#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

struct node

{

int data;

struct node \*next;

};

struct node \*start = NULL;

struct node \*create\_ll(struct node \*);

struct node \*display(struct node \*);

struct node \*insert\_beg(struct node \*);

struct node \*insert\_end(struct node \*);

struct node \*insert\_before(struct node \*);

struct node \*insert\_after(struct node \*);

struct node \*delete\_beg(struct node \*);

struct node \*delete\_end(struct node \*);

struct node \*delete\_node(struct node \*);

struct node \*delete\_after(struct node \*);

struct node \*delete\_list(struct node \*);

struct node \*sort\_list(struct node \*);

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

{

int option;

do

{

printf("\n\n \* \*\*\*\*MAIN MENU \* \*\*\*\*");

printf("\n 1 : Create a list");

printf("\n 2 : Display the list");

printf("\n 3 : Add a node at the beginning");

printf("\n 4 : Add a node at the end");

printf("\n 5 : Add a node before a given node");

printf("\n 6 : Add a node after a given node");

printf("\n 7 : Delete a node from the beginning");

printf("\n 8 : Delete a node from the end");

printf("\n 9 : Delete a given node");

printf("\n 10 : Delete a node after a given node");

printf("\n 11 : Delete the entire list");

printf("\n 12 : Sort the list");

printf("\n 13 : EXIT");

printf("\n\n Enter your option : ");

scanf("%d", &option);

switch (option)

{

case 1:

start = create\_ll(start);

printf("\n LINKED LIST CREATED");

break;

case 2:

start = display(start);

break;

case 3:

start = insert\_beg(start);

break;

case 4:

start = insert\_end(start);

break;

case 5:

start = insert\_before(start);

break;

case 6:

start = insert\_after(start);

break;

case 7:

start = delete\_beg(start);

break;

case 8:

start = delete\_end(start);

break;

case 9:

start = delete\_node(start);

break;

case 10:

start = delete\_after(start);

break;

case 11:

start = delete\_list(start);

printf("\n LINKED LIST DELETED");

break;

case 12:

start = sort\_list(start);

break;

}

} while (option != 13);

return 0;

}

struct node \*create\_ll(struct node \*start)

{

struct node \*new\_node, \*ptr;

int num;

printf("\n Enter - 1 to end");

printf("\n Enter the data: ");

scanf("%d", &num);

while (num != -1)

{

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data = num;

if (start == NULL)

{

new\_node->next = NULL;

start = new\_node;

}

else

{

ptr = start;

while (ptr->next != NULL)

ptr = ptr->next;

ptr->next = new\_node;

new\_node->next = NULL;

}

printf("\n Enter the data : ");

scanf("%d", &num);

}

return start;

}

struct node \*display(struct node \*start)

{

struct node \*ptr;

ptr = start;

while (ptr != NULL)

{

printf("\t % d", ptr->data);

ptr = ptr->next;

}

return start;

}

struct node \*insert\_beg(struct node \*start)

{

struct node \*new\_node;

int num;

printf("\n Enter the data : ");

scanf("%d", &num);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data = num;

new\_node->next = start;

start = new\_node;

return start;

}

struct node \*insert\_end(struct node \*start)

{

struct node \*ptr, \*new\_node;

int num;

printf("\n Enter the data : ");

scanf("%d", &num);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data = num;

new\_node->next = NULL;

ptr = start;

while (ptr->next != NULL)

ptr = ptr->next;

ptr->next = new\_node;

return start;

}

struct node \*insert\_before(struct node \*start)

{

struct node \*new\_node, \*ptr, \*preptr;

int num, val;

printf("\n Enter the data : ");

scanf("%d", &num);

printf("\n Enter the value before which the data has to be inserted : ");

scanf("%d", &val);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data = num;

ptr = start;

while (ptr->data != val)

{

preptr = ptr;

ptr = ptr->next;

}

preptr->next = new\_node;

new\_node->next = ptr;

return start;

}

struct node \*insert\_after(struct node \*start)

{

struct node \*new\_node, \*ptr, \*preptr;

int num, val;

printf("\n Enter the data : ");

scanf("%d", &num);

printf("\n Enter the value after which the data has to be inserted : ");

scanf("%d", &val);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data = num;

ptr = start;

preptr = ptr;

while (preptr->data != val)

{

preptr = ptr;

ptr = ptr->next;

}

preptr->next = new\_node;

new\_node->next = ptr;

return start;

}

struct node \*delete\_beg(struct node \*start)

{

struct node \*ptr;

ptr = start;

start = start->next;

free(ptr);

return start;

}

struct node \*delete\_end(struct node \*start)

{

struct node \*ptr, \*preptr;

ptr = start;

while (ptr->next != NULL)

{

preptr = ptr;

ptr = ptr->next;

}

preptr->next = NULL;

free(ptr);

return start;

}

struct node \*delete\_node(struct node \*start)

{

struct node \*ptr, \*preptr;

int val;

printf("\n Enter the value of the node which has to be deleted: ");

scanf("%d", &val);

ptr = start;

if (ptr->data == val)

{

start = delete\_beg(start);

return start;

}

else

{

while (ptr->data != val)

{

preptr = ptr;

ptr = ptr->next;

}

preptr->next = ptr->next;

free(ptr);

return start;

}

}

struct node \*delete\_after(struct node \*start)

{

struct node \*ptr, \*preptr;

int val;

printf("\n Enter the value after which the node has to deleted: ");

scanf("%d", &val);

ptr = start;

preptr = ptr;

while (preptr->data != val)

{

preptr = ptr;

ptr = ptr->next;

}

preptr->next = ptr->next;

free(ptr);

return start;

}

struct node \*delete\_list(struct node \*start)

{

struct node \*ptr; // Lines 252-254 were modified from original code to fix unresposiveness in output window

if (start != NULL)

{

ptr = start;

while (ptr != NULL)

{

printf("\n % d is to be deleted next", ptr->data);

start = delete\_beg(ptr);

ptr = start;

}

}

return start;

}

struct node \*sort\_list(struct node \*start)

{

struct node \*ptr1, \*ptr2;

int temp;

ptr1 = start;

while (ptr1->next != NULL)

{

ptr2 = ptr1->next;

while (ptr2 != NULL)

{

if (ptr1->data > ptr2->data)

{

temp = ptr1->data;

ptr1->data = ptr2->data;

ptr2->data = temp;

}

ptr2 = ptr2->next;

}

ptr1 = ptr1->next;

return start; // Had to be added

}

}