup_c(const` char*). These functions replace the built-in malloc and strdup to count the allocated number of bytes.

Allowed memory operations Ordinary variables and fixed-sized arrays. The amount of memory for the above cases will be ignored in the measurement, because their sizes are not affected by the input size.

Allowed string functions almost all string functions except strdup are allowed, such as strlen, strcmp, strncmp, strcpy, strncpy, strcat, strncat, etc. You can also use strtok, but most of your homework assignments can be done without it. Try not to use it.

```
// ALLOWED USAGE EXAMPLES
int i, j, num_items;
char buf[1024];
char *s1, *s2;
struct container *C1, *C2;

s1 = strdup_c(buf);
s2 = (char*)malloc_c(sizeof(char)*(strlen(s1)+1));
strcpy(s2,s1);
if ( strcmp(s1, s2) == 0 ) {
   ...
}
C1 = (struct container*)malloc_c(sizeof(struct container)*num_items);
```

Unallowed functions strdup and malloc, and most memory manipulation functions such as calloc, realloc, memcpy, memccpy, memmove, wmemmove or other direct memory copy/move functions. These functions can update a large number of bytes at a single execution cycle so that memory move/copy operations should be efficiently implemented, therefore break UNIT TIME OPERATION assumption in the algorithm design.

```
// UNALLOWED USAGE EXAMPLES
s1 = strdup(buf);
C1 = (struct container*)malloc(sizeof(struct container)*num_items);

// some compilers may allow this, but do not use in your homework
char s2[num_items];
// direct memory copy (hardware acceleration may help)
memcpy(C2, C1, sizeof(struct container)*num_items);
```

Text I/O: use the given text file I/O functions,

```
struct container *read_container_arr_textfile( const char infile[], int *pN );
void write_container_arr_textfile( const char outfile[], struct container A[], int N );
```

These function are for you not to spend time in implementing file interfaces. Any modification on these functions should be clearly notified in your code with reasonable explanations (better not modify).

struct container: use the predefined functions for struct container type.

```
// set, compare, and swap the container contents
int assign_container(struct container *a, const char s[]);
int compare_container(struct container *a, struct container *b);
void swap_container(struct container *a, struct container *b, struct container *temp);

// array processing function for comparing C[i] and C[j]
int compare_container_arr(struct container C[], int i, int j);
// array processing function for swapping C[i] and C[j]
void swap_container_arr(struct container C[], int i, int j, struct container *temp);
// print the container array to the given FILE pointer (stdout for screen)
void print_container_arr( FILE *fp, struct container C[], int N )
```

1 Homework 3-1

To do:
1. Implement quick sort function for large-sized container objects.
2. Find and print the words at median-1 location (one before the median), median, and median+1 (one after the median)

