Linked List Operation

Team Members:-
Adithya Vimukthi -824
Shehan Nandasena - 854
Randula Senanayake - 804
Sarindu Shehan - 1230

Link List operations in c code

```c
//including libraries
#include <stdio.h>
#include <stdlib.h>

//define the structure of a node
struct node {
    int data;
    struct node* nlink;
};
```

```c
//Define the functions
struct node* insertdata(int getdata, struct node* hlink);

void printList(struct node* hlink);

void srhnode(struct node* hlink);

int updatenode(struct node* hlink);

int addnode(struct node* hlink);

void comprenode(struct node* hlink);

void displaynode(struct node* hlink);

int countlistsize(struct node* hlink);

int sortList(struct node* hlink);

void deleteNode(struct node **head);

void removeList(struct node **hlink);
```

Main function

```c
int main(){
    struct node* headp = NULL;          //Start with empty list

    headp = insertdata(12, headp);      //Inserting first five nodes
    headp = insertdata(32, headp);
    headp = insertdata(-5, headp);
    headp = insertdata(123, headp);
    headp = insertdata(-23, headp);

    printList(headp);   //Print the list
    printf("\n");

    srhnode(headp);     //Serching the nodes by node value
    printf("\n");

    updatenode(headp);  //Updating the exsisting node value
    printList(headp);
    printf("\n");

    addnode(headp);     //Adding new node as last node of the list
    printList(headp);
    printf("\n");

    comprenode(headp);  //Compere 2 node values
    printf("\n");

    displaynode(headp); //Display a single node
    printf("\n");

    countlistsize(headp);//Count the size of the list (Number of nodes)
    printf("\n");

    sortList(headp);    //Sorting the nodes according to values
    printList(headp);
    printf("\n");

    deleteNode(&headp); //Deleting a single node
    printList(headp);
    printf("\n");

    removeList(&headp); //Deleting the whole list
    printList(headp);
    printf("\n");

    return 0;
```

```c
//Function for create nodes
struct node* insertdata (int getdata, struct node* hlink){
    struct node* loop = NULL;
    struct node* prev = NULL;

    //Alocating memory for new node
    struct node* newnode = malloc(sizeof(struct node));

    //Assigning value for new node
    (*newnode).data = getdata;
    (*newnode).nlink = NULL;

    if(hlink == NULL){              //If list is empty make the new node as first node
        hlink = newnode;
    }else{                          //If the list has nodes
        loop = hlink;
        while(loop!=NULL){          //Find the last node
            prev = loop;
            loop=(*loop).nlink;
        }
        (*prev).nlink = newnode;    //Add new node as last node
    }
    return hlink;                   //Update the list
}
```

```c
//function for print the list
void printList(struct node* hlink){
    struct node*loop = NULL;
    loop = hlink;                       //Assigning head node
    while(loop!=NULL){
        printf("| %d |",(*loop).data);  //Print the node data part
        printf(" %p |",(*loop).nlink);  //Print the next node address
        loop=(*loop).nlink;             //go to next node
        if(loop != NULL){               //Arrow printing mechanism
            printf(" --> ");
        }else{
            printf(" - End of the List -");
        }
    }
    printf("\n");
}
```

```c
//Function for search node
void srhnode(struct node* hlink){
    int srhval,nodenum,fudornot;            //Creating variables
    struct node*loop = NULL;
    loop = hlink;

    printf("Enter a value for search: ");   //Ask searching value from user
    scanf("%d", &srhval);

    while(loop!=NULL){
        nodenum ++;
        if(srhval == (*loop).data){         //Check whether any node value is equal to given search value
            fudornot = 1;
            break;
        }
        loop=(*loop).nlink;                 //Go to next node
    }

    if(fudornot == 1){                      //Publish the results of search
        printf("Value found in: %d node ",nodenum);
    }else{
        printf("Value not found in list.");
    }
    printf("\n");
}
```

```c
//function for update exsisting node
int updatenode(struct node* hlink){
    int nodenum,upval,getnum,fudornot;
    struct node*loop = NULL;
    loop = hlink;

    printf("Enter updating node number: "); //Get updating node number
    scanf("%d", &getnum);

    printf("Enter updating value : ");      //Get updating value
    scanf("%d", &upval);

    while(loop!=NULL){
        nodenum ++;
        if(nodenum == getnum){
            (*loop).data = upval;           //Updating the node value
            fudornot = 1;
            break;
        }
        loop=(*loop).nlink;
    }

    if(fudornot == 1){                      //Publish updated results
