DS Assignments

Assignment – 1

Implement the functions given below :-

1. insert_at_first(head, data)
2. insert_at_last(head, data)
3. delete_first(head)
4. delete_last(head)
5. delete_list(head)
6. find_node(head, data)

1. insert_at_first :

Input : head → pointer to first node.
data → data to be inserted.

Read data from user and insert the given data in first position.

Cases : **List empty, data = 10**

head = NULL , After inserting data into list, list contains
10 → NULL

List not empty, data 40

10 → 20 → 30 → NULL, After inserting 40 into list

40 → 10 → 20 → 30 → NULL

2. Insert_at_last :

Input : head → pointer to first node
data → data to be inserted at the end.

Cases : List empty
List not empty

1. List empty – Update the head with new node address.
2. List not empty – Traverse to the last node and establish the link between last node and new node.

3. delete_first :

Input : head → pointer to the first node.

Cases : List empty
List not empty

1. List empty → Return LIST_EMPTY (in empty list node can't be deleted)
2. List not empty → Update the head with next node address, delete the first node.

Example :

head → 10 → 20 → 30 → 40 → 50 → NULL

head 10 → 20 → 30 → 40 → 50 → NULL

head → 20 → 30 → 40 → 50 → NULL

4. delete_last

Input : head → pointer to first node

Cases : List empty
List not empty

1. List empty → Return LIST_EMPTY (in empty list node can't be deleted)
2. List not empty → Traverse to the last node, update the previous node address and delete the last node.

5. delete_list

Input : head → pointer to first node

Cases : List empty
List not empty

1. List empty → Return LIST_EMPTY (in empty list node can't be deleted)
2. List not empty → Delete all nodes one by one.

NOTE : Should not update head directly with NULL without freeing the nodes.

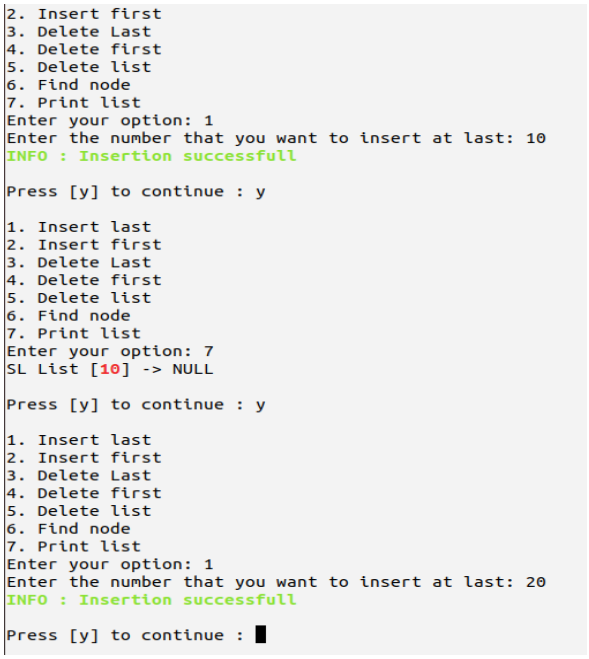
6. find_node

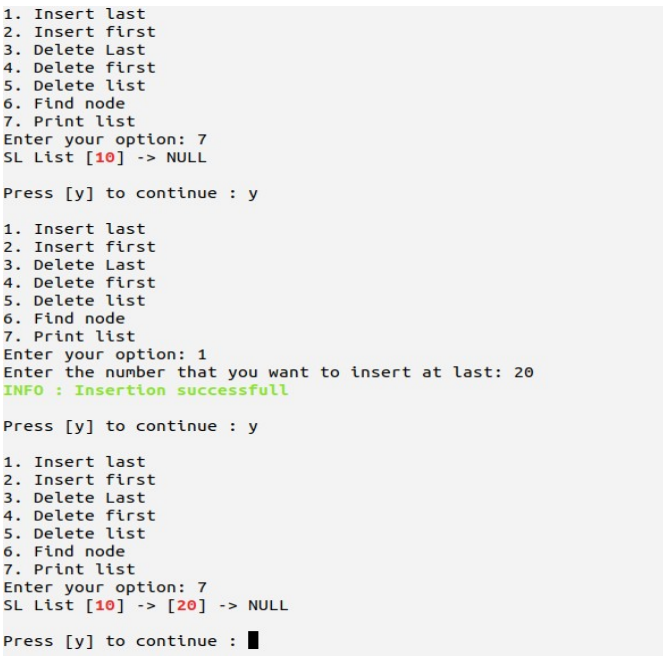
Inputs : head → pointer to first node.
data → data to be found in the list.

Cases : List empty
List not empty
1. Data found
2. Data not found

1. List empty → Return LIST_EMPTY (in empty list can't search data)
2. List not empty
    1. Traverse through the list to search the data
    2. If data found return DATA_FOUND
3. Else return DATA_NOT_FOUND

Output Images :





```
1. Insert last
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 2
Enter the element you have to insert at the first : 5
INFO : Insertion successfull

Press [y] to continue : y

1. Insert last
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 7
SL List [5] -> [10] -> [20] -> NULL

Press [y] to continue :
```

```
1. Insert last
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 3
INFO : Delete last Successfull

Press [y] to continue : y

1. Insert last
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 7
SL List [5] -> [10] -> NULL

Press [y] to continue :
```

```
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 1
Enter the number that you want to insert at last: 30
INFO : Insertion successfull

Press [y] to continue : y

1. Insert last
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 1
Enter the number that you want to insert at last: 40
INFO : Insertion successfull

Press [y] to continue : y

1. Insert last
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 7
SL List [5] -> [10] -> [20] -> [30] -> [40] -> NULL

Press [y] to continue :
```

```
1. Insert last
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 4
INFO : Delete first Successfull

Press [y] to continue : y

1. Insert last
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 7
SL List [10] -> [20] -> [30] -> [40] -> NULL

Press [y] to continue :
```

```
SL List [10] -> [20] -> [30] -> [40] -> NULL

Press [y] to continue : y

1. Insert last
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 6
Enter the key to find : 20
20 found in the list

Press [y] to continue :
```

```
SL List [10] -> [20] -> [30] -> [40] -> NULL

Press [y] to continue : y

1. Insert last
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 6
Enter the key to find : 23
List is empty or Key not found

Press [y] to continue :
```

```
SL List [10] -> [20] -> [30] -> [40] -> N

Press [y] to continue : y

1. Insert last
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 5
INFO : Delete list Successfull

Press [y] to continue : y

1. Insert last
2. Insert first
3. Delete Last
4. Delete first
5. Delete list
6. Find node
7. Print list
Enter your option: 7
List is Empty !!!!
```

Assignment – 2

Implement the functions given below.

1. insert_after(head, gdata, ndata)
2. insert_before(head, gdata, ndata)
3. delete_element(head, gdata)
4. insert_Nth(head, ndata, n)

1. insert_after :

Input : head → pointer to first node.
gdata → given data (should be present in the list).
ndata → data to be inserted in the list after gdata.

Cases : List Empty
List Non-Empty
1.Given data present
2.Given data not present

Case-1	**List Empty**		
Input	Head = NULL		
Output	return LIST_EMPTY		
Case-2 Sub-case:1	**List Non-Empty Given data present**	**Case-2 Sub-case:2**	**List Non-Empty Given data not present**
Example	If given_data = 40, new_data = 45	Example	If given_data = 60, new_data = 45
Input	10 → 20 → 30 → 40 → 50	Input	10 → 20 → 30 → 40 → 50
Output	10 → 20 → 30 → 40 → 45 → 50	Output	Return DATA_NOT_FOUND
Prototype	**int sl_insert_after(Slist *head, data_t g_data, data_t n_data);** head : Pointer to the first node g_data : Given data n_data : New data to be inserted into the list return value : status (LIST_EMPTY, SUCCESS, DATA_NOT_FOUND)		

2. insert_before :

Cases : List Empty
List Non-Empty
1.Given data present
2.Given data not present

Case-1	**List Empty**		
Input	Head = NULL		
Output	return LIST_EMPTY		
Case-2 Sub-case:1	**List Non-Empty Given data present**	**Case-2 Sub-case:2**	**List Non-Empty Given data not present**
Example	If given_data = 40, new_data = 45	Example	If given_data = 60, new_data = 45
Input	10 → 20 → 30 → 40 → 50	Input	10 → 20 → 30 → 40 → 50
Output	10 → 20 → 30 → 45 → 40 → 50	Output	Return DATA_NOT_FOUND

Case-3	**Data found in the first**
Example	If given_data = 10, new_data = 45
Input	10 → 20 → 30 → 40 → 50
Output	45 → 10 → 20 → 30 → 40 → 50
Prototype	**int sl_insert_after(Slist **head, data_t g_data, data_t n_data);** head : Pointer to the first node g_data : Given data n_data : New data to be inserted into the list return value : status (LIST_EMPTY, SUCCESS, DATA_NOT_FOUND)

3. delete_element :

