Project 1 – C – LinkedList

For this project you will write C code to represent and work with LinkedList. The specific implementation details are up to you, but you must use the names and method signatures given in the requirements and you should make efficient use of memory. Your code should build with the -ansi -pedantic -Wall flags with gcc with no warnings.

1. Create a header file called printandcombine.h which defines the following:
    a. Any forward declarations required for your functions.
2. Create a C file called printandcombine.c that defines the following functions:
    a. char * term_to_string(term_t * term) that returns a string representation of a term..
    b. void print_linklist(node_t * curr) prints a linkedlist. You do not need to print the terms in any specific order.
    c. node_t * combine_like_terms(const node_t * current_node_ptr) returns a new linkedlistl that is the result of combining the like terms.

Main function is located in a C program called project1.c which reads a set of terms from terms.txt & stores the objects in a linkedlist and print it. Then, it prints the result of combining like terms in the linkedlist. **Please print your name at the start & end of your program's output**.

You should use good design principles, including using header/source files and deciding on which data types to use for the data members of your structs. Use whitespace, comments, and good variable names to improve readability. Your output should be neat and concise. Feel free to add additional c/h files as you find appropriate, however **you should not modify the buildlinklist.c/h files and common.h**. Include a comment with your name at the top of each source file. **Your code must compile with gcc using the -ansi and –pedantic flags.** Code that fails to compile will receive less than 50% credit *no matter how minor the error is*. I strongly recommend that you not use Visual Studio as it is difficult to ensure ANSI compliant C code. You can also use the provided make file to build your code. That is what I will be using to compile & test your code.

Submit the **zipped source code** including **all files required to compile and run** your program. Do **not** include any files in the zip file that are not required to compile your code (eg. .o, .exe, swapfiles, backups, etc.). Your submission should include *at least* the following files:

- project1.c
- common.h
- buildlinklist.c / buildlinklist.h
- printandcombine.c / printandcombine.h
- Any other C files you created (optional)
- The provided makefile

In addition to your zipped source, **fill in the end of this document & submit a PDF separately from your code**. Be prepared to demo your programs in class on the due date. **Start early!** If you submit early you can receive feedback and resubmit for a higher grade.

Sample Output (user input is bold):

```
$ make clean
rm -f *.o *~ project1.exe

$ make
gcc  -Wall -pedantic -ansi -c buildlinklist.c
gcc  -Wall -pedantic -ansi -c printandcombine.c
gcc  -Wall -pedantic -ansi project1.c buildlinklist.o printandcombine.o -o project1

$ ./project2.exe
NAME: SAMPLE OUTPUT
Original: : 1 + 5x + 3x^2 + 0 + 6x^2 + 2x + 7x^3 + 3x
Combined: : 7x^3 + 9x^2 + 10x + 1
NAME: SAMPLE OUTPUT
```

Briefly describe your implementation in the box below:

Term_to_string checks if the coefficient is 0 and returns a 0 if that is the case, returns a 1 if the exponent is 0, otherwise returns the coefficient, variable, a ^ and then the exponent. Print_linklist goes through the whole list and calls the term_to_string in order to print them. Combine_like_terms find the max exponent then goes through a for loop and finds all coefficients to terms with each exponent, adds those coefficients and makes one term for each exponent. It then returns a list with all new nodes using the newly created terms.

Briefly describe any issues you ran into (if any) with your implementation in the box below:

I ran into tons of issues while working on this project. Starting with the toString. I kept getting errors and trying different things until eventually I got something that works. I then began working on the combine_like_terms and I kept getting segmatation faults. After days upon days of working on it I realized there was something in my print_link_list method that was causing the term to be NULL. I then rewrote it as simply as possible and that solved all of my issues.

Copy the command you used to compile your code *without formatting* in the box below:

```
 make all
./printandcombine
```

Copy your output *without formatting* in the box below:

```
 gatlinfarrington@Gatlins-MacBook-Pro C Project % ./project1
NAME: SAMPLE OUTPUT
Original: 3x + 7x^3 + 2x + 6x^2 + 0 + 3x^2 + 5x + 1 +
Combined: : 7x^3 + 9x^2 + 10x + 1 +
NAME: SAMPLE OUTPUT
```

Copy your *entire* printandcombine.h file *without formatting* in the box below:

```
 //
//  printandcombine.h
//
//
//  Created by Gatlin Farrington on 6/30/21.
//

#ifndef printandcombine_h
#define printandcombine_h

#include <stdio.h>

//takes the nodes in the list and combines terms with like exponents
node_t * combine_like_terms(const node_t * current_node_ptr);
```

```
//converts a term to a string
//char * term_to_string(term_t * term);
char * term_to_string(term_t * term);

//prints the list given the starting node
void print_linklist(node_t * curr);



#endif /* printandcombine_h */
```

Copy your term_to_string () method *without formatting* in the box below:

```
 char * term_to_string(term_t * term) {
      char * ptr = NULL;
      ptr = malloc(sizeof(char));
    if(term-> coefficient == 0){
        sprintf(ptr, "%d", 0);
    }else if (term ->exponent == 0) {
          sprintf(ptr, "%d", term->coefficient);
      }else if (term->exponent == 1) {
          sprintf(ptr, "%d%c", term->coefficient, term->var);
      }else {
          sprintf(ptr, "%d%c^%d", term->coefficient, term->var, term->exponent);
      }
    return ptr;
}
```

Copy your print_linklist() method *without formatting* in the box below:

```
 void print_linklist(node_t *curr){

    node_t *nodeCurrent = curr;

    do{
        printf("%s + ", term_to_string(nodeCurrent->term));
    }while((nodeCurrent = nodeCurrent->next_node));

}
```

Copy your combine_like_terms () method *without formatting* in the box below:

```
 node_t * combine_like_terms(const node_t * current_node_ptr) {
    if(current_node_ptr == NULL){
        return NULL;
    }
    const node_t* it = current_node_ptr;
    node_t* combined = NULL;
    //find highest power
    int maxexponent = 0;
    term_t* TERMY = it->term;
    do{
            term_t *t = it->term;

            maxexponent = MAX(maxexponent, t->exponent);

    }while((it = it->next_node));

    maxexponent++;
    int sumOfCoef = 0;
    for(int j = 0; j<maxexponent; j++){
```

```
        it = current_node_ptr;
        //for each term check if exp is equal
                do{
                    term_t* x = it->term; //OR HERE
                    if(x==NULL){
                        continue;
                    }
                    if(x->exponent == j){
                       sumOfCoef += x->coefficient;
                    }
                }while((it = it->next_node));
        //make a new term with sumOfCoef and exp from the for loop
        term_t* termish = malloc(sizeof(term_t));
        termish->coefficient = sumOfCoef;
        termish->var = 'x';
        termish->exponent = j;
        //add the new node to the list
        add_node(&combined, termish);
        //reset sumOfCoef for the next time through the loop
        sumOfCoef = 0;
    }
    //return the node that has all following it
    return combined;
}
```