**Create bin directory**

**Directory Inc-Header.h**

#ifndef \_\_HEADER\_\_

#define \_\_HEADER\_\_

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

STANDARD HEADER FILES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MACROS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SIZE 25

#define MAX\_ID\_LEN 5

#define IN

#define OUT

#define TRUE 1

#define FALSE 0

#define SUCCESS 0

#define FAILURE 1

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

STRUCTURES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef struct Node

{

char id[MAX\_ID\_LEN]; // ID with max length 5

int val; // Integer value

char \*str; // String data

struct Node \*next; // Pointer to next node

} Node;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FUNCTION PROTOTYPES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

extern void myfflush(void);

extern void get\_string\_input(char \*);

extern int insert\_at\_beg(Node \*\*, char \*, char \*, int);

extern int insert\_in\_middle(Node \*\*, char \*, char \*, char \*, int);

extern int insert\_at\_end(Node \*\*, char \*, char \*, int);

extern int delete\_at\_beg(Node \*\*);

extern int delete\_in\_middle(Node \*\*, char \*str);

extern int delete\_at\_end(Node \*\*);

extern void display\_list(Node \*);

extern void free\_list(Node \*\*);

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

#endif

**directory make-Makefile:**

CC = gcc

CFLAGS = -c -Wall -g

INCLUDEPATH = ./../include

SRCPATH = ./../src

OBJPATH = ./../obj

BINPATH = ./../bin

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

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

mv exe $(BINPATH)

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

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

mv main.o $(OBJPATH)/

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

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

mv functions.o $(OBJPATH)/

clear:

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

**Create obj directory**

**Directory src-Functions.c**

#include <header.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Local Functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int create\_node(Node \*\*new\_node, int data\_len, char \*id, int val);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Create Node Function

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int create\_node(Node \*\*new\_node, int data\_len, char \*id, int val)

{

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

if (NULL == \*new\_node)

{

perror("error while creating node");

return FAILURE;

}

(\*new\_node)->str = (char \*)calloc(data\_len, sizeof(char));

if (NULL == (\*new\_node)->str)

{

perror("error while creating node, allocating memory for str");

return FAILURE;

}

strcpy((\*new\_node)->id, id);

(\*new\_node)->val = val;

(\*new\_node)->next = NULL;

return SUCCESS;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Insert at Beginning

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int insert\_at\_beg(Node \*\*head, char \*str, char \*id, int val)

{

Node \*new\_node = NULL;

if (FAILURE == create\_node(&new\_node, strlen(str) + 1, id, val))

{

return FAILURE;

}

strcpy(new\_node->str, str);

new\_node->next = \*head;

\*head = new\_node;

return SUCCESS;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Insert in Middle

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int insert\_in\_middle(Node \*\*head, char \*str, char \*after, char \*id, int val)

{

Node \*tmp = NULL;

Node \*new\_node = NULL;

if (FAILURE == create\_node(&new\_node, strlen(str) + 1, id, val))

{

return FAILURE;

}

strcpy(new\_node->str, str);

if(NULL == \*head)

{

\*head = new\_node;

return SUCCESS;

}

tmp = \*head;

while (NULL != tmp->next)

{

if(!strcmp(tmp->str, after))

{

new\_node->next = tmp->next;

tmp->next = new\_node;

return SUCCESS;

}

tmp = tmp->next;

}

tmp->next = new\_node;

return SUCCESS;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Insert at End

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int insert\_at\_end(Node \*\*head, char \*str, char \*id, int val)

{

Node \*tmp = NULL;

Node \*new\_node = NULL;

if (FAILURE == create\_node(&new\_node, strlen(str) + 1, id, val))

{

return FAILURE;

}

strcpy(new\_node->str, str);

if(NULL == \*head)

{

\*head = new\_node;

return SUCCESS;

}

tmp = \*head;

while (NULL != tmp->next)

{

tmp = tmp->next;

}

tmp->next = new\_node;

return SUCCESS;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FUNCTION\_NAME: myfflush

DESCRIPTION:

This function is a replacement of inbuilt

function fflush(stdin)

PARAMETERS:

No parameters

RETURN VALUE:

It returns void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void myfflush(void)

{

while('\n' != getchar());

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FUNCTION\_NAME: get\_string\_input

DESCRIPTION:

This function reads a string from the stdin.

PARAMETERS:

Parameters are self descriptive.

RETURN VALUE:

It returns void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void get\_string\_input(char \*input\_string)

{

while(1)

{

fgets(input\_string, SIZE-1, stdin);

int len = strlen(input\_string);

if (1 == len)

{

printf("\tempty string, enter again: ");

memset(input\_string, 0, SIZE);

continue;

}

if ('\n' == input\_string[len - 1])

{

input\_string[len - 1] = '\0';

}

else

{

myfflush();

}

break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Display List

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display\_list(Node \*head)

{

if (NULL == head)

{

printf("\tList is empty\n");

return;

}

else

{

printf("\tElements in list are:\n");

}

while (head != NULL)

{

printf("\tID: %s, Value: %d, String: %s\n", head->id, head->val, head->str);

head = head->next;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Search and Update by Name

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

Node \*tmp = \*head;

while (tmp != NULL)