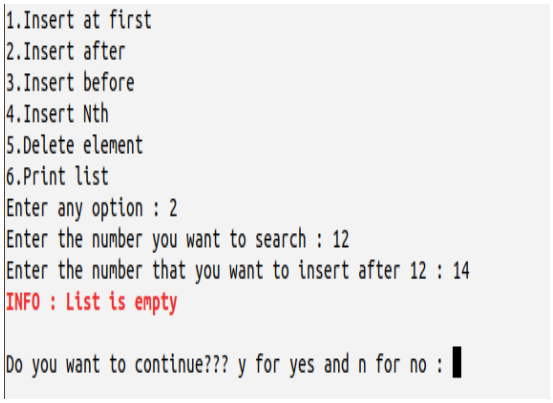
Description	**Write a function to delete the given data in the single linked list.**		
Cases	1. List Empty 2. List Non-Empty 1. Given data present 2. Given data not present		
Case-1	**List Empty**		
Input	Head = NULL		
Output	return LIST_EMPTY		
Case-2 Sub-case:1	**List Non-Empty Given data present**	**Case-2 Sub-case:2**	**List Non-Empty Given data not present**
Example	If given_data = 40	Example	If given_data = 60
Input	10 → 20 → 30 → 40 → 50	Input	10 → 20 → 30 → 40 → 50
Output	10 → 20 → 30 → 50	Output	Return DATA_NOT_FOUND
Prototype	**int sl_delete_element(Slist **head, data_t g_data);** head : Pointer to the first node g_data : Given data return value : status (LIST_EMPTY, SUCCESS, DATA_NOT_FOUND)		

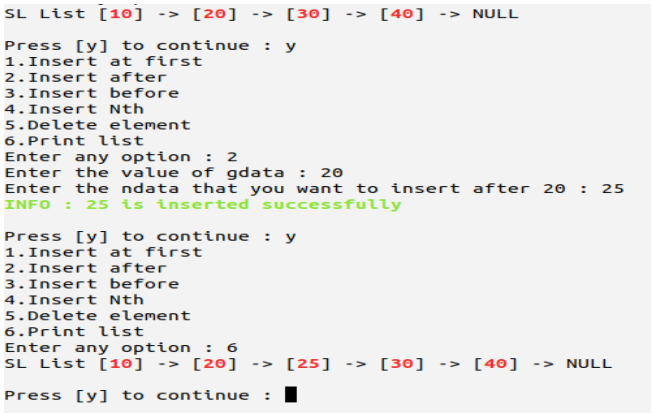
4. insert_Nth :

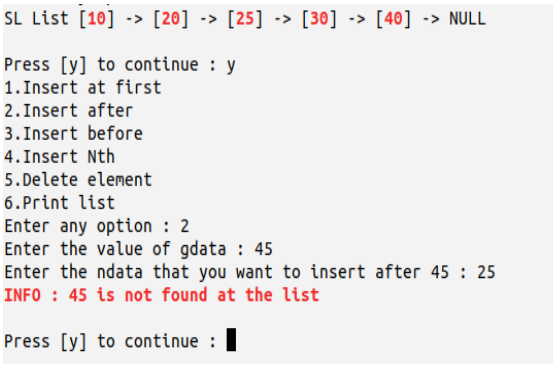
Description	**Write a function to insert the given data exactly at the ‘n’ position in the single linked list.**		
Cases	1. List Empty 2. List Non-Empty 1. Given ‘n’th position present 2. Given ‘n’th position not present		
Case-1	**List Empty**		
Input	Head = NULL		
Output	return LIST_EMPTY		
Case-2 Sub-case:1	**List Non-Empty Given data present**	**Case-2 Sub-case:2**	**List Non-Empty Given data not present**
Example	If n = 3, n_data = 23	Example	If n = 10, n_data = 23
Input	10 → 20 → 30 → 40 → 50	Input	10 → 20 → 30 → 40 → 50
Output	10 → 20 → 23 → 30 → 50	Output	Return POSITION_NOT_FOUND

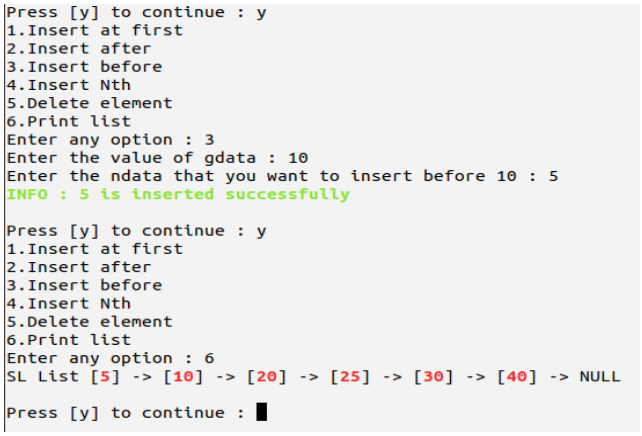
Prototype	**int sl_insert_Nth(Slist **head, data_t ndata, int n);** head : Pointer to the first node n : Position number ndata : New data, to be inserted into the list return value : status (LIST_EMPTY, SUCCESS, POSITION_NOT_FOUND)

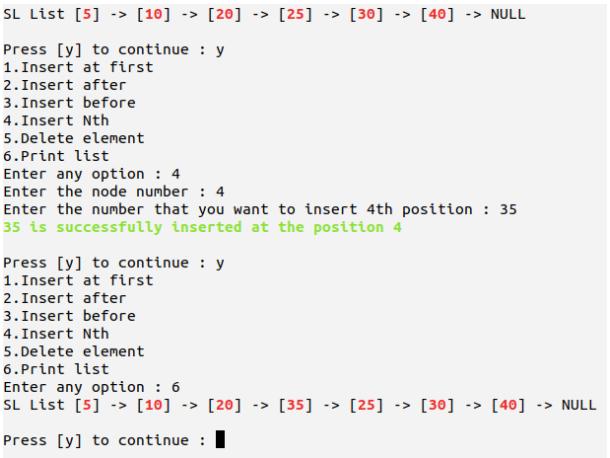
Output Images :

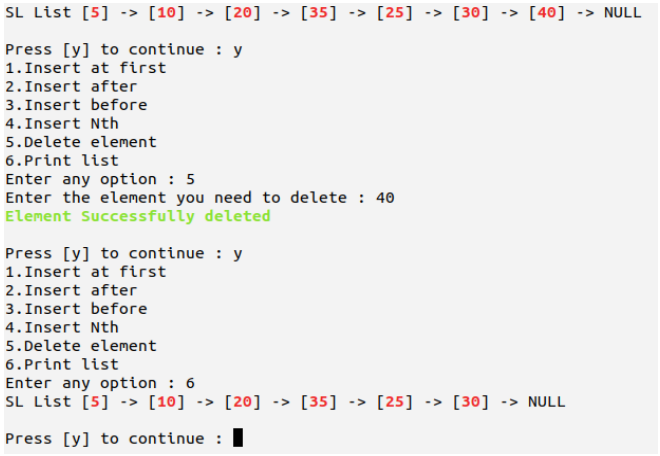












Assignment – 3

Implement functions given below.

1. dll_insert_first(head, tail, data)
2. dll_insert_last(head, tail, data)
3. dll_delete_first(head, tail)
4. dll_delete_last(head, tail)
5. dll_delete_list(head, tail)

1. dll_insert_at_first :

Input : head → pointer to first node.
tail → pointer to last node
data → data to be inserted.

Read data from user and insert the given data in first position.

Cases : **List empty, data = 10**

head = NULL, tail = NULL , After inserting data into list, list contains
head → 10 ← **tail**
head and tail should be updated with new node address

List not empty, data 40

head → 10 ↔ 20 ↔ 30 ← **tail**, After inserting 40 into list

head → 40 ↔ 10 ↔ 20 ↔ 30 ← **tail**

2. dll_Insert_at_last :

Input : head → pointer to first node
data → data to be inserted at the end.

Cases : List empty
List not empty

1. List empty – Update the head and tail with new node address.
2. List not empty – Establish the link between last node and new node using tail pointer.

3. dll_delete_first :

Input : head → pointer to the first node.
Tail → pointer to last node

Cases : List empty
List not empty

1. List empty → Return LIST_EMPTY (in empty list node can't be deleted)
2. List not empty → Update the head with next node address, delete the first node.

Example :

head → 10 → 20 → 30 → 40 → 50 → NULL

head 10 → 20 → 30 → 40 → 50 → NULL

head → 20 → 30 → 40 → 50 → NULL

4. dll_delete_last

Input : head → pointer to first node
tail → pointer to last node

Cases : List empty
List not empty

1. List empty → Return LIST_EMPTY (in empty list node can't be deleted)
2. List not empty → Update the previous node address in tail pointer, and update link of previous node and delete the last node.

5. delete_list

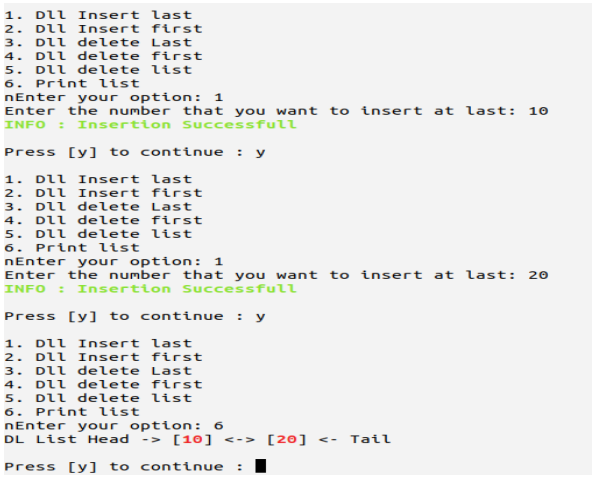
Input : head → pointer to first node
tail → pointer to last node

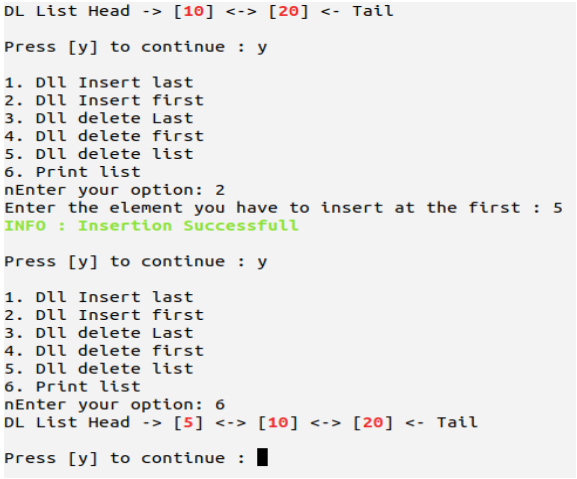
Cases : List empty
List not empty

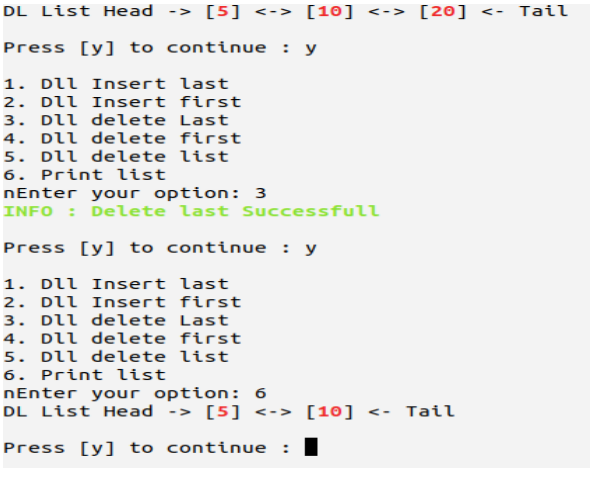
1. List empty → Return LIST_EMPTY (in empty list node can't be deleted)
2. List not empty → Delete all nodes one by one from head or tail.