File(s) to submit: `hw3-1.c`

Template source code: `template-hw3-1.c`

Sample input files: `in39.txt, in40.txt, in201.txt, in290.txt`

Sample execution output: `# cat: display the content of a text file`

```
# the input files, in__.txt: number of words, a sequence of words
# which are not sorted

$ cat in39.txt
39
akt bxou qhajctv hangyp rmbe nlrl rcohs xwjbrrz qbtgvo mqsmycy
rqj pztx jfrr krozo ojdbri migem lzxrsq hvz mawhziy yoru
hontxbo lmxate kch njbsen pmy dcsek xfjiyby ezwvb kufjcv benfvrj
qzotc sbsmer powi ahvdij gukxs yrm keuctsk reptn asw

# for the odd number of words, median is right at the center
# so for n=39, median is 20th word, median-/+1 are 19th and 21th
# see the sorted output file below

$ ./hw3-1.exe ../input/in39.txt quick39
MEDIAN-1, MEDIAN, MEDIAN+1: mawhziy migem mqsmycy
TIME: 0.00460 seconds
MEMORY USAGE: 180224 bytes

# the output file generated by your code should have a sorted list
# of all the words in the input file

$ cat quick39.txt
39
ahvdij akt asw benfvrj bxou dcsek ezwvb gukxs hangyp hontxbo
hvz jfrr kch keuctsk krozo kufjcv lmxate lzxrsq mawhziy migem
mqsmycy njbsen nlrl ojdbri pmy powi pztx qbtgvo qhajctv qzotc
rcohs reptn rmbe rqj sbsmer xfjiyby xwjbrrz yoru yrm

$ cat in40.txt
40
swjjskf gedvfis cgjz ckigvko yusu voglt kkuv fctayjn qzi oiulg
fhmkrcw mndnp hur xqi kprix pmseksl bdjui oarp oih lwr
kzr amkibt zwdy lyrniyf evtzzwp ziup pgksuhd zfgve fgdkdk kzbatcn
wsn ytjw qfzunc jospehw alktib khbuue whsc ypya bgxkv rqjzlh

# for the even number of words, for example, n=40,
# median is 20th (same as n=39), median-/+1 are 19th and 21th
# note: in general, median location is (n+1)/2

$ ./hw3-1.exe ../input/in40.txt quick40
MEDIAN-1, MEDIAN, MEDIAN+1: lwr lyrniyf mndnp
TIME: 0.00456 seconds
MEMORY USAGE: 184320 bytes
```

```
$ cat quick40.txt
40
alktib amkibt bdjui bgxkv cgjz ckigvko evtzzwp fctayjn fgdkdk fhmkrcw
gedvfis hur jospehw khbuue kkuv kprix kzbatcn kzr lwr lyrniyf
mndnp oarp oih oiulg pgksuhd pmseksl qfzunc qzi rqjzlh swjjskf
voglt whsc wsn xqi ypya ytjw yusu zfgve ziup zwdy

$ ./hw3-1.exe ../input/in201.txt quick201
MEDIAN-1, MEDIAN, MEDIAN+1: ngysgqtspt nnx nqjorjudu
TIME: 0.03556 seconds
MEMORY USAGE: 843776 bytes

$ ./hw3-1.exe ../input/in290.txt quick290
MEDIAN-1, MEDIAN, MEDIAN+1: mrracj msqi msqyo
TIME: 0.05246 seconds
MEMORY USAGE: 1208320 bytes
```

Requirements

1. The sorting functions should sort words given by `struct container*` type.

   ```
   void quick_sort_container_arr(struct container *C, int n) { ... }
   ```

   (*Note:* you may define your own functions that are used by the above one.)
2. The sorted list should be saved to a file given by the second command-line argument — already implemented in the template code.
3. Execution:

   ```
   $ ./hw3-1.exe infile sortedfile
   ```

   infile: input file, `number_of_words word word ...`

   sortedfile: list of words, sorted, ascending

Evaluation

1. *correctness:* output files should be in the sorted order. The words at the median-1, median, median+1 should be correct.
2. *execution time:* all the submitted codes are compiled and executed at the same machine, and their execution times are measured. Do not compare the absolute time. Your machine is different from the one that generated the above results.
3. *memory usage:* in the same way, the amount of used (allocated) memory will be measured.

2 Homework 3-2

To do: If we only need median and media± 1, sorting the full list of words is not necessary. Therefore, modify the quick sort algorithm to reduce the computation time (*Hint:* do recursive calls on the partitions where the median is only).

