Name - Chiradeep Banik

Enroll - 20UCS176 Sec - A

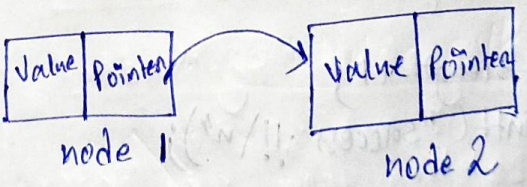
Q.1 Demonstrate the implementation of a Singly Linked list. Create 4 nodes and display all the data using the first node only.

Ans →

- **Singly Linked List**

It is a data structure that stores data as individual nodes.

where every node in the list holds the pointer to the next node in the list



- →

```
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node* next;
};
```

→

```
struct linked_list {
    struct node *head;
    struct node *tail;
    int length;
};

void push(struct linked_list *list, int data) {
    struct node* new_node = (struct node*) malloc(sizeof(struct node));
    new_node->data = data;
    new_node->next = NULL;
    if (list->length == 0) {
        list->head = new_node;
        list->tail = new_node;
    } else {
        list->tail->next = new_node;
        list->tail = new_node;
    }
    list->length++;
    printf("success !!\n");
}

void print_list(struct node *first) {
    struct node *current = first;
    for(; current != NULL; current = current->next) {
        printf("%d->", current->data);
    }
    printf("NULL\n");
}
```

```
Void main(){
    Struct linked_list list;
    list.head = NULL;
    list.tail = NULL;
    list.length = 0;
    push(&list, 1);
    push(&list, 2);
    push(&list, 3);
    push(&list, 4);
    print_list (list.head);
}
```

Console →

→ Success !!
Success !!
Success !!
Success !!
1→2→3→4→NULL

→

Q2 Consider a scenario of Singly Linked List and write down in your own words about how you plan to design a user friendly and menu driven program for the same.

→ In order to design a menu driven program we must use switch cases. So we must define all our cases →

- Pushing to the back
- Pushing to the front
- Deleting from the back.
- Deleting from the back.
- Pushing to any index.
- Deleting from any index.
- Geting the value from index.
- Printing the list.
- Closing the program.

Q.3

Write down in your own words:

a. What is Validation?

b. What is the requirement for validation in a program?

c. How do you plan to handle the pointers properly in a singly linked list program?

→ (a) Validation

↳ It is the checking of any ~~the~~ data that is inputted into a program. It is to check if the data entered into the program is correct or not.

(b) Validation Requirement

↳ It is important to validate our input in order to avoid breaking our program or avoid generating garbage results.

(c) Pointers in Singly Linked List

↳ pointers are used to connect one node to another in a linked list. One node contains the pointer to the next node.

Q.4 Create a user friendly and menu driven Singly Linked List program. to perform the following:

a. Insert at rear position.

b. Insert at front position.

c. Insert at any position.

d. Display the elements.

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int value;
    struct node *next;
};
struct linked_list {
    struct node *head;
    struct node *tail;
    int length;
};
```

```
void push (struct linked_list *list, int value){
    struct node *new_node = (struct node *)malloc(sizeof
                                (struct node));
    new_node->value = value;
    new_node->next = NULL;
    if (list->length == 0){
        list->head = new_node;
        list->tail = new_node;
    } else {
        list->tail->next = new_node;
        list->tail = new_node;
    }
    list->length++;
    printf("success!!\n");
}
void shift(struct linked_list *list, int value){
    struct node *new_node = (struct node *)malloc(
                        sizeof(struct node));
    if (list->length == 0){
        push(list, value);
    }else{
        new_node->value = value;
        new_node->next = list->head;
        list->head = new_node;
        list->length++;
        printf("success!!\n");
    }
```

```
void insert(struct linked_list *list, int value, int position){
    struct node *new_node = (struct node*) malloc(sizeof(
                                struct node));
    if (position < 0 || position > list->length) {
        printf("Invalid position !!\n");
    } else if (position == 0){
        shift(list, value);
    } else if (position == list->length){
        push(list, value);
    } else {
        new_node->value = value;
        struct node *temp = list->head;
        for (int i=0; i < position-1; i++) {
            temp = temp->next;
        }
        new_node->next = temp->next;
        temp->next = new_node;
        list->length++;
        printf("success!!\n");
    }
}
```

```
void print_list(struct linked_list *list){
    struct node *current = list->head;
    for(; current != NULL; current = current->next){
        printf("%d->", current->value);
    }
    printf("NULL\n");
}
void main(){
    struct linked_list *list = (struct linked_list *) malloc
                    (sizeof(struct linked_list));
    list->head = NULL;
    list->tail = NULL;
    list->length = 0;
    int choice, value, position;
    while(1){
        printf("1. Insert at rear position\n");
        printf("2. Insert at front position\n");
        printf("3. Insert at any position\n");
        printf("4. Display the element\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

→

```
switch (choice) {
    case 1:
        printf("Enter the value to be inserted : ");
        scanf("%d", &value);
        printf("\n");
        push(list, value);
        printf("\n");
        break;
    case 2:
        printf("Enter the value to be inserted : ");
        scanf("%d", &value);
        printf("\n");
        ~~push~~ shift(list, value);
        printf("\n");
        break;
    case 3:
        printf("Enter the value to be inserted : ");
        scanf("%d", &value);
        printf("Enter the position : ");
        scanf("%d", &position);
        printf("\n");
        insert(list, value, position);
        printf("\n");
        break;
```

```
case 4:
    printf("\n");
    printList(list);
    printf("\n");
    break;
case 5:
    printf("\n Exiting.... \n");
    free(list);
    exit(0);
default:
    printf("\nInvalid choice !! \n");
    break;
}
}
```

Console →

→ 1. Insert at rear position
2. Insert at front position
3. Insert at any position
4. Display the elements
5. Exit

→ Enter your choice : 1

→ Enter the value to be inserte : 11

→ Success !!

1. Insert at rear position
2. Insert at front position
3. Insert at any position
4. Display the elements
5. Exit

→ Enter your choice : 3
→ Enter the value to be inserted : 1
→ Enter the position : 1

→ Success !!

1. Insert at rear position
2. Insert at front position
3. Insert at any position
4. Display the elements
5. Exit

→ Enter your choice : 4

→ 1 → 1 → NULL

1. Insert at rear position
2. Insert at front position
3. Insert at any position
4. Display the elements
5. Exit

→ Enter your choice : 5

→ Exiting. ...

Q.5

What will happen if we traverse the HEAD/START of a list?

→ Traversing a list means to visit every node of the list. As only the HEAD/START node ~~contains~~ is the first node, we can only traverse the list, if we start from the HEAD node only.

Q.6

// LIST :: 10 → 20 → 30 → 40 → 50

→

Output :

↳ 10 20 30 40 50

Explanation

↳ The temp node will be equal to NULL only for the last element. So the whole list is printed before termination of the loop.