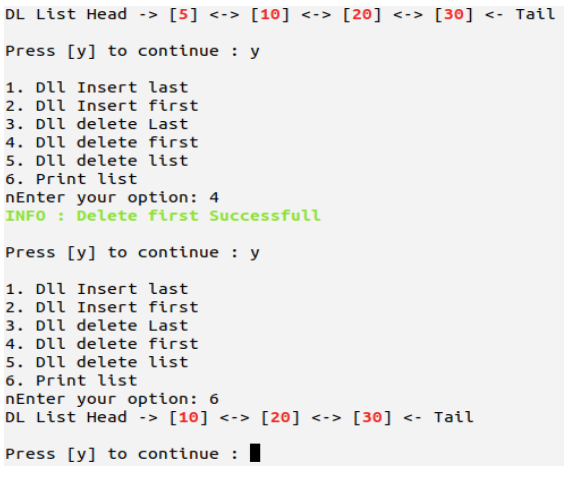
NOTE : Should not update head and tail directly with NULL without freeing the nodes.

Output Images :









```
DL List Head -> [10] <-> [20] <-> [30] <- Tail

Press [y] to continue : y

1. Dll Insert last
2. Dll Insert first
3. Dll delete Last
4. Dll delete first
5. Dll delete list
6. Print list
nEnter your option: 5
INFO : Delete list Successfull

Press [y] to continue : y

1. Dll Insert last
2. Dll Insert first
3. Dll delete Last
4. Dll delete first
5. Dll delete list
6. Print list
nEnter your option: 6
List is Empty !!!!

Press [y] to continue :
```

Assignment – 4

Implement functions given below :

1. dll_insert_after(head, tail, gdata, ndata)
2. dll_insert_before(head, gdata, ndata)
3. dll_delete_element(head, tail, gdata)

1. dll_insert_after :

Description	**Write a function to insert the new data after the given data.**		
Cases	1. List Empty 2. List Non-Empty 1. Given data present 2. Given data not present		
Case-1	**List Empty**		
Input	head = NULL, tail = NULL		
Output	return LIST_EMPTY		
Case-2 Sub-case:1	**List Non-Empty Given data present**	**Case-2 Sub-case:2**	**List Non-Empty Given data not present**
Example	If given_data = 40, new_data = 45	Example	If given_data = 60, new_data = 45
Input	10 ↔ 20 ↔ 30 ↔ 40 ↔ 50	Input	10 ↔ 20 ↔ 30 ↔ 40 ↔ 50
Output	10 ↔ 20 ↔ 30 ↔ 40 ↔ 45 ↔ 50	Output	Return DATA_NOT_FOUND
Prototype	**int dl_insert_after(Dlist *head, Dlist *tail, data_t gdata, data_t ndata);** head : Pointer to the first node tail : Pointer to the last node g_data : Given data n_data : New data to be inserted into the list return value : status (LIST_EMPTY, SUCCESS, DATA_NOT_FOUND)		

2. dll_insert_before

Description	**Write a function to insert the new data before the given data.**
Cases	1. List Empty

	2. List Non-Empty 1. Given data present 2. Given data not present
Case-1	**List Empty**
Input	Head = NULL
Output	return LIST_EMPTY

Case-2 Sub-case:1	**List Non-Empty Given data present**	**Case-2 Sub-case:2**	**List Non-Empty Given data not present**
Example	If given_data = 40, new_data = 45	Example	If given_data = 60, new_data = 45
Input	10 ↔ 20 ↔ 30 ↔ 40 ↔ 50	Input	10 ↔ 20 ↔ 30 ↔ 40 ↔ 50
Output	10 ↔ 20 ↔ 30 ↔ 45 ↔ 40 ↔ 50	Output	Return DATA_NOT_FOUND

Case-3	**Data found in the first**
Example	If given_data = 10, new_data = 45
Input	10 ↔ 20 ↔ 30 ↔ 40 ↔ 50
Output	45 ↔ 10 ↔ 20 ↔ 30 ↔ 40 ↔ 50
Prototype	**int dl_insert_before(Dlist **head, data_t gdata, data_t ndata);** head : Pointer to the first node tail : Pointer to the last node gdata : Given data ndata : New data to be inserted into the list return value : status (LIST_EMPTY, SUCCESS, DATA_NOT_FOUND)

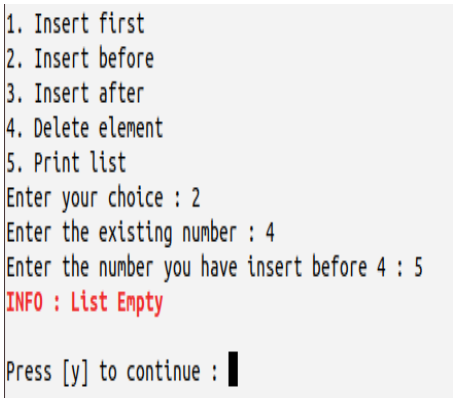
3. dll_delete_element

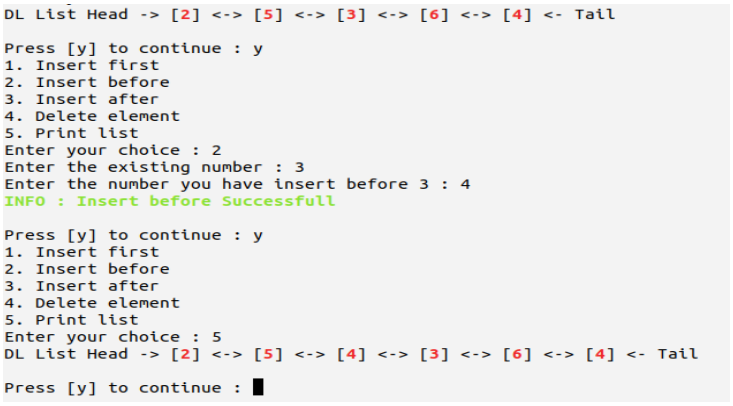
Description	**Write a function to delete the given data in the double linked list.**
Cases	1. List Empty 2. List Non-Empty 1. Given data present 2. Given data not present
Case-1	**List Empty**
Input	head = NULL, tail = NULL
Output	return LIST_EMPTY

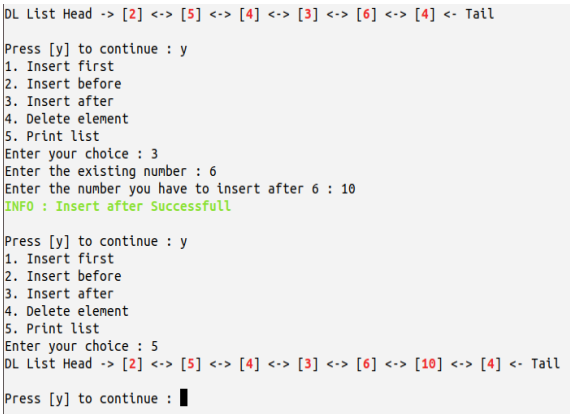
Case-2 Sub-case:1	**List Non-Empty Given data present**	**Case-2 Sub-case:2**	**List Non-Empty Given data not present**
Example	If given_data = 40	Example	If given_data = 60
Input	10 ↔ 20 ↔ 30 ↔ 40 ↔ 50	Input	10 ↔ 20 ↔ 30 ↔ 40 ↔ 50
Output	10 ↔ 20 ↔ 30 ↔ 50	Output	Return DATA_NOT_FOUND

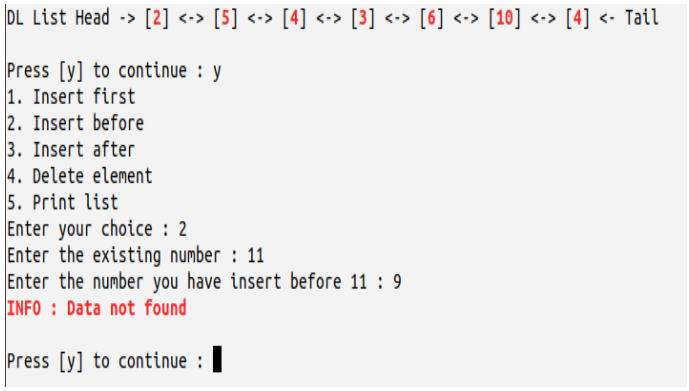
Prototype	**int dl_delete_element(Dlist **head, Dlist **tail, data_t g_data);** head : Pointer to the first node tail : Pointer to the last node g_data : Given data return value : status (LIST_EMPTY, SUCCESS, DATA_NOT_FOUND)