File(s) to submit: `hw3-2.c`

Template source code: `template-hw3-2.c`

Sample input files: `in39.txt, in40.txt, in201.txt, in290.txt`

Sample execution output: `$ ./hw3-2.exe ../input/in39.txt`

```
MEDIAN-1, MEDIAN, MEDIAN+1: mawhziy migem mqsmycy
TIME: 0.00344 seconds
MEMORY USAGE: 180224 bytes

$ ./hw3-2.exe ../input/in40.txt
MEDIAN-1, MEDIAN, MEDIAN+1: lwr lyrniyf mndnp
TIME: 0.00196 seconds
MEMORY USAGE: 184320 bytes

$ ./hw3-2.exe ../input/in201.txt
MEDIAN-1, MEDIAN, MEDIAN+1: ngysgqtspt nnx nqjorjudu
TIME: 0.01608 seconds
MEMORY USAGE: 843776 bytes

$ ./hw3-2.exe ../input/in290.txt
MEDIAN-1, MEDIAN, MEDIAN+1: mrracj msqi msqyo
TIME: 0.02102 seconds
MEMORY USAGE: 1208320 bytes
```

Requirements

1. The median search functions should return three container objects at median-1, median, and median+1 locations by the argument '`struct container *M3`' (3 container objects pre-allocated).

```
void quick_locate_median3_container_arr(
    struct container *M3,  /* to store median-1, median, median+1 */
    struct container *C,   /* input container list */
    int n                  /* number of container objects */
) { ... }
```

(*Note:* you may also define your own functions that are used by the above one.)

2. No saving file is required. Execution:

```
$ ./hw3-2.exe infile
```

infile: input file, `number_of_words word word ...`

Evaluation

1. *correctness:* the words at the median-1, median, median+1 should be correct.
2. *execution time:* all the submitted codes are compiled and executed at the same machine, and their execution times are measured.
3. *memory usage:* in the same way, the amount of used (allocated) memory will be measured.

3 Homework 3-3

To do: Replace quicksort in hw3-1 with heapsort

1. Implement HEAP sort function for large-sized `container` objects.
2. Find and print the words at median-1 location (one before the median), median, and median+1 (one after the median)

File(s) to submit: `hw3-3.c`

Template source code: `template-hw3-3.c`

Sample input files: `in39.txt, in40.txt, in201.txt, in290.txt`

Sample execution output: (*Note:* same as hw3-1 except execution time and memory usage)

```
$ ./hw3-3.exe ../input/in39.txt heap39.txt
MEDIAN-1, MEDIAN, MEDIAN+1: mawhziy migem mqsmycy
TIME: 0.00909 seconds
MEMORY USAGE: 176128 bytes

$ ./hw3-3.exe ../input/in40.txt heap40.txt
MEDIAN-1, MEDIAN, MEDIAN+1: lwr lyrniyf mndnp
TIME: 0.00902 seconds
MEMORY USAGE: 180224 bytes

$ ./hw3-3.exe ../input/in201.txt heap201.txt
MEDIAN-1, MEDIAN, MEDIAN+1: ngysgqtspt nnx nqjorjudu
TIME: 0.06805 seconds
MEMORY USAGE: 839680 bytes

$ ./hw3-3.exe ../input/in290.txt heap290.txt
MEDIAN-1, MEDIAN, MEDIAN+1: mrracj msqi msqyo
TIME: 0.10763 seconds
MEMORY USAGE: 1204224 bytes
```

Requirements

1. The sorting functions should sort words given by `struct container*` type.

   ```
   void heap_sort_container_arr(struct container *C, int n) { ... }
   ```

   (*Note:* you may define your own functions that are used by the above one. If your code uses heap array starting from 1, add lines to adjust the index.)
2. The sorted list should be saved to a file given by the second command-line argument — already implemented in the template code.
3. Execution:

   ```
   $ ./hw3-3.exe infile sortedfile
   ```

   infile: input file, `number_of_words word word ...`
   sortedfile: list of words, sorted, ascending

Evaluation

1. *correctness:* output files should be in the sorted order. The words at the median-1, median, median+1 should be correct.
2. *execution time:* all the submitted codes are compiled and executed at the same machine, and their execution times are measured.
3. *memory usage:* in the same way, the amount of used (allocated) memory will be measured.