{

if (strcmp(tmp->str, search) == 0)

{

strcpy(tmp->str, replace);

printf("\tNode updated: %s\n", replace);

return SUCCESS;

}

tmp = tmp->next;

}

printf("\tNode with string \"%s\" not found.\n", search);

return FAILURE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Delete Functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int delete\_at\_beg(Node \*\*head)

{

Node \*tmp = \*head;

if (NULL == tmp)

{

printf("\tList is empty, deletion failed\n");

return FAILURE;

}

\*head = (\*head)->next;

free(tmp->str);

free(tmp);

return SUCCESS;

}

int delete\_in\_middle(Node \*\*head, char \*str)

{

Node \*tmp = \*head;

if (NULL == tmp)

{

printf("\tList is empty, deletion failed\n");

return FAILURE;

}

else

{

if (!strcmp(tmp->str, str))

{

\*head = tmp->next;

free(tmp->str);

free(tmp);

return SUCCESS;

}

}

if (NULL == tmp->next)

{

printf("\t\"%s\" does not exist in the list\n", str);

return FAILURE;

}

do

{

if (!strcmp(tmp->next->str, str))

{

Node \*free\_node = tmp->next;

tmp->next = tmp->next->next;

free(free\_node->str);

free(free\_node);

return SUCCESS;

}

tmp = tmp->next;

} while (NULL != tmp->next);

printf("\t\"%s\" does not exist in the list\n", str);

return FAILURE;

}

int delete\_at\_end(Node \*\*head)

{

Node \*tmp = \*head;

if (NULL == tmp)

{

printf("\tList is empty, deletion failed\n");

return FAILURE;

}

if (NULL == tmp->next)

{

free(tmp->str);

free(tmp);

\*head = NULL;

return SUCCESS;

}

while (NULL != tmp->next->next)

{

tmp = tmp->next;

}

Node \*free\_node = tmp->next;

tmp->next = NULL;

free(free\_node->str);

free(free\_node);

return SUCCESS;

}

void free\_list(Node \*\*head)

{

Node \*tmp = \*head;

while (tmp != NULL)

{

Node \*free\_node = tmp;

tmp = tmp->next;

free(free\_node->str);

free(free\_node);

}

\*head = NULL;

}

**Directory src – main.c**

#include <header.h>

int main(int argc, char \*argv[])

{

Node \*head = NULL;

while (1)

{

system("clear");

int choice = 0;

char str[SIZE];

char after[SIZE];

char id[MAX\_ID\_LEN];

int val;

memset(str, 0, SIZE);

memset(after, 0, SIZE);

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*Linked List Operation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printf("\t1. Insert at Beginning\n");

printf("\t2. Insert after a Node\n");

printf("\t3. Insert at End\n");

printf("\t4. Delete at Beginning\n");

printf("\t5. Delete a specified Node\n");

printf("\t6. Delete at End\n");

printf("\t7. Display\n");

printf("\t8. Search and Update by Name\n");

printf("\t9. Exit\n\n");

printf("\tEnter your choice: ");

scanf("%d", &choice);

myfflush();

switch(choice)

{

case 1:

printf("\tEnter string: ");

get\_string\_input(str);

printf("\tEnter ID (max 5 chars): ");

get\_string\_input(id);

printf("\tEnter value: ");

scanf("%d", &val);

if (SUCCESS == insert\_at\_beg(&head, str, id, val))

{

printf("\tInserted Successfully\n");

}

break;

case 2:

printf("\tEnter string: ");

get\_string\_input(str);

printf("\tEnter string after which you want to insert: ");

get\_string\_input(after);

printf("\tEnter ID (max 5 chars): ");

get\_string\_input(id);

printf("\tEnter value: ");

scanf("%d", &val);

if (SUCCESS == insert\_in\_middle(&head, str, after, id, val))

{

printf("\tInserted Successfully\n");

}

break;

case 3:

printf("\tEnter string: ");

get\_string\_input(str);

printf("\tEnter ID (max 5 chars): ");

get\_string\_input(id);

printf("\tEnter value: ");

scanf("%d", &val);

if (SUCCESS == insert\_at\_end(&head, str, id, val))

{

printf("\tInserted Successfully\n");

}

break;

case 4:

if (SUCCESS == delete\_at\_beg(&head))

{

printf("\tDeleted Successfully\n");

}

break;

case 5:

printf("\tEnter string which you want to delete: ");

get\_string\_input(str);

if (SUCCESS == delete\_in\_middle(&head, str))

{

printf("\tDeleted Successfully\n");

}

break;

case 6:

if (SUCCESS == delete\_at\_end(&head))

{

printf("\tDeleted Successfully\n");

}

break;

case 7:

display\_list(head);

break;

case 8:

printf("\tEnter name to search and update: ");

get\_string\_input(str);

printf("\tEnter new name to replace: ");

get\_string\_input(after);

if (SUCCESS == search\_update\_name(&head, str, after))

{

printf("\tUpdated Successfully\n");

}

break;

case 9:

free\_list(&head);

printf("\tExited\n");

return SUCCESS;

default:

printf("\tWrong choice, try again\n");

break;

}

printf("\n\tPress enter to continue...");

myfflush();

}

return SUCCESS;

}