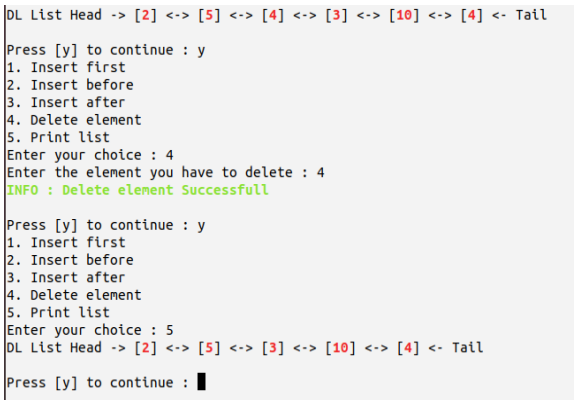
Output Images :

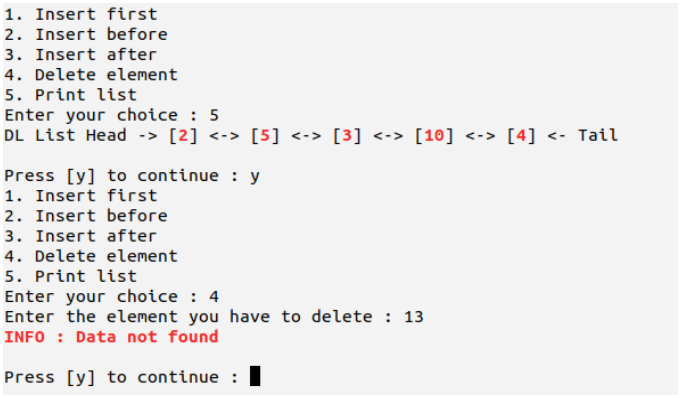












Assignment – 5

Implement functions given below :

1. sl_find_mid(head, mid)
2. sl_get_Nth_last(head, n, data)

1. sl_find_mid

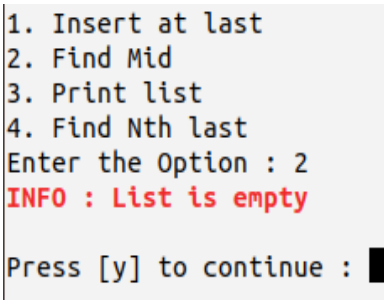
Cases	1. List Empty 2. List Non-Empty
Case-1	**List Empty**
Input	Head = NULL
Output	return LIST_EMPTY

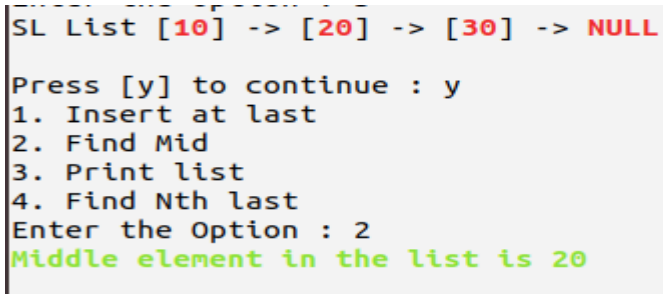
Case-2 Sub-case:1	**List Non-Empty List contains Odd nodes**	**Case-2 Sub-case:2**	**List Non-Empty List contains Even nodes**
Input	10 -> 20 -> 30 -> 40 -> 50	Input	10 -> 20 -> 40 -> 50
Output	30, return SUCCESS	Output	20 (or) 40, return SUCCESS
Prototype	**int sl_find_mid(Slist *head, int * mid);** head : Pointer to the first node mid : Pointer to the mid node data return value : status (LIST_EMPTY, SUCCESS)		
Note	**Traverse the linked list only once**		

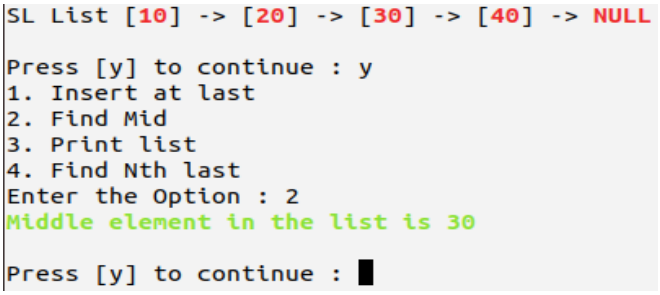
2. sl_get_Nth_last

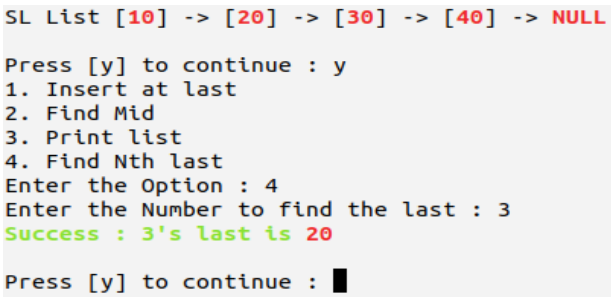
Cases	1. List Empty 2. List Non-Empty
Case-1	**List Empty**
Input	Head = NULL
Output	return LIST_EMPTY
Case-2	**List Non-Empty**
Input	10 -> 20 -> 30 -> 40 -> 50, n = 2
Output	40 (From the last, second node conatins the data 40)
Prototype	**int sl_get_nth_from_last(Slist *head, int n, int *data);** head : Pointer to the first node data : Pointer to Nth last node data n : Position from the last node return value : status (LIST_EMPTY, SUCCESS, POSITION NOT FOUND)
Note	**Traverse the linked list only once**

Output Images :









```
Press [y] to continue : y
1. Insert at last
2. Find Mid
3. Print list
4. Find Nth last
Enter the Option : 4
Enter the Number to find the last :
5
INFO : Position not found

Press [y] to continue : 
```

Assignment – 6

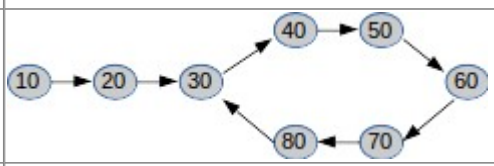
Implement functions given below :

1. insert_sorted(head, ndata)
2. find_loop(head)

1. insert_sorted

Cases	1. List Empty 2. List Non-Empty
Case-1	**List Empty**
Input	Head = NULL, ndata = 10
Output	10
Case-2	**List Non-Empty**
Example	If ndata = 45
Input	10 → 20 → 30 → 40 → 50
Output	10 → 20 → 30 → 40 → 45 → 50
Prototype	**int sl_insert_sorted(Slist **head, data_t n_data);** head : Pointer to the first node ndata : New data to be inserted into the sorted list return value : status (SUCCESS, FAILURE)

2. sl_find_loop

Cases	1. List Empty 2. List Non-Empty
Case-1	**List Empty**
Input	Head = NULL
Output	return LIST_EMPTY
Case-2 Sub-Case:1	**List Non-Empty**
Input	
Output	Loop detected.
Case-2 Sub-Case:2	**List Non-Empty**

Input		
Output	Loop not detected.	
Prototype	**int sl_find_loop(Slist *head);** head : Pointer to the first node return value : status (LIST_EMPTY, LOOP_DETECTED, LOOP_NOT_DETECTED)	
Note	**Traverse the linked list only once**	

Output Images :

```
1. Insert at last
2. Print list
3. Insert sorted
4. Find loop
5. Create loop
Enter Your choice : 3
Enter the data to be inserted : 10
INFO : Insert sorted success

Press [y] to continue : y
1. Insert at last
2. Print list
3. Insert sorted
4. Find loop
5. Create loop
Enter Your choice : 2
SL List [10] -> NULL

Press [y] to continue :
```

```
Enter Your choice : 3
Enter the data to be inserted : 5
INFO : Insert sorted success

Press [y] to continue : y
1. Insert at last
2. Print list
3. Insert sorted
4. Find loop
5. Create loop
Enter Your choice : 3
Enter the data to be inserted : 2
INFO : Insert sorted success

Press [y] to continue : y
1. Insert at last
2. Print list
3. Insert sorted
4. Find loop
5. Create loop
Enter Your choice : 2
SL List [2] -> [5] -> [10] -> NULL

Press [y] to continue :
```

```
SL List [1] -> [2] -> [4] -> [5] -> [10] -> NULL

Press [y] to continue : y
1. Insert at last
2. Print list
3. Insert sorted
4. Find loop
5. Create loop
Enter Your choice : 4
INFO : Loop is not found

Press [y] to continue :
```

```
1. Insert at last
2. Print list
3. Insert sorted
4. Find loop
5. Create loop
Enter Your choice : 5
Enter a data to create the loop : 4
INFO : Loop created successfully

Press [y] to continue : y
1. Insert at last
2. Print list
3. Insert sorted
4. Find loop
5. Create loop
Enter Your choice : 4
INFO : Loop found

Press [y] to continue :
```

Assignment – 7

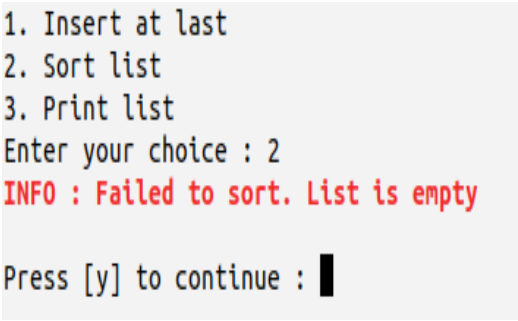
Implement functions given below :