4 Homework 3-4

To do: Heap is probably the best data structure for median search. We can use heap-related functions to efficiently search the median. Here is the sketch of one of the feasible algorithms:

End result: left subtree items $<$ median $<$ right subtree items

Input: unsorted list of items

1. Build a **MAX** heap with the left half items
2. Build a **MIN** heap with the left half items

Condition: We want to move the items so that all of the left subtree items $<$ those of the right. This can be validated by comparing roots only.

3. If root(L) $>$ root(R), swap those, and do **max**/**min**-heapify both subtrees
4. Repeat the above until no change
5. The median is root(L), median+1 is root(R), and the median-1 is the second maximum item in subtree L

File(s) to submit: `hw3-4.c`

Template source code: `template-hw3-4.c`

Sample input files: `in39.txt, in40.txt, in201.txt, in290.txt`

Sample execution output:
```
$ ./hw3-4.exe ../input/in39.txt
MEDIAN-1, MEDIAN, MEDIAN+1: mawhziy migem mqsmycy
TIME: 0.00557 seconds
MEMORY USAGE: 176128 bytes

$ ./hw3-4.exe ../input/in40.txt
MEDIAN-1, MEDIAN, MEDIAN+1: lwr lyrniyf mndnp
TIME: 0.00448 seconds
MEMORY USAGE: 180224 bytes

$ ./hw3-4.exe ../input/in201.txt
MEDIAN-1, MEDIAN, MEDIAN+1: ngysgqtspt nnx nqjorjudu
TIME: 0.03824 seconds
MEMORY USAGE: 839680 bytes

$ ./hw3-4.exe ../input/in290.txt
MEDIAN-1, MEDIAN, MEDIAN+1: mrracj msqi msqyo
TIME: 0.05416 seconds
MEMORY USAGE: 1204224 bytes
```

Requirements

1. The median search functions should return three container objects at median-1, median, and median+1 locations by the argument '`struct container *M3`' (3 container objects pre-allocated).

```
void quick_locate_median3_container_arr(
    struct container *M3,  /* to store median-1, median, median+1 */
    struct container *C,   /* input container list */
    int n                  /* number of container objects */
) { ... }
```

   (*Note:* you may also define your own functions that are used by the above one.)

2. You may need **max**/**min**-heapify functions and Build-**Max**/**Min**-Heap functions
3. No saving file is required. Execution:

```
$ ./hw3-4.exe infile
```

infile: input file, `number_of_words word word ...`

Evaluation

1. *correctness:* the words at the median-1, median, median+1 should be correct.
2. *execution time:* all the submitted codes are compiled and executed at the same machine, and their execution times are measured.
3. *memory usage:* in the same way, the amount of used (allocated) memory will be measured.

Evaluation Scheme

The total score of homework 1, and 10% of the whole course evaluation.

10pts basic submission score

10pts ID/name exist and are correct

40pts no compilation error and correct execution result. Will be evaluated by the unknown examples. Score deduction if not implemented properly, for example, using wrong algorithm.

10pts execution time

- 10/5/0 3 levels
- 10: extremely faster than other students
- 5: similar to other students
- 0: too slow

Most students will receive 5 points.

10pts memory usage, 3 levels (10/5/0) as well. Most students may receive 5 points. TA will check the codes and see if there is any unallowed function usages.

20pts code reading scores (requirements, etc.)

X 0 COPY makes whole scores 0. COPIED/BEING COPIED same. Leave citations as much as possible.

Submission format

Files to submit Source files only. 10pts deduction when unnecessary files are included (input files, execution files, project files, etc.)

Submission make a zip file `hw3.zip` of `hw3-1.c`, `hw3-2.c`, of `hw3-3.c` and `hw3-4.c`, and upload it to `lms.knu.ac.kr`. The LMS system changes the submitted files, and this zipping is to preserve the source file name.

Due Thursday 4/14 23:59 LMS time

Late submission Friday 4/15 09:59 LMS time, -10pts per hour