        printf("Node %d value is updated to %d ",getnum,upval);
    }else{
        printf("Node is not found");
    }
    printf("\n");
    return 0;
}
```

```c
//Add node to the list
int addnode(struct node* hlink){
    int newval;

    printf("Enter new value : ");                   //Ask for new value
    scanf("%d", &newval);

    struct node* loop = NULL;
    struct node* prev = NULL;

    struct node* newnode = malloc(sizeof(struct node)); //Memory allocation for new node

    (*newnode).data = newval;                       //Creating new node and assigning value
    (*newnode).nlink = NULL;

    if(hlink == NULL){                              //Adding the new node to the list
        hlink = newnode;
    }else{
        loop = hlink;
        while(loop!=NULL){
            prev = loop;
            loop=(*loop).nlink;
        }
        (*prev).nlink = newnode;
    }
    return 0;
}
```

```c
//Function for compering 2 node values
void comprenode(struct node* hlink){
    int val1, val2, node1, node2, nodenum; //Creating variables
    struct node*loop = NULL;
    loop = hlink;

    printf("Enter node number 1 : "); //Get node numbers
    scanf("%d", &node1);

    printf("Enter node number 2 : ");
    scanf("%d", &node2);

    while(loop!=NULL){
        nodenum ++;
        if(nodenum == node1){           //Assign node values to variables
            val1 = (*loop).data;
        }else if(nodenum == node2){
            val2 = (*loop).data;
        }
        loop=(*loop).nlink;
    }
    if (val1 > val2){                   //Compering node values and printing results
        printf("Node %d  value is grater than node %d  value ", node1, node2);
    }else if (val1 == val2){
        printf("Node %d  value is equals to node %d  value ", node1, node2);
    }else{
        printf("Node %d  value is lower than node %d  value ", node1, node2);
    }
    printf("\n");
}
```

```c
//Function for display single node
void displaynode(struct node* hlink){
    int disnode, nodeval,nodenum;
    struct node*loop = hlink;

    printf("Enter node number to display: ");   //Get node number
    scanf("%d", &disnode);

    while(loop!=NULL){
        nodenum ++;
        if(nodenum == disnode){                 //Visit for given node
            nodeval = (*loop).data;
            printf("Node Vlaue is: %d",nodeval);//Print the node value
            break;
        }
        loop=(*loop).nlink;
    }
    printf("\n");
}
```

```c
//Function for count the list size
int countlistsize(struct node* hlink){
    int nodenum;
    struct node*loop = hlink;

    while(loop!=NULL){
        nodenum ++;                     //Counting the nodes
        loop=(*loop).nlink;
    }

    printf("List size is: %d",nodenum); //Print the results
    printf("\n");
    return nodenum;
}
```

```c
int sortList(struct node* hlink) {
        //Node current will point to head
        struct node *loop = hlink, *next = NULL, *prev = NULL;
        int temp,code;

        //Get sorting order
        printf("Sort Low to High - Enter 1 , Sort High to Low - Enter 2 : ");
        scanf("%d", &code);

        if (code == 1){ //Sorting the list to ascending order
            while(loop != NULL) {
                next = (*loop).nlink;
                while(next != NULL) {
                    //If current node's data is greater than index's node data, swap the data between them
                    if((*loop).data > (*next).data) {
                        temp = (*loop).data;
                        (*loop).data = (*next).data;
                        (*next).data = temp;
                    }
                    next = (*next).nlink;
                }
                loop = (*loop).nlink;
            }

        }else if (code == 2){//Sorting the list to descending order
            while(loop != NULL) {
                next = (*loop).nlink;
                while(next != NULL) {
                    //If current node's data is lower than index's node data, swap the data between them
                    if((*loop).data < (*next).data) {
                        temp = (*loop).data;
                        (*loop).data = (*next).data;
                        (*next).data = temp;
                    }
                    next = (*next).nlink;
                }
                loop = (*loop).nlink;
            }

        }else {
            printf("Invalide index!");
        }

    printf("\n");
```

```c
//Function for Delete node
void deleteNode(struct node **hlink){
      struct node *temp;
      int val;
      struct node *loop  = *hlink;

      printf("Enter deleting value : "); //Get deleting value
      scanf("%d", &val);

      //Move to head node to the next and free the head.