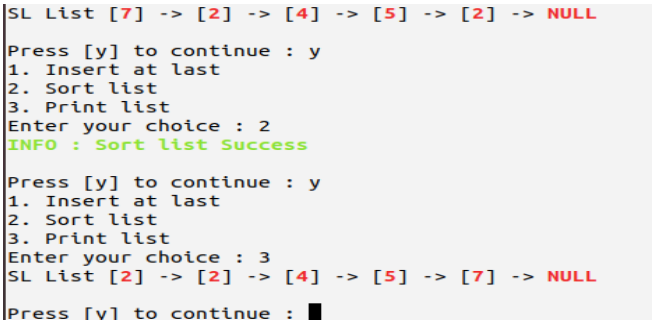
1.sl_sort(head)

Cases	1. List Empty 2. List Non-Empty
Case-1	**List Empty**
Input	Head = NULL
Output	return LIST_EMPTY
Case-2	**List Non-Empty**

Input	50 → 40 → 30 → 20 → 10
Output	10 → 20 → 30 → 40 → 50
Prototype	**int sl_sort(Slist **head);** head : Pointer to the first node return value : status (LIST_EMPTY, SUCCESS)
Note	**Don't swap the data present in the nodes, swap the nodes itself.**

Output Images :





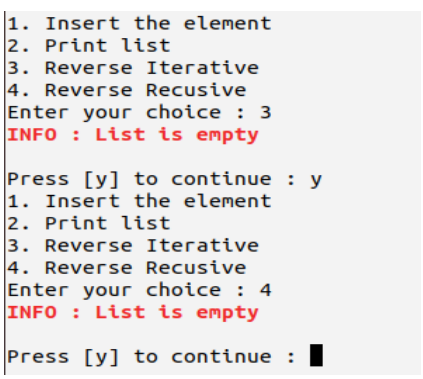
Assignment – 8

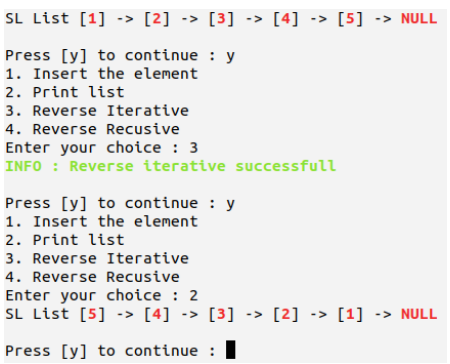
Implement functions given below :

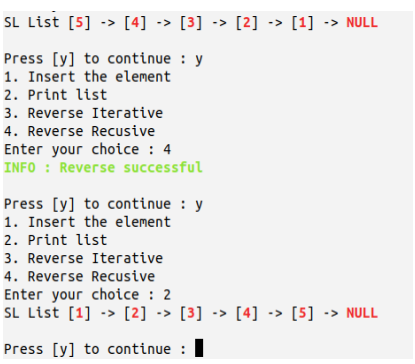
1. sl_reverse_iterative(head)
2. sl_reverse_recursive(head)

Cases	1. List Empty 2. List Non-Empty
Case-1	**List Empty**
Input	Head = NULL
Output	return LIST_EMPTY
Case-2	**List Non-Empty**
Input	50 → 40 → 30 → 20 → 10
Output	10 → 20 → 30 → 40 → 50
Prototype	**int sl_reverse_iterative/recursive(Slist **head);** head : Pointer to the first node return value : status (LIST_EMPTY, SUCCESS)

Output Images :







Assignment – 9

Write a function to remove the duplicate values present in the SLL.

Cases	1. List Empty 2. List Non-Empty
Case-1	**List Empty**
Input	Head = NULL
Output	return LIST_EMPTY
Case-2	**List Non-Empty**
Input	5 → 3 → 4 → 5 → 2 → 1 → 4 → 5 → 3
Output	5 → 3 → 4 → 2 → 1
Prototype	**int sl_remove_duplicates(Slist **head);** head : Pointer to the first node return value : status (LIST_EMPTY, SUCCESS)

NOTE : Don't sort the list

Sample Output :

SL List → 1 → 2 → 4 → 2 → 5 → 4 → 3 → 1 → 2

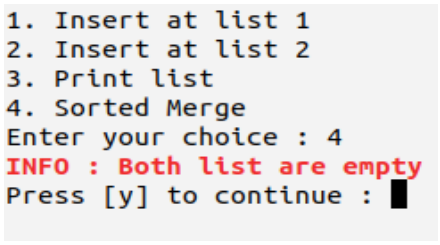
Output → 1 → 2 → 4 → 5 → 3

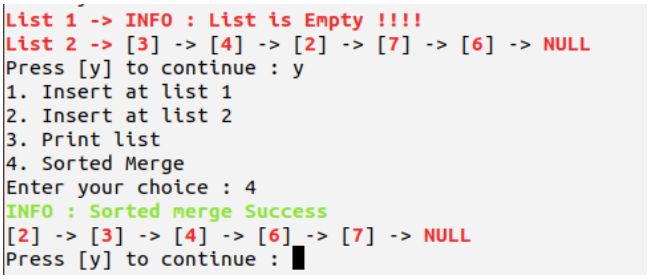
Assignment – 10

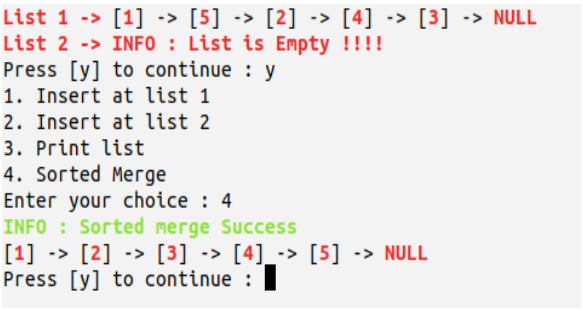
WAF to merge and sort two linked list.

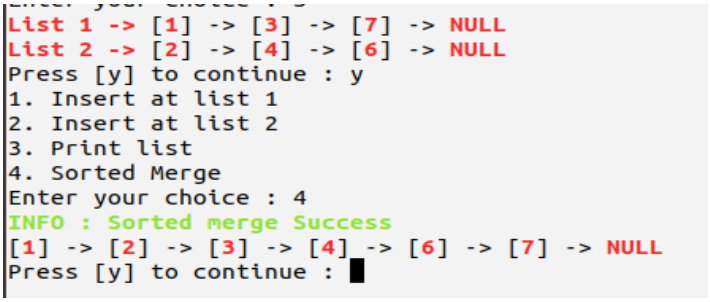
Cases	1. List Empty 2. List Non-Empty
Case-1	**List Empty**
Input	Head = NULL
Output	return LIST_EMPTY
Case-2	**List Non-Empty**
Input	List – 1 30 → 10 → 50 List – 2 20 → 5 → 35
Output	5 → 10 → 20 → 30 → 35 → 50
Prototype	**int sl_sorted_merge(Slist **head1, Slist **head2);** head1 : Pointer to the first node of the first linked list head2 : Pointer to the first node of the second linked list return value : status (LISTS_EMPTY, SUCCESS)
Note	1. If second list is EMPTY, no need to append it. 2. If first list is EMPTY, update head1 to the head2 3. If both list are EMPTY, return LISTS_EMPTY 4. Finally sort the list

Output Images :









Assignment – 11 & 12

Implement the stack using array and linked list:

1. push(stack, data)
2. pop(stack)
3. peek(stack, data)
4. peep(stack)

1. Push (insert)

Inputs : stack → Pointer that contains address of structure variable (array)

stack → Pointer to the first node (LL)

data → Data to be inserted into stack

Cases :

1. Stack Full → Return STACK_FULL
2. Stack not full → Push the given data into stack

2. Pop (delete)

Inputs : stack → Pointer that contains address of structure variable (array)
stack → Pointer to the first node (LL)

Cases :

1. Stack empty → Return STACK_EMPTY
2. Stack not empty → Pop (delete) the top of the data from the stack.

3. Peek int peek(stack_t *stk, int *data);

Inputs : stack → pointer that contains address of the structure variable (array)
stack → Pointer to the first node (LL).
Data → integer pointer

Cases :

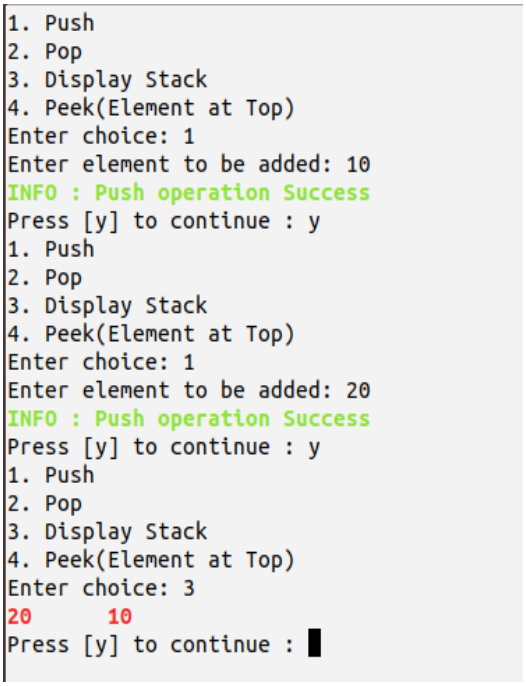
1. Stack empty → Return STACK_EMPTY
2. Stack not empty → Store the top most value in pointer variable and return SUCCESS.

