## Assignment overview

The purpose of this assignment is to provide the student with experience with pointers, structures, and dynamic memory in a C program. It will provide the student with initial experience with multiple C source files and C header files.

The goal of the exercise is to implement a program to called "input_search.c", to process and search input text.

1. Performing any word processing capability requires the program to store an indeterminate amount of data for retrieval and editing.

   This assignment will provide an exploration of what it takes to manage the input and storage of this type of data.

   The amount of data is unknown, and the program must be able to handle any volume of data. This means the program will have to allow for any amount of text.

   The program will allow for any number of lines of text and any number of words in each line of the text.

   Your program must be written using dynamically allocated memory to process anywhere from one to an unlimited number of lines of text. Each line must be able to process anywhere from one to an unlimited number of words. Therefore, using a set size array of characters or a set size array of lines is not viable. No constant values should be used in the array declarations.

   To keep this assignment easy, the user can declare the input character array of a specific set size. For example, assuming the input string array is labeled *str* then the line:

   *char str[1024];*

   is allowed.

2. The mechanism we will use to store these words and sentences will be based on specific structures. The structures must be defined using typedef in order to create variable aliases for each structure. Each structure can only contain the two elements listed in each. Do not add any other elements to these structures.

   The structure for the **word construct** is:

   (a) a pointer to a character
   (b) an integer to store the exact number of characters in this Word

   The type of **word** is a pointer to this word structure.

   The structure for the **line construct** is:

   (a) a pointer to the **word** listed above
   (b) an integer to store the exact number of words in this Line

   The type of **line** is a pointer to this line structure.

   The structure for the **para construct** (for paragraph) is:

   (a) a pointer to the **line** listed above
   (b) an integer to store the exact number of lines in this para

   The type of **para** is a pointer to this para structure.

3. To make the assignment simple, we provide the type definitions for the three structures and their constructor and destructor functions. You are NOT allowed to change the type definitions and the constructor and destructor functions. Please note that these are provided without proper comments. You should add comments and course required information.

```
// ===== type definitions should be in  a .h file, e.g. definitions.h
typedef struct {
  char *cp;
  int size;
} word_struct;
typedef word_struct* word;

typedef struct {
  word *wp;
  int size;
} line_struct;
typedef line_struct* line;

typedef struct {
  line *lp;
  int size;
} para_struct;
typedef para_struct* para;
// ===== end of type definitions
```

   Constructor and destructor functions of word.

```
// ===== this should be in word.c
word word_ini(void) {
  word w;
  w = (word) malloc(sizeof(word_struct));
  w->cp = NULL;
  w->size = 0;
  return w;
}

void word_free(word wd) {
  if (wd->cp != NULL) free(wd->cp);
  free(wd);
}
// ===== end of word.c
```

Constructor and destructor functions of line.

```
// ===== this should be in line.c
line line_ini(void) {
  line l;
  l = (line) malloc(sizeof(line_struct));
  l->wp = NULL;
  l->size = 0;
  return l;
}

void line_free(line l) {
  int i;
  for (i=0; i<l->size; i++) {
    word_free(l->wp[i]);
  }
  free(l);
}
// ===== end of line.c
```

Constructor and destructor functions of para.

```
// ===== this should be in para.c
para para_ini(void) {
  para p;
  p = (para) malloc(sizeof(para_struct));
  p->lp = NULL;
  p->size=0;
  return p;
}

void para_free(para p) {
  int i;
  for (i=0; i<p->size; i++) {
    line_free(p->lp[i]);
  }
  free(p);
}
// ===== end of para.c
```

4. To organize the .c and .h files for the three structure, we suggest that you have the following file.

| | |
|---|---|
| • definitions.h | type definitions |
| • headers.h | include definitions.h, word.h, line.h, para.h |
| • word.h | function prototypes for word |
| • word.c | functions for word |
| • line.h | function prototypes for line |
| • line.c | functions for line |
| • para.h | function prototypes for para |
| • para.c | functions for para |

5. For this assignment, you can only access variables of word, line, and para through functions provided in word.h, line.h, and para.h.