      if((*(*hlink)).data == val)
      {
          temp = *hlink;    //Backup to free the memory
          *hlink = (*(*hlink)).nlink;
          free(temp);
      }
      else //If deleting middle node or end node
      {
          while((*loop).nlink != NULL)
          {
              if((*(*loop).nlink).data == val)
              {
                  temp = (*loop).nlink;
                  (*loop).nlink =(*(*loop).nlink).nlink;
                  free(temp);
                  break;
              }
              else
                  loop = (*loop).nlink;//Go to next node
          }
      }
      printf("\n");
}
```

```c
// Function to delete the entire linked list
void removeList(struct node **hlink){

   struct node* loop = *hlink;
   struct node* nlink;
   int val;

    printf("Press 1 for delete list: "); //Get confirmation for delete list
    scanf("%d", &val);

    if(val == 1){
        while (loop != NULL)
        {
            nlink = (*loop).nlink;      //Deleteing all nodes
            free(loop);
            loop = nlink;

        }
        *hlink = NULL;                  //Clear the head node
        printf("List is deleted");
    }

}
```

Results

1st result set

```
PS D:\Github\List_operation_in_C\list_operations> cd "d:\Github\List_operation_in_C\list_operations\" ; if ($?) { gcc list_new.c -o list_new } ; if ($?) { .\list_new }
| 12 | 000001efc4d81470 | --> | 32 | 000001efc4d81490 | --> | -5 | 000001efc4d814b0 | --> | 123 | 000001efc4d814d0 | --> | -23 | 0000000000000000 | - End of the List -

Enter a value for search: 32
Value found in: 2 node

Enter updating node number: 3
Enter new value : 99
| 12 | 00000276db6d1470 | --> | 32 | 00000276db6d1490 | --> | 321 | 00000276db6d14b0 | --> | 123 | 00000276db6d14d0 | --> | -23 | 00000276db6d14f0 | --> | 99 | 0000000000000000 | - End of the List -

Enter node number 1 : 6
Enter node number 2 : 4
Node 6  value is lower than node 4  value

Enter node number to display: 3
Node Vlaue is: 321

List size is: 6

Sort Low to High - Enter 1 , Sort High to Low - Enter 2 : 2

| 321 | 00000276db6d1470 | --> | 123 | 00000276db6d1490 | --> | 99 | 00000276db6d14b0 | --> | 32 | 00000276db6d14d0 | --> | 12 | 00000276db6d14f0 | --> | -23 | 0000000000000000 | - End of the List -

Enter deleting value : 32

| 321 | 00000276db6d1470 | --> | 123 | 00000276db6d1490 | --> | 99 | 00000276db6d14b0 | --> | 12 | 00000276db6d14f0 | --> | -23 | 0000000000000000 | - End of the List -

Press 1 for delete list: 1
List is deleted

PS D:\Github\List_operation_in_C\list_operations>
```

2nd result set

```
PS D:\Github\List_operation_in_C\list_operations> cd "d:\Github\List_operation_in_C\list_operations\" ; if ($?) { gcc list_new.c -o list_new } ; if ($?) { .\list_new }
| 12 | 0000020ca32b1470 | --> | 32 | 0000020ca32b1490 | --> | -5 | 0000020ca32b14b0 | --> | 123 | 0000020ca32b14d0 | --> | -23 | 0000000000000000 | - End of the List -

Enter a value for search: -5
Value found in: 3 node

Enter updating node number: 5
Enter updating value : -67
Node 5 value is updated to -67
| 12 | 0000020ca32b1470 | --> | 32 | 0000020ca32b1490 | --> | -5 | 0000020ca32b14b0 | --> | 123 | 0000020ca32b14d0 | --> | -67 | 0000000000000000 | - End of the List -

Enter new value : 88
| 12 | 0000020ca32b1470 | --> | 32 | 0000020ca32b1490 | --> | -5 | 0000020ca32b14b0 | --> | 123 | 0000020ca32b14d0 | --> | -67 | 0000020ca32b14f0 | --> | 88 | 0000000000000000 | - End of the List -

Enter node number 1 : 5
Enter node number 2 : 3