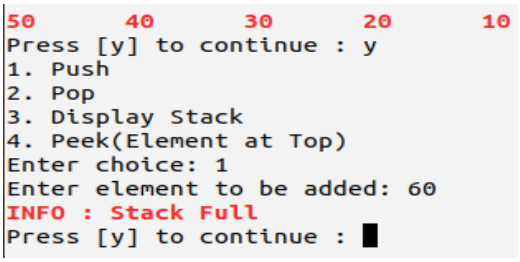
Peep :

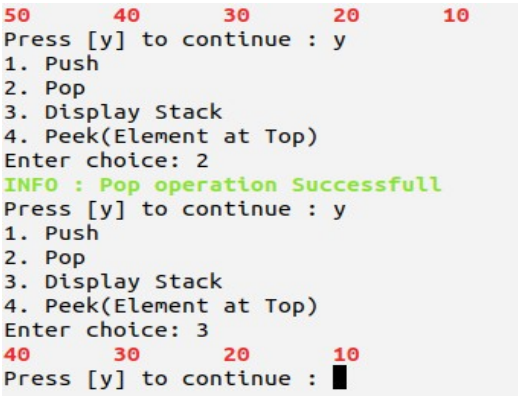
Input : Stack → Pointer that holds address of structure variable (array)
stack → Pointer to the first node (LL)
Output : Print all the data's stored in stack.
Cases :

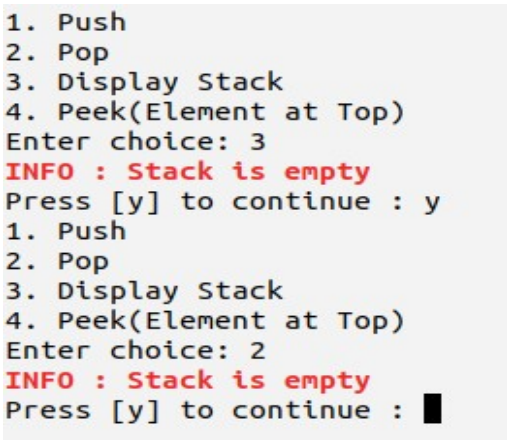
1. Stack empty → Return STACK_EMPTY
2. Stack not empty → print the elements in the stack.

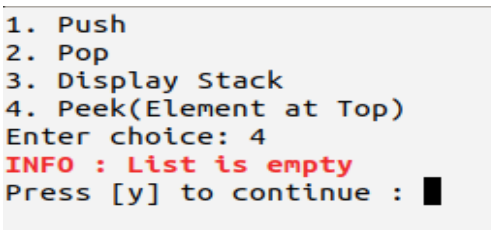
Output Images :

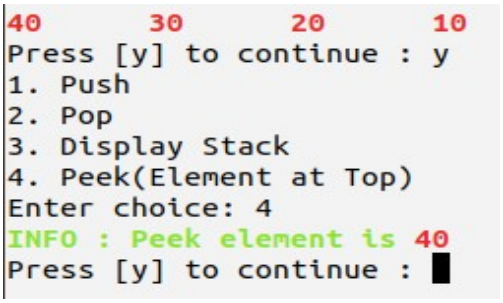


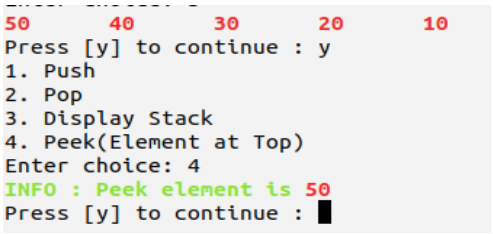












Assignment – 13

Implement given below functions.

1. Infix – Postfix converstion
2. Postfix evaluation

1. Infix – Postfix converstion

Title	Infix to Postfix
Filename	infix_postfix.c
Description	**Write a function to convert the given infix expression into the postfix form**
Cases	1. Single digit numbers 2. Multiple digit numbers
Case-1	**Single digit numbers**
Input	2 * 3 – 3 + 8 / 4 / (1 + 1)
Output	2 3 * 3 – 8 4 / 1 1 + / +
Case-2	**Multiple digit numbers**
Input	2 * 30 – 3 + 8 / 4 / (10 + 1)
Output	2, 30, *, 3, –, 8, 4, /, 10, 1, +, / ,+
Sample Prototype	**Char * infix_postfix(char *infix);** infix : Pointer the base address of the infix array return : Pointer the base address of the postfix array

2. Postfix evaluation

Title	Postfix Evaluation
Filename	postfix_eval.c
Description	**Write a function to evaluate the postfix expression.**
Cases	1. Single digit numbers 2. Multiple digit numbers
Case-1	**Single digit numbers**
Input	2 3 * 3 – 8 4 / 1 1 + / +
Output	4
Case-2	**Multiple digit numbers**
Input	2, 30, *, 3, –, 8, 4, /, 10, 1, +, / ,+
Output	57.181
Sample Prototype	**double postfix_evaluation(char *postfix);** Postfix : Pointer the base address of the postfix array return : Result of the expression in double.

Output Images :

```
1. Infix-Postfix Converstion
2. Postfix Evaluation
Enter your choice : 1
Enter the Infix expression : 1+2*3
PostFix expression : 123*+
Press [y] to continue : y
1. Infix-Postfix Converstion
2. Postfix Evaluation
Enter your choice : 2
Enter Postfix expression (enter only digits and operators) : 123*+

Result : 7
Press [y] to continue :
```

```
1. Infix-Postfix Converstion
2. Postfix Evaluation
Enter your choice : 1
Enter the Infix expression : 2*3-3+8/4/(1+1)
PostFix expression : 23*3-84/11+/+
Press [y] to continue : y
1. Infix-Postfix Converstion
2. Postfix Evaluation
Enter your choice : 2
Enter Postfix expression (enter only digits and operators) : 23*3-84/11+/+

Result : 4
Press [y] to continue :
```

Assignment – 14

Implement given below functions.

1. Infix – Prefix converstion
2. Prefix evaluation

1. Infix – Prefix converstion

Title	Infix to Prefix
Filename	infix_prefix.c
Description	**Write a function to convert the given infix expression into the prefix form**
Cases	1. Single digit numbers 2. Multiple digit numbers
Case-1	**Single digit numbers**
Input	2 * 3 – 3 + 8 / 4 / (1 + 1)
Output	+ – * 2 3 3 / / 8 4 + 1 1

Case-2	**Multiple digit numbers**
Input	2 * 30 – 3 + 8 / 4 / (10 + 1)
Output	+, -, *, 2, 30, 3, /, /, 8, 4, +, 10, 1
Sample Prototype	**Char * infix_prefix(char *infix);** infix : Pointer the base address of the infix array return : Pointer the base address of the prefix array

2. Prefix Evaluation

Title	Prefix Evaluation
Filename	prefix_eval.c
Description	**Write a function to evaluate the prefix expression.**
Cases	1. Single digit numbers 2. Multiple digit numbers
Case-1	**Single digit numbers**
Input	+ – * 2 3 3 / / 8 4 + 1 1
Output	4
Case-2	**Multiple digit numbers**
Input	+, -, *, 2, 30, 3, /, /, 8, 4, +, 10, 1
Output	57.181
Sample Prototype	**double prefix_evaluation(char *prefix);** Prefix : Pointer the base address of the prefix array return : Result of the expression in double.

Output Images :

```
1. Infix-Prefix Converstion
2. Prefix Evaluation
Enter your choice : 1
Enter the Infix expression : 1+2*3
PreFix expression : +1*23
Press [y] to continue : y
1. Infix-Prefix Converstion
2. Prefix Evaluation
Enter your choice : 2
Enter Prefix expression (enter only digits and operators) : +1*23

Result : 7
Press [y] to continue :
```

```
1. Infix-Prefix Converstion
2. Prefix Evaluation
Enter your choice : 1
Enter the Infix expression : 2*3-3+8/4/(1+1)
PreFix expression : +-*233//84+11
Press [y] to continue : y
1. Infix-Prefix Converstion
2. Prefix Evaluation
Enter your choice : 2
Enter Prefix expression (enter only digits and operators) : +-*233//84+11

Result : 4
```

Assignment – 15 & 16 (Circular queue)

Implement functions given below :

1. Enqueue(queue, data) → array, Enqueue(front, rear, data) → LL implementation
2. Dequeue(queue) → array, Dequeue(front, rear) → LL implementation
3. Front_data(queue) → array, front_data(front) → LL implementation
4. Print_queue(queue) → array, Print_queue(front) → LL implementation

1. Enqueue :

Inputs : queue → Pointer contains structure variable (Array).
Front → pointer to first node (LL)
rear → pointer to last node (LL)
data → Data to be added.

Cases : Array Implementation

1. Queue Full → Return QUEUE_FULL (for LL no need to check this condition).
2. Queue not full → Add data into queue.

2. Dequeue :

Inputs : queue → Pointer contains structure variable (Array).
Front → pointer to first node (LL)
rear → pointer to last node (LL)

Cases : Array & LL implementation

1. Queue Empty → Return QUEUE_EMPTY.
2. Queue Not Empty → Delete the data from the front end (front++ in array, free node in LL).

