

**Project 1: Implementation of Linked Lists**  
**KECE208 Data Structures and Algorithms**  
**Fall, 2020**

Instructor: **Hwangnam Kim**  
TA: **Hyeontae Joo, Sangmin Lee**

**GOAL**

In project1, you will be assigned to use your knowledge with Linked List and Stack in order to fill in the functions that are provided at project files. Students are required to fill in empty functions in order to get outputs and get grades on those outputs. In this project, you are allowed to use “struct linked\_list” which is the list structure for this project that contains head, tail and size of the list. Also, you will be given “struct linked\_node,” which contains prev, next, and element of the node. Please refer to ‘linked\_list.h’ for more detail.

**Given Codes:**

**linked\_list.h**

**linked\_list.c**

**project1\_main.c**

You are responsible for completing implementation of ‘linked\_list.c’ in order to make the program run properly. You are not allowed to modify ‘linked\_list.h’ and ‘project1\_main.c’.

- **linked\_list.h:** This is the header file of linkLQS.c that declares all the functions and values that are going to be used in linkLQS.c. You do not have to add any more functions or variables in order to complete the project and you are NOT ALLOWED to.
- **linked\_list.c:** This contains implementations and definitions of functions and variables and you are required to fill some of the functions that contain comments “Your code starts from here.” By filling and completing implementation of those functions, you will get proper output for this project.
- **project1\_main.c:** This file is provided to students in order to debug whether their codes work correctly or not. The usage of this class will be explained later during project1 guide lecture. Time and location will be announced shortly.
- **Makefile:** This file is supposed to automatically compile the c files listed within this file. If students type “make”, this will compile all the code. But when students type “make clean”, this will erase executable code and force the user to recompile later.
- **project1.exe:** This is an executable code and provides right results of commands. Students must be having same output as this executable file generates. In order to execute this file, just type “./project1.exe”

**Structure of the linked list and the linked node:**

```
struct linked_node{
    int value;
    struct linked_node* next;
    struct linked_node* prev;
};

struct linked_list{
    int type_of_list;
    struct linked_node* head;
    struct linked_node* tail;
    int number_of_nodes;
};
```

These are written in *linked\_list.h* file and you are not allowed to modify any of these.

```
int list_exist;
```

Also, the above value is declared as a global variable and this value should be equal to 0 when there is no list created, and should be equal to 1 if a list is created. Since this value is used in print\_list function, students should modify this value whenever the existence of list changes.

**Description of the functions that are provided is listed below:**

1. You need to follow exactly what the descriptions say and implement each function according to the description.
2. Each function also has some error cases that it should be dealt with. Each error case always should alert the user with message and exit the function. Descriptions below will show what alarm message each function should generate, and you have to follow exactly the way it is written.
3. Always assume that node value is unique and more than one node with same value cannot be entered to the list. This condition will never be tested so that students do not have to implement this part.
4. Node value can be any integer value and you can assume that only integer will be inserted to the function so that you do not have to worry about handling this kind of error case.
5. When removing or inserting a node to a list, you should always change the *number\_of\_nodes* value of the list.
6. You are not allowed to use any library nor codes that are available through the web, friends, and etc. You are only required to use the give code and the files that are included within the .h file.
7. **If you cannot make your code when TA ask you to, we look on as cheating**

`struct linked_list* create_list (int number_of_nodes, int list_type)`

This function creates a *list* with the number of nodes. Number of nodes should be greater than 0 but if the input is less than 1, it should print the error message “**Function create\_list: the number of nodes is not specified correctly**” and exit the function. Also, if a *list* already exists, then it should print the error message “**Function create\_list: a list already exists**” and exit the function. There are two types of list, normal or stack, but the input value for the type is not 0 or 1, it should print the error message “**Function create\_list: the list type is wrong**”. When the nodes are inserted to the *list*, the node value created randomly but no duplication. And value’s range is from 1 to number of nodes. After the *list* is generated as specified, the function returns this *list*. This function allows the list to insert new nodes to either the head or the tail according to the input value(list\_type).

`struct linked_node* create_node (int node_value)`

This function creates a node that can be inserted to a list. *Node value* can be any integer.

`void insert_node(struct linked_list* list, struct linked_node* node)`

This function inserts a *node* to a *list*. This insertion will always insert a *node* to a head of the *list* which means the head of the *list* should be changed after a *new node* is inserted. For least hints, there is no duplication check in this function.

`void print_list(struct linked_list* list)`

This function prints a *list* in proper way.

Functions `create_list`, `insert_node` and `create_node` will be provided to students so that those can be used as reference of how to implement the functions and deal with the error cases. Also, `print_list` function will be provided to students as debugging tool so that students can see whether the output is properly made or not.

**The assignments:**

`void remove_list(struct linked_list* list)`

This function removes the *list*. When the *list* is removed, all the memory should be released to operating system (OS) so that OS lets other computer programs use this. While deleting the *list*, every node should be freed separately; `free(list)` will not remove every node in the *list*. To check whether the nodes are removed perfectly, for every deletion of a node, this function should print message “**The node with value n (corresponding value) is deleted!**” Also, if the whole *list* is deleted, this function should print message “**The list is completely deleted: n nodes are deleted**”.