   - first implement word.c and then test that word.c is working properly.
   - second implement line.c and then test that line.c is working properly.
   - third implement para.c and then test that para.c is working properly.

   Here is a sample main() program testing word, line, and para where *word_str_cp(word, char*);* and *word_cp(word, word);* are in my word.h, *line_add(line, word);*, *line_print(line);*, and *line_reset(line);* are in my line.h, *para_add(para, line);* and *para_print(para);* are in my para.h. You can compile with command **gcc -o testing main.c word.c line.c para.c**.

```
#include <stdio.h>
#include "headers.h"

int main(void) {
  char test[]="testing";

  word w=word_ini();
  word w1=word_ini();
  line l=line_ini();
  para p=para_ini();

  word_str_cp(w, "cs2211");
  line_add(l, w);
  word_str_cp(w, test);
  line_add(l, w);

  line_print(l);
  para_add(p, l);

  line_reset(l);
  word_str_cp(w, "asn4");
  line_add(l, w);
  word_str_cp(w1, test);
  word_cp(w, w1);
  line_add(l, w);

  line_print(l);
  para_add(p, l);

  para_print(p);

  word_free(w);
  word_free(w1);
  line_free(l);
  para_free(p);

  return 0;
}
```

6. The following are in my word.h, line.h, and para.h You can decide what functions you want to use except constructor and destructor functions which are provided.

```
// === word.h
word word_ini(void);
void word_str_cp(word, char*);
void word_cp(word, word);
int word_cmp(word, word);
void word_print(word);
void word_free(word);

// === line.h
line line_ini(void);
void line_add(line, word);
void line_cp(line, line);
void line_print(line);
int line_search(line, word);
int line_word_position(line, word);
void line_reset(line);
void line_free(line);

// === para.h
para para_ini(void);
void para_add(para, line);
void para_print(para);
int para_search_print(para, word);
void para_free(para);
```

7. Your program must:

   The program prompt the user for a line of text.

   - process each input line of characters:
     - word in an input line is any consecutive non white space characters
     - breaking up each word in the line of characters (ignoring ALL white space)
     - for each word, store
       the characters of the word
       the size (number of characters) in that word
     - store each word separately in an instance of the word structure
     - add each instance of word structure in input order to an instance of line structure
   - add this instance of line structure to an instance of paragraph
   - repeat this process until the user enters an empty line (hits the enter key without text).

   After the user enters an empty prompt (hits the enter key on a blank line), the program must:

   - print each line in the order it was entered by the user.
   - for each line, print each word in the order it was entered by the user.
   - it must use the paragraph structure.

   Next, the program will prompt the user for a word to search for in the text entered.

   - if it finds the word, it will print out which line and position the word was found in.
     - it must search the entire text and report each time the word is found.
     - it must use the paragraph structure.
   - it must inform the user if it does not find the word.

- repeat this process until the user enters an empty line (hits the enter key without text).

8. Because this is of unknown size, the three structures can only use dynamic memory.

   The three structures can not have any array declarations ( i.e. array[10]; ). Also you are not allowed to use linked list in this assignment.

   To complete the assignment, you will need to use *realloc()*.

   Note: keep in mind how *realloc()* works. The added amount of dynamic memory allocated in the heap must include the memory already allocated for that pointer plus the new memory. So, if **myPointer* already had 64 bytes allocated and you needed to expand that block by 32 more bytes, then the requested memory size would be 96 (64 + 32) and not 32.

   *myPointer = (variableType *) realloc (myPointer, (96 * sizeof(variableType));*

9. Required Coding Standards

   All code is to be indented correctly. Variable names should be meaningful.

   Comments at the very beginning (top – first lines) of the .c and .h file must be:

Assuming that the above is saved in a file, run your program and redirect the file as stdin.

You may also want to test larger texts. In this case, you should create files for larger texts since typing each time from you keyboard is too time consuming.