3. Front_data :

Inputs : queue → Pointer contains structure variable (Array).
Front → pointer to first node (LL)
rear → pointer to last node (LL)

Cases : Array & LL implementation

1. Queue Empty → Return QUEUE_EMPTY.
2. Queue Not Empty → Return Front data

Output Images :

```
1. Enqueue
2. Dequeue
3. Front-data
4. Print Queue
Enter the option : 4
Queue is empty
Press [y] to continue : y
1. Enqueue
2. Dequeue
3. Front-data
4. Print Queue
Enter the option : 1
Enter the element you want to insert : 10
INFO : Enqueue Success
Press [y] to continue : y
1. Enqueue
2. Dequeue
3. Front-data
4. Print Queue
Enter the option : 4
Front -> 10  <- Rear
Press [y] to continue :
```

```
Enter the element you want to insert : 20
INFO : Enqueue Success
Press [y] to continue : y
1. Enqueue
2. Dequeue
3. Front-data
4. Print Queue
Enter the option : 1
Enter the element you want to insert : 30
INFO : Enqueue Success
Press [y] to continue : y
1. Enqueue
2. Dequeue
3. Front-data
4. Print Queue
Enter the option : 1
Enter the element you want to insert : 40
INFO : Enqueue Success
Press [y] to continue : y
1. Enqueue
2. Dequeue
3. Front-data
4. Print Queue
Enter the option : 4
Front -> 10  20  30  40  <- Rear
Press [y] to continue :
```

```
Front -> 10  20  30  40  <- Rear
Press [y] to continue : y
1. Enqueue
2. Dequeue
3. Front-data
4. Print Queue
Enter the option : 2
INFO : Dequeue Sucess
Press [y] to continue : y
1. Enqueue
2. Dequeue
3. Front-data
4. Print Queue
Enter the option : 4
Front -> 20  30  40  <- Rear
Press [y] to continue :
```

```
Front -> 20  30  40  <- Rear
Press [y] to continue : y
1. Enqueue
2. Dequeue
3. Front-data
4. Print Queue
Enter the option : 3
INFO : Success
The front data in the queue is 20
Press [y] to continue :
```

```
Queue is empty
Press [y] to continue : y
1. Enqueue
2. Dequeue
3. Front-data
4. Print Queue
Enter the option : 2
INFO : Queue is empty
Press [y] to continue :
```

```
INFO : Queue is empty
Press [y] to continue : y
1. Enqueue
2. Dequeue
3. Front-data
4. Print Queue
Enter the option : 3
INFO : Queue is empty
Press [y] to continue :
```

```
Front -> 10  20  30  40  50  60  70  80  90  100  <- Rear
Press [y] to continue : y
1. Enqueue
2. Dequeue
3. Front-data
4. Print Queue
Enter the option : 1
Enter the element you want to insert : 110
INFO : Queue is full
Press [y] to continue :
```

Assignment – 17

Write a binary search function in both iterative and recursive methods

Objective :

To understand the working of Binary Search

Requirements :

1.Prompt the user to Enter the size of the Array and read it
2.Declare an integer array and read the elements of array
3.Prompt the user to Enter the key element to search and read it
4.Call the binary search function to search the element

Sample Prototype	**int binary_search_iterative(int *ptr , int size ,int key)** ptr Base address of the integer array size Length of the array key Element to be searched return Integer value **int binary_Search_recursive(int *ptr , int size,int key,int low,int high)** ptr Base address of the integer array size Length of the array key Element to be searched low Starting index high Ending index return Integer

Sample Output :

Case 1 : Data is present in the array

./a.out

Enter the size of array : 5
Elements of array :
1 2 3 4 5
Enter the key to search = 3
Key element 3 is found at 3rd position

Case 2 : Data is not present in the array

./a.out

Enter the size of array : 5
Elements of array :
1 2 3 4 5
Enter the key to search = 25
Key element 25 is not present in array

Assignment – 18

WAF to sort given array using bubble sort, insertion sort and selection sort

Objective :

To understand the working of Sorting Techniques

Requirements :

1.Prompt the user to Enter the size of the Array and read it
2.Declare an integer array and read the elements of array
3.Call the different sort function to sort the element of the array
4.Display the sorted elements of array

Bubble Sort :

Title	Bubble Sort
Filename	bubble.c
Description	Write a function to sort given array using bubble sort
Input	5 4 3 2 1
Output	1 2 3 4 5
Sample Prototype	**void bubble_sort(int *ptr)** ptr Base address of the integer array return void

Insertion Sort :

Title	Insertion Sort
Filename	insertion.c
Description	Write a function to sort given array using Insertion sort
Input	5 4 3 2 1
Output	1 2 3 4 5
Sample Prototype	**void insertion_sort(int *ptr)** ptr Base address of the integer array return void

Selection Sort :

Title	Selection Sort
Filename	Selection.c
Description	Write a function to sort given array using Selection sort
Input	5 4 3 2 1
Output	1 2 3 4 5
Sample Prototype	**void selection_sort(int *ptr)** ptr Base address of the integer array return void

Sample Output :
./a.out

Enter the size of array : 5
Elements of array :
1 22 3 14 25
Select the option :
1.Bubble Sort
2.Insertion Sort
3.Selection Sort
Choice : 2
The sorted elements :
1 3 14 22 25
Do you want to continue if yes then press [y/Y] ::Y
Enter the size of array : 4
Elements of array :
11 2 13 4 5
Select the option :
1.Bubble Sort
2.Insertion Sort
3.Selection Sort
Choice : 1
The sorted elements are : 2 4 5 11 13
Do you want to continue if yes then press [y/Y] ::N

Assignment – 19

WAF to sort given array using quick sort

Objective:

To understand the working of Quick sort

Requirement:

1.Prompt the user to Enter the size of the Array
2.Declare an integer array and read the elements of array
3.Call quick sort function
4.Display the sorted elements of array

Quick Sort :

Title	Quick Sort
Filename	quick_sort.c
Description	Write a function to sort given array using quick sort
Input	5 4 3 2 1
Output	1 2 3 4 5
Sample Prototype	int quick_sort(int *ptr ,int size) ptr Base address of the integer array low Starting index high Ending index return integer value Int partition(int *ptr ,int low,int high) ptr Base address of the integer array low Starting index high Ending index return integer value

Sample Output :
./a.out

Enter the size of array : 5
Elements of array :
1 22 3 14 25
The sorted elements :
1 3 14 22 25

Assignment – 20

WAF to sort given array using Merge sort

Objective:

To understand the working of Merge sort

Requirement:

1.Prompt the user to Enter the size of the Array
2.Declare an integer array and read the elements of array
3.Call merge sort function
4.Display the sorted elements of array

Merge Sort :

<table>
<tr><td>Title</td><td>Merge Sort</td></tr>
<tr><td>Filename</td><td>merge_sort.c</td></tr>
<tr><td>Description</td><td>Write a functions to sort given array using merge sort</td></tr>
<tr><td>Input</td><td>5 4 3 2 1</td></tr>
<tr><td>Output</td><td>1 2 3 4 5</td></tr>
<tr><td>Sample Prototype</td><td>void merge_sort(int *ptr ,int size)

<table><tr><td>ptr</td><td>Base address of the integer array</td></tr><tr><td>size</td><td>Length of the array</td></tr><tr><td>return</td><td>void</td></tr></table>
void merge(int *ptr ,int size,int *L,int nL ,int *R,int nR)<table><tr><td>ptr</td><td>Base address of the integer array</td></tr><tr><td>size</td><td>Length of the array</td></tr><tr><td>L</td><td>Base address of the left sub integer array</td></tr><tr><td>nL</td><td>Length of the left sub array</td></tr><tr><td>R</td><td>Base address of the right sub integer array</td></tr><tr><td>nR</td><td>Length of the right sub array</td></tr><tr><td>return</td><td>integer value</td></tr></table></td></tr>
</table>

Sample Output :
./a.out
Enter the size of array : 5
Elements of array :
1 22 3 14 25
The sorted elements :
1 3 14 22 25

Assignment – 21

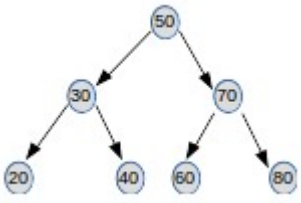
WAF to implement the function of BST

Objective :

Understand the working of Binary search Tree(BST)

Requirement :

1.Add the insert() to insert elements in the bst
2.Add the inorder() to print the elements of bst

Title	Binary Search Tree
Filename	bst_search.c , bst_maxnode.c
Description	Write a function to search data in bst Write a function to find max data node in bst
Cases	1. Tree Empty----->return BST_EMPTY 2. Tree Non-Empty
Case-1	Tree Non-Empty
Input	 Enter the data to search : 30
Output	Data found Max data : 80

Sample Output :

./a.out

Select the Option:
1.Create BST
2.Print BST
3.Search an element in BST
4.Min and Max element of BST
Choice: 2
BST is Empty
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create BST
2.Print BST
3.Search an element in BST
4.Min and Max element of BST
Choice: 3
Enter the Key element to search : 25
BST is Empty
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create BST
2.Print BST
3.Search an element in BST
4.Min and Max element of BST
Choice: 1
Enter the 5 element : 10 30 15 5 4
Do you want to continue ,if yes then type[y/Y]:Y

Select the Option:
1.Create BST
2.Print BST
3.Search an element in BST
4.Min and Max element of BST
Choice: 3
Enter the Key element to search : 15
15 is present in the BST
Do you want to continue ,if yes then type[y/Y]:N

Sample Output :
./a.out
Select the Option:
1.Create BST
2.Print BST
3.Search an element in BST
4.Min and Max element of BST
Choice: 2
BST is Empty
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create BST
2.Print BST
3.Search an element in BST
4.Min and Max element of BST
Choice: 1
Enter the 5 element : 10 30 15 5 4
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create BST
2.Print BST
3.Search an element in BST
4.Min and Max element of BST
Choice: 4
Minimum element is 4
Maximum element is 30
Do you want to continue ,if yes then type[y/Y]:N

Assignment – 22

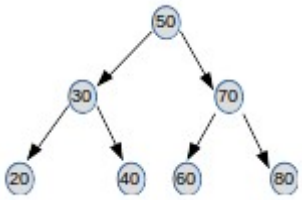
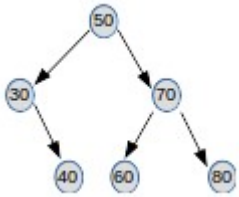
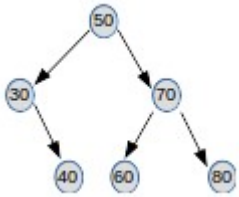
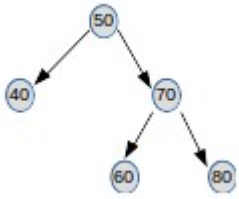
WAF to delete the given data node from the BST