Node 5  value is lower than node 3  value

Enter node number to display: 1
Node Vlaue is: 12

List size is: 6

Sort Low to High - Enter 1 , Sort High to Low - Enter 2 : 1

| -67 | 0000020ca32b1470 | --> | -5 | 0000020ca32b1490 | --> | 12 | 0000020ca32b14b0 | --> | 32 | 0000020ca32b14d0 | --> | 88 | 0000020ca32b14f0 | --> | 123 | 0000000000000000 | - End of the List -

Enter deleting value : 12

| -67 | 0000020ca32b1470 | --> | -5 | 0000020ca32b14b0 | --> | 32 | 0000020ca32b14d0 | --> | 88 | 0000020ca32b14f0 | --> | 123 | 0000000000000000 | - End of the List -

Press 1 for delete list: 1
List is deleted

PS D:\Github\List_operation_in_C\list_operations>
```

Full code

```
//including libraries
#include <stdio.h>
#include <stdlib.h>

//define the structure of a node
struct node {
   int data;
   struct node* nlink;
};

//Define the functions
struct node* insertdata(int getdata, struct node* hlink);

void printList(struct node* hlink);

void srhnode(struct node* hlink);

int updatenode(struct node* hlink);

int addnode(struct node* hlink);

void comprenode(struct node* hlink);

void displaynode(struct node* hlink);

int countlistsize(struct node* hlink);

int sortList(struct node* hlink);

void deleteNode(struct node **head);

void removeList(struct node **hlink);

//main function
int main(){
        struct node* headp = NULL;                  //Start with empty list

        headp = insertdata(12, headp);              //Inserting first five nodes
        headp = insertdata(32, headp);
        headp = insertdata(-5, headp);
        headp = insertdata(123, headp);
        headp = insertdata(-23, headp);

        printList(headp);  //Print the list
        printf("\n");

        srhnode(headp);  //Serching the nodes by node value
        printf("\n");

        updatenode(headp);  //Updating the exsisting node value
        printList(headp);
        printf("\n");

        addnode(headp);          //Adding new node as last node of the list
        printList(headp);
        printf("\n");

        comprenode(headp);       //Compere 2 node values
        printf("\n");

        displaynode(headp);      //Display a single node
```

```
	printf("\n");

	countlistsize(headp);//Count the size of the list (Number of nodes)
	printf("\n");

	sortList(headp);   //Sorting the nodes according to values
	printList(headp);
	printf("\n");

	deleteNode(&headp);	//Deleting a single node
	printList(headp);
	printf("\n");

	removeList(&headp);	//Deleting the whole list
	printList(headp);
	printf("\n");

	return 0;
}

//Function for create nodes
struct node* insertdata (int getdata, struct node* hlink){
   struct node* loop = NULL;
   struct node* prev = NULL;

	//Alocating memory for new node
   struct node* newnode = malloc(sizeof(struct node));

	//Assigning value for new node
   (*newnode).data = getdata;
   (*newnode).nlink = NULL;

   if(hlink == NULL){				//If list is empty make the new node as first node
      hlink = newnode;
   }else{					//If the list has nodes
      loop = hlink;
      while(loop!=NULL){		//Find the last node
         prev = loop;
         loop=(*loop).nlink;
      }
      (*prev).nlink = newnode;	//Add new node as last node
   }
   return hlink;				//Update the list
}

//function for print the list
void printList(struct node* hlink){
	struct node*loop = NULL;
	loop = hlink;					//Assigning head node
	while(loop!=NULL){
		printf("| %d |",(*loop).data);	//Print the node data part
		printf(" %p |",(*loop).nlink);	//Print the next node address