`void remove_node(struct linked_list* list, int rm_node_value)`

This function removes a *node* with specified value. If there is only one node in the *list*, remove the *node* and remove the *list* also since there is nothing left. While removing a *node*, the *node* should be perfectly freed. If the type of list is stack, print the error message “**Function remove\_node: The list type is not normal. Removal is not allowed**” Also, if there is no such node to remove from the *list*, print the error message “**Function remove\_node: There is no such node to remove**” and exit the function.

`void push_Stack (struct linked_list* list, struct linked_node* node)`

This function inserts a node in stack manner. If the type of list is not stack, print the error message “**Function**

**push\_Stack: The list type is not a stack**" The new node will be always inserted to tail of the list which means the tail of the list should be changed after a new node is inserted.

**void pop\_Stack (struct linked\_list\* list, int number)**

This function removes some nodes in stack manner; the tail of the list will be removed, repeatedly. The parameter (variable *number\_of\_nodes*) means the number of nodes that will be removed. When parameter is bigger than 1, popping a node with *n* times, you do not remove node at one go. If there is only one node in the list, please make sure it frees (de-allocates) both the node and the list. If the list is not a stack type, print the error message **"Function pop\_Stack: The list type is not a stack"** and exit the function. If the *number\_of\_nodes* parameter is less than 1 or more than the number of nodes in the stack, respectively print the error message **"Function popStack: The number of nodes which will be removed is more than 1"** and **"Function popStack: The number of nodes which will be removed is more than that in the stack"**, then exit the function. The removed nodes should be freed.

**void search\_node(struct linked\_list\* list, int find\_node\_value)**

This function finds the node from the *list* that value is same with *find\_node\_value* and count the order of the node. This function should print message **"The order of (node\_value) is (order)."** and error message **"Function search\_node : There is no such node to search."**.

#### **Comments:**

1. You should comment within the function in order to make TAs and yourself what you have been tried to do with codes written. If there is some code that is not understandable and there is no comment on that, points will be deducted.
2. Comments should be all in ENGLISH. Sometimes compile error or running error occurs because the written language is not English. So we strongly recommend you to write comments in English. If a student did not write comments in English and some error happened because of that, all the responsibility belongs to the student so functionality points will be deducted.

### Project Requirements & Scoring Criteria (100 points total):

1. **This is an individual project.** Making a copy of the content of your friend's HW, books, google, naver, etc., will get an **F for this course**. You will not get any partial points for redoing the work. Even if you have different code or variable names, using the same function(s) that is not originated from C library which is initially installed with your Linux (or cygwin, etc.) will be regarded as copying. The person who provides the source will also consider as cheating and will be penalized. Also, asking your friends to do your work will definitely be considered as cheating.
2. If your code does not compile; meaning, there are errors and we cannot fix to grade your work, you will automatically get 0 for Project1 operation part. Please make sure to have compiled your code before turning it in.
3. Compiling in Linux (cygwin) environment is strongly recommended. (5 *extra* points) The initially given code may not be compiled in other compilers. If you use a compiler other than *gcc* compiler, you must be able to show the compiling process directly to the TA.
4. **Codes should have neither error nor warning when compiling: 5 points in total**
  - A. No errors: 3 points
  - B. No warnings: 2 points
5. **List operations: 80 points in total**

The following functions should be implemented: 80 points

  - A. void remove\_list(linked\_list\* list) (15 points)
    - i. Pointer Handling: each node is freed separately and list is freed
    - ii. Message is printed as instructed
    - iii. Comments
  - B. void remove\_node(linked\_list\* list, int rm\_node\_value) (20 points)
    - i. Error handlings as instructed
    - ii. Pointer Handling: the removed node is freed well.
    - iii. Number of nodes decreased correctly
    - iv. Connection of previous and next node each other
    - v. Comments
  - C. void push\_stack(linked\_list\* list, int number)(15 points)
    - i. Error handlings as instructed
    - ii. Number of nodes inserted correctly
    - iii. Connection of previous and next node each other
    - iv. Comments
  - D. void pop\_stack(linked\_list\* list, int number)(15 points)
    - i. Error handlings as instructed
    - ii. Number of nodes decreased correctly
    - iii. Connection of previous and next node each other
    - iv. Comments
  - E. void search\_node(linked\_list\* list, int find\_node\_value) (15 points)
    - i. Error handlings as instructed
    - ii. Display of result as instructed
    - iii. Comments
6. **Submission: 15 points in total**
  - ① **Due date is 11:59PM, October 26 (Monday)**
    - Submit as early as possible. No excuse for returned email (send failure).
    - **No late submission is allowed!!!!!!**
  - ② Include a README file in your zip file which explains your program: 10 points
  - ③ All files should be compressed in one .zip file: 2 points
    - The file name should be: [DSA PROJ1]2020170000\_홍길동.zip: 3 points
    - Submit the zip file to [lsm5505@korea.ac.kr](mailto:lsm5505@korea.ac.kr)
    - File should include:
      - i. linked\_list.c
      - ii. README file(It should include short explanation of each function. Korean can be allowed).