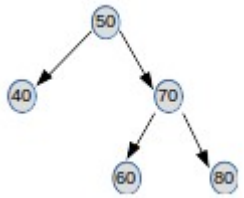
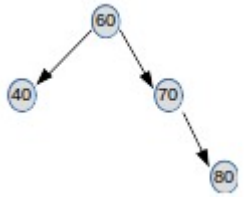
Objective :

Understand the working of Binary search Tree(BST)

Requirement :

1.Add the insert() to insert elements in the bst
2.Add the inorder() to print the elements of bst

Title	Delete nodes
Filename	bst_delete_node.c
Description	Write a function to delete the given data from the bst
Cases	Node to be deleted may be, 1. Leaf node 2. Node with single child 3. Node with two children
Case-1	Leaf node
Input	
Output	
Case-2	Node with single child
Input	
Output	After deleting 30 
Case-3	Node with two children

Input	
Output	After deleting 50 

Sample Output :
./a.out
Select the Option:
1.Create BST
2.Print BST
3.Delete an element of BST
Choice: 2
BST is Empty
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create BST
2.Print BST
3.Delete an element of BST
Choice: 3
Enter the Key element to search : 25
BST is Empty ,hence we cannot delete the element 25
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create BST
2.Print BST
3.Delete an element of BST
Choice: 1
Enter the 5 element : 10 30 15 5 4
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create BST
2.Print BST
3.Delete an element of BST
Choice: 3
Enter the element to delete : 15
15 is deleted in the BST
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create BST
2.Print BST
3.Delete an element of BST
Choice: 2
4 5 10 30
Do you want to continue ,if yes then type[y/Y]:N

Assignment – 23

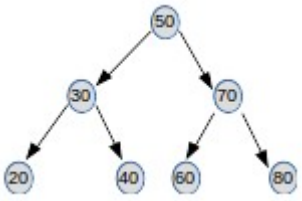
WAF to find height and total number of nodes in the BST

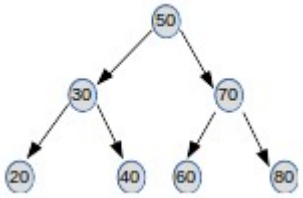
Objective :

Understand the working of Binary search Tree(BST)

Requirement :

1.Add the insert() to insert elements in the bst
2.Add the inorder() to print the elements of bst

Title	Find height
Filename	bst_find_height.c
Description	Write a function to find the height of the given tree
Cases	1. Tree Empty – return BST_EMPTY macro 2. Tree Non-Empty
Case-2	Tree Non-Empty
Input	
Output	Height = 2

Title	Number of Nodes
Filename	bst_number_nodes.c
Description	Write a function to find the total number of nodes of the given tree
Cases	1. Tree Empty---> return BST_EMPTY macro 2. Tree Non-Empty
Case-1	Tree Non-Empty
Input	
Output	number of nodes = 7

Sample Output

./a.out

```
Select the Option:
1.Create BST
2.Print BST
3.Height of BST
4.Total number of nodes in  BST
Choice: 2
BST is Empty
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create BST
2.Print BST
3.Height of BST
4.Total number of nodes in  BST
Choice: 1
Enter the 5 element : 10 30 15 5 4
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create BST
2.Print BST
3.Height of BST
4.Total number of nodes in  BST
Choice: 3
Height of BST is : 3
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create BST
2.Print BST
3.Height of BST
4.Total number of nodes in  BST
Choice: 4
Total number of nodes in BST is : 5
Do you want to continue ,if yes then type[y/Y]:N
```

Assignment – 24

WAF to sort given array using Heap sort

Objective:

To understand the working of Heap sort

Requirement:

1.Prompt the user to Enter the size of the Array
2.Declare an integer array and read the elements of array
3.Call Heap sort function
4.Display the sorted elements of array
5.To implement this function we should add dependency functions(ie. buildmaxheap and maxheapify)

<table>
<tr><td>Title</td><td>Heap Sort</td></tr>
<tr><td>Filename</td><td>heap_sort.c</td></tr>
<tr><td>Description</td><td>Write a function to sort given array using heap sort</td></tr>
<tr><td>Input</td><td>5 4 3 2 1</td></tr>
<tr><td>Output</td><td>1 2 3 4 5</td></tr>
<tr><td>Sample Prototype</td><td>void heap_sort(int *ptr,int size)
<table>
<tr><td>ptr</td><td>Base address of the integer array</td></tr>
<tr><td>size</td><td>Length of the array</td></tr>
<tr><td>return</td><td>void</td></tr>
</table>
void buildmaxheap(int *ptr,int size)
<table>
<tr><td>ptr</td><td>Base address of the integer array</td></tr>
<tr><td>size</td><td>Length of the array</td></tr>
<tr><td>return</td><td>void</td></tr>
</table>
void maxheapify(int *ptr,int i,int size)
<table>
<tr><td>ptr</td><td>Base address of the integer array</td></tr>
<tr><td>i</td><td>To heapify a subtree rooted with node i which is an index in arr[]</td></tr>
<tr><td>size</td><td>Length of the array</td></tr>
<tr><td>return</td><td>void</td></tr>
</table>
</td></tr>
</table>

Sample Output :
./a.out
Enter the size of array : 5
Elements of array :
1 22 3 14 25
The sorted elements :
1 3 14 22 25

Assignment – 25

WAF to create hash table, to search data , to insert and delete element in hash table. Also to delete entire hash table.

Objective:

To understand the working of Hashing

Requirements:

In this assignments create 5 functions

1.hash_create(hash_t *arr,int size)

As the name implies create a hash table of given size

Title	Create hash table
Filename	create_hash_table.c
Description	Write a function to create the hash table
Input	Size Ex : Size = 5
Output	Hash table with the given size Hash table 0 → -1 1 → -1 2 → -1 3 → -1 4 → -1

2.hash_insert(hash_t *arr ,int data)

Every time we call this function it should add one data in the hash table

Title	Hash table insert
Filename	hash_insert.c
Description	Write a function to insert the data in the hash table
Input	Hash Table
Output	Status [SUCCESS / FAILURE]

3.hash_search(hash_t *arr,int data)

Given the data this function should search whether data is present in the hash table or not.

Title	Hash table search
Filename	hash_search.c
Description	Write a function to search the data in the hash table
Input	Hash Table
Output	Display data is found in the hash table or not

4.hash_delete_element(hash_t *arr , int data)

Given a data this function should first check whether that data is present in the hash table or not. If it is present then delete the element and return SUCCESS else FAILURE

Title	Hash delete element
Filename	hash_delete_element.c
Description	Write a function to delete the given item from the hash table
Input	Hash Table
Output	Status [SUCCESS / FAILURE]

5.hash_delete_table(hash_t *arr)
This function should delete the entire hash table ,Allocated Memory spaces should be deleted**(**Free**)**

Title	Hash Table delete
Filename	hash_table_delete.c
Description	Write a function to delete the entire hash table
Input	Hash Table
Output	Status [SUCCESS / FAILURE]

In addition to these file you should add
6.hash_print(hash_t *arr)
This function is used to print the element of the hash table

Sample Output
./a.out

```
Enter the size of Hash table : 5
Select the Option:
1.Create Hash table
2.Insert  element in Hash table
3.Search an element in Hash table
4.Delete an element from Hash table
5.Delete Entire Hash table
6.Print the elements of Hash table
Choice: 1
Hash table created
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create Hash table
2.Insert  element in Hash table
3.Search an element in Hash table
4.Delete an element from Hash table
5.Delete Entire Hash table
6.Print the elements of Hash table
Choice: 6
Hash table
[0] → -1
[1] → -1
[2] → -1
[3] → -1
[4] → -1
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create Hash table
2.Insert  element in Hash table
3.Search an element in Hash table
4.Delete an element from Hash table
5.Delete Entire Hash table
6.Print the elements of Hash table
Choice: 2
Enter the 5 element to add in  Hash table : 10 20 33 22 55
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create Hash table
2.Insert  element in Hash table
3.Search an element in Hash table
4.Delete an element from Hash table
```

5.Delete Entire Hash table
6.Print the elements of Hash table
Choice: 6
Hash table
[0] → 10 →20 → 55
[1] → -1
[2] → 22
[3] → 33
[4] → -1
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create Hash table
2.Insert element in Hash table
3.Search an element in Hash table
4.Delete an element from Hash table
5.Delete Entire Hash table
6.Print the elements of Hash table
Choice: 4
Enter the element to delete : 10
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create Hash table
2.Insert element in Hash table
3.Search an element in Hash table
4.Delete an element from Hash table
5.Delete Entire Hash table
6.Print the elements of Hash table
Choice: 6
Hash table
[0] → 20 → 55
[1] → -1
[2] → 22
[3] → 33
[4] → -1
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create Hash table
2.Insert element in Hash table
3.Search an element in Hash table
4.Delete an element from Hash table
5.Delete Entire Hash table
6.Print the elements of Hash table
Choice: 5
Hash table Deleted
Do you want to continue ,if yes then type[y/Y]:Y
Select the Option:
1.Create Hash table
2.Insert element in Hash table
3.Search an element in Hash table
4.Delete an element from Hash table
5.Delete Entire Hash table
6.Print the elements of Hash table
Choice: 6
Hash table
[0] → -1
[1] → -1
[2] → -1
[3] → -1
[4] → -1

Do you want to continue ,if yes then type[y/Y]:N