		loop=(*loop).nlink;				//go to next node
		if(loop != NULL){			//Arrow printing mechanism
			printf(" --> ");
		}else{
			printf(" - End of the List -");
		}
	}
	printf("\n");
}

//Function for search node
```

```
void srhnode(struct node* hlink){
	int srhval,nodenum,fudornot;				//Creating variables
	struct node*loop = NULL;
	loop = hlink;

	printf("Enter a value for search: "); //Ask searching value from user
   scanf("%d", &srhval);

   while(loop!=NULL){
	nodenum ++;
	if(srhval == (*loop).data){			//Check whether any node value is equal to given search value
		fudornot = 1;
		break;
		}
		loop=(*loop).nlink;						//Go to next node
	}

	if(fudornot == 1){						//Publish the results of search
		printf("Value found in: %d node ",nodenum);
	}else{
		printf("Value not found in list.");
	}
	printf("\n");
}

//function for update exsisting node
int updatenode(struct node* hlink){
	int nodenum,upval,getnum,fudornot;
	struct node*loop = NULL;
	loop = hlink;

	printf("Enter updating node number: "); //Get updating node number
   scanf("%d", &getnum);

   printf("Enter updating value : ");		//Get updating value
   scanf("%d", &upval);

   while(loop!=NULL){
	nodenum ++;
	if(nodenum == getnum){
		(*loop).data = upval;				//Updating the node value
		fudornot = 1;
		break;
		}
		loop=(*loop).nlink;
	}

	if(fudornot == 1){						//Publish updated results
		printf("Node %d value is updated to %d ",getnum,upval);
	}else{
		printf("Node is not found");
	}
	printf("\n");
	return 0;
}

//Add node to the list
int addnode(struct node* hlink){
	int newval;

	printf("Enter new value : ");					//Ask for new value
   scanf("%d", &newval);
```

```
   struct node* loop = NULL;
   struct node* prev = NULL;

   struct node* newnode = malloc(sizeof(struct node)); //Memory allocation for new node

   (*newnode).data = newval;                                                    //Creating new node and assigning
value
   (*newnode).nlink = NULL;

   if(hlink == NULL){                                                           //Adding the new node to the
list
      hlink = newnode;
   }else{
      loop = hlink;
      while(loop!=NULL){
         prev = loop;
         loop=(*loop).nlink;
      }
      (*prev).nlink = newnode;
   }
   return 0;
}

//Function for compering 2 node values
void comprenode(struct node* hlink){
        int val1, val2, node1, node2, nodenum; //Creating variables
        struct node*loop = NULL;
        loop = hlink;

        printf("Enter node number 1 : "); //Get node numbers
   scanf("%d", &node1);

   printf("Enter node number 2 : ");
   scanf("%d", &node2);

   while(loop!=NULL){
        nodenum ++;
        if(nodenum == node1){                   //Assign node values to variables
                val1 = (*loop).data;
                }else if(nodenum == node2){
                        val2 = (*loop).data;
                }
                loop=(*loop).nlink;
        }
   if (val1 > val2){                            //Compering node values and printing results
        printf("Node %d  value is grater than node %d  value ", node1, node2);
        }else if (val1 == val2){
                printf("Node %d  value is equals to node %d  value ", node1, node2);
        }else if (val1 < val2){
                printf("Node %d  value is lower than node %d  value ", node1, node2);
        }
        printf("\n");
}

//Function for display single node
void displaynode(struct node* hlink){
        int disnode, nodeval,nodenum;
        struct node*loop = hlink;

