**DataStructure Introduction assignments**

Mandatory

1. Refer the example code of single linked list.

a. Modify existing data structure to include below additional data members. Update all relevant existing functions to accommodate this member update and test.

{

char id[MAX\_ID\_LEN]; //let max id length be 5

int val;

}

b. Add a function below to perform search and update based on name (str) input.

extern int search\_update\_name(Node \*\*, char \*search, char \*replace);

c. Add functions below() to perform a sorted insert operation (ascending order) on the list based on name (str) and on val.

//insert based on name (str member) in increasing order

extern int sorted\_insert\_name(Node \*\*, char \*, int val

A computer screen shot of a program

Description automatically generated

**In include:**

**Header.h**

/\* header.h \*/

#ifndef HEADER\_H

#define HEADER\_H

typedef struct Node {

int data;

struct Node\* next;

} Node;

void print\_list(Node\* head);

void push(Node\*\* head\_ref, int new\_data);

void free\_list(Node\* head);

#endif

**In src**

**Functions.c**

/\* functions.c \*/

#include <stdio.h>

#include <stdlib.h>

#include "header.h"

void print\_list(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

printf("%d -> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

void push(Node\*\* head\_ref, int new\_data) {

Node\* new\_node = (Node\*)malloc(sizeof(Node));

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

void free\_list(Node\* head) {

Node\* temp;

while (head != NULL) {

temp = head;

head = head->next;

free(temp);

}

}

**Main.c**

/\* main.c \*/

#include <stdio.h>

#include "header.h"

#include <stdlib.h>

int main() {

Node\* head = NULL;

push(&head, 1);

push(&head, 2);

push(&head, 3);

push(&head, 4);

printf("Linked List: ");

print\_list(head);

free\_list(head);

return 0;

}

A computer screen shot of a program

Description automatically generated

**Makefile**

# Compiler and flags

CC = gcc

CFLAGS = -c -Wall -g

INCLUDEPATH = ./include

SRCPATH = ./src

OBJPATH = ./obj

BINPATH = ./bin

# Target to create the executable

$(BINPATH)/exe: $(OBJPATH)/main.o $(OBJPATH)/functions.o

$(CC) -Wall -g -o $(BINPATH)/exe $(OBJPATH)/main.o $(OBJPATH)/functions.o

# Rule to compile main.o

$(OBJPATH)/main.o: $(SRCPATH)/main.c $(INCLUDEPATH)/header.h

$(CC) $(CFLAGS) $(SRCPATH)/main.c -I $(INCLUDEPATH)/

mv main.o $(OBJPATH)/

# Rule to compile functions.o

$(OBJPATH)/functions.o: $(SRCPATH)/functions.c $(INCLUDEPATH)/header.h

$(CC) $(CFLAGS) $(SRCPATH)/functions.c -I $(INCLUDEPATH)/

mv functions.o $(OBJPATH)/

# Clean up the build

clear:

rm -f $(BINPATH)/exe $(OBJPATH)/main.o $(OBJPATH)/functions.o