        printf("Enter node number to display: ");       //Get node number
   scanf("%d", &disnode);

   while(loop!=NULL){
```

```
	nodenum ++;
	if(nodenum == disnode){                                    //Visit for given node
		nodeval = (*loop).data;
		printf("Node Vlaue is: %d",nodeval);//Print the node value
		break;
		}
		loop=(*loop).nlink;
	}
	printf("\n");
}

//Function for count the list size
int countlistsize(struct node* hlink){
	int nodenum;
	struct node*loop = hlink;

	while(loop!=NULL){
	nodenum ++;                                                //Counting the nodes
		loop=(*loop).nlink;
	}

	printf("List size is: %d",nodenum);  //Print the results
	printf("\n");
	return nodenum;
}

int sortList(struct node* hlink) {
	//Node current will point to head
	struct node *loop = hlink, *next = NULL, *prev = NULL;
	int temp,code;

		//Get sorting order
	printf("Sort Low to High - Enter 1 , Sort High to Low - Enter 2 : ");
		scanf("%d", &code);

		if (code == 1){ //Sorting the list to ascending order
			while(loop != NULL) {
		next = (*loop).nlink;
		while(next != NULL) {
		  //If current node's data is greater than index's node data, swap the data between them
			if((*loop).data > (*next).data) {
			temp = (*loop).data;
			(*loop).data = (*next).data;
			(*next).data = temp;
			}
			next = (*next).nlink;
		}
		loop = (*loop).nlink;
		}

			}else if (code == 2){//Sorting the list to descending order
				while(loop != NULL) {
		next = (*loop).nlink;
		while(next != NULL) {
		  //If current node's data is lower than index's node data, swap the data between them
			if((*loop).data < (*next).data) {
			temp = (*loop).data;
			(*loop).data = (*next).data;
			(*next).data = temp;
			}
			next = (*next).nlink;
		}
		loop = (*loop).nlink;
```

```
            }

                        }else {
                                    printf("Invalide index!");
                        }

    printf("\n");
    return 0;
}

//Function for Delete node
void deleteNode(struct node **hlink){
     struct node *temp;
     int val;
     struct node *loop  = *hlink;

     printf("Enter deleting value : "); //Get deleting value
     scanf("%d", &val);

     //Move to head node to the next and free the head.
     if((*(*hlink)).data == val)
     {
        temp = *hlink;    //Backup to free the memory
        *hlink = (*(*hlink)).nlink;
        free(temp);
     }
     else //If deleting middle node or end node
     {
        while((*loop).nlink != NULL)
        {
           if((*(*loop).nlink).data == val)
           {
              temp = (*loop).nlink;
              (*loop).nlink =(*(*loop).nlink).nlink;
              free(temp);
              break;
           }
           else
              loop = (*loop).nlink;//Go to next node
        }
     }
              printf("\n");
}

// Function to delete the entire linked list
void removeList(struct node **hlink){

   struct node* loop = *hlink;
   struct node* nlink;
   int val;

            printf("Press 1 for delete list: "); //Get confirmation for delete list
    scanf("%d", &val);

            if(val == 1){
                        while (loop != NULL)
                        {
                        nlink = (*loop).nlink;                    //Deleteing all nodes
                        free(loop);
                        loop = nlink;

                        }
                        *hlink = NULL;                                          //Clear the head node
```

```
                printf("List is deleted");
        }

}
```