LAB-05

29/01/2024

a) Stack implementation using SLL.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* top = NULL;
void push(int data) {
    struct Node* newNode = (struct Node*) malloc( size of (struct Node));
    if (newNode == NULL){
        printf (" Oreeflow \n");
        retnen;
    }
    newnode → data = data;
    newnode → next = top;
    top = newnode;
}
int pop() {
    if (top == NULL) {
        printf (" Underflow \n");
        retnen -1 ;
    }
    struct Node* temp = top;
    int templ = temp → data;
    top = temp → next;
    free(temp);
    retnen templ;
}
void display() {
    ff (top == NULL) {
        printf (" Empty \n");
        retnen;
    }
    struct Node* current = top;
    printf ("Stack: ");
    while (current != NULL) {
        printf("%d", current → data);
        current = current → next;
    }
```

```
    printf("\n");
}
int main() {
  int choice, data;
  while (1) {
    printf("Stack operation Menu: \n");
    printf("1. Push \t 2. Pop \t 3. Display \t 4. exit \n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
      case 1: printf("Enter data to push: ");
              scanf("%d", &data);
              push(data);
              break;
      case 2: if (top == NULL) {
                printf("Underflow \n"); }
              else {
                printf("Popped element: %d \n", pop());
              }
              break;
      case 3: display();
              break;
      case 4: printf("Exiting....");
              exit(0);
      default: printf("Invalid choice! \n");
    }
  }
  return 0;
}
```

Output:

```
Stack operation Menu:
1. Push 2. Pop 3. Display 0. Exit
Enter your choice: 1
Enter data: 3
Enter your choice: 1
Enter data: 4
Enter your choice: 2
Popped element: 4.
Enter your choice: 3
Stack: 3
```

02) Queue implementation using SLL.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Queue {
    struct Node* front;
    struct Node* rear;
};
void initialise (struct Queue* queue) {
    queue -> front = NULL;
    queue -> rear = NULL;
}
int isEmpty (struct Queue* queue) {
    return (queue -> front == NULL)
}
void enqueue (struct Queue* queue, int data) {
    struct Node* newnode = (struct Node*) malloc (sizeof (struct Node));
    if (newnode == NULL) {
        print ("Overflow \n");
        return;
    }
    newnode -> data = data;
    newnode -> next = NULL;
    if (isEmpty (queue)) {
        queue -> front = newnode;
        queue -> rear = newnode;
    }
    else {
        queue -> rear -> next = newnode;
        queue -> rear = newnode;
    }
    printf ("Enqueued! \n");
}
int dequeue (struct Queue* queue) {
    if (isEmpty (queue)) {
        printf ("Underflow \n");
        return -1;
    }
```

```
struct Node* temp = queue->front;
int data = temp->data;
if (queue->front == queue->rear) {
    queue->front = NULL;
    queue->rear = NULL;
}
else {
    queue->front = queue->front->next;
}
free(temp);
printf("Dequeued %d \n", data);
return data;
}
void display(struct Queue* queue) {
if (isEmpty(queue)) {
    printf("Empty \n");
    return;
}
struct Node* current = queue->front;
printf("Queue: ");
while(current != NULL) {
    printf("%d", current->data);
    current = current->next;
}
printf("\n");
}
int main(){
struct Queue q;
initialise(&q);
int choice, data;
while(1) {
printf("Queue Operation Menu: \n");
printf("1. Enqueue \t 2. Dequeue \t 3. Display \t 4. Exit \n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch(choice){
    case 1: printf("Enter data to enqueue: ");
            scanf("%d", &data);
            enqueue(&q, data);
            break;
    case 2: dequeue(&q); break;
```

```
        case 3: display(q);
                break;
        case 4: printf("Exiting...");
                exit(0);
        default: printf("Invalid choice\n");
    }
  }
  return 0;
}
```

Output:

```
Queue Operation Menu:
1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 1
Enter data: 3
Enqueued: 3
Enter your choice: 1
Enter data: 4
Enqueued: 4
Enter your choice: 1
Enter data: 5
Enqueued: 5
Enter your choice: 2
Dequeued: 3
Enter your choice: 3
Queue: 4 5
Enter your choice: 0
Exiting...
```

08) Sorting, Reversing & concatenation of SLL

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node * next;
};
struct Node* head = NULL;
void insertbeg (int data) {
    struct Node* newnode = (struct Node*) malloc (sizeof (struct Node));
    newnode -> data = data;
    newnode -> next = head;
    head = newnode;
}
void insertend (int data) {
    struct Node* newnode = (struct Node*) malloc ( size of (struct Node));
    new node -> data = data;
    new node -> next = NULL;
    if (head == NULL) {
        head = newnode;
    }
    else {
        struct Node* temp = head;
        while (temp -> next != NULL) {
            temp = temp -> next;
        }
        temp -> next = newnode;
    }
}
void insertat middle (int data, int pos) {
    struct Node* newnode = (struct Node*) malloc (size of (struct Node));
    new node -> data = data;
    new node -> next = NULL;
    if (head == NULL) {
        head = newnode;
    }
    else {
        struct Node* temp = head;
        for (int i = 1; i < pos - 1; i++) {
            if (temp != NULL) {
                temp = temp -> next;
```

29/1/24.

End

SLL - Stacks, Queues.

SLL - Sort, Reverse, Concatenate

```
    }
    if (temp != NULL) {
      newnode -> next = temp -> next;
      temp -> next = newnode;
    }
    else {
      printf ("Invalid pos \n");
    }
  }
}
void sortlist() {
  if (head == NULL) return;
  struct Node *i, *j;
  int temp;
  for (i = head; i -> next != NULL; i = i -> next) {
    for (j = i -> next; j != NULL; j = j -> next) {
      if (i -> data > j -> data) {
        temp = i -> data;
        i -> data = j -> data;
        j -> data = temp;
      }
    }
  }
}
void reverselist() {
  struct Node* prev = NULL;
  struct Node* current = head;
  struct Node* nextnode = NULL;
  while (current != NULL) {
    nextnode = current -> next;
    current -> next = prev;
    prev = current;
    current = nextnode;
  }
  head = prev;
}
```

```
void concatenate lists (struct Node * second) {
    if (head == NULL) {
        head = second;
    }
    else {
        struct Node * temp = head;
        while (temp -> next != NULL) {
            temp = temp -> next;
        }
        temp -> next = second;
    }
}
int main() {
    int choice, data, position;
    struct Node * list 2 = NULL;
    while (1) {
        printf ("\n 1. Insert at beginning 2. Insert at End 3. Insert at
                middle 4. Sort 5. Reverse 6. Concatenate 0. Exit \n");
        printf (" Enter your choice: ");
        scanf (" %d", &choice);
        switch (choice) {
            case 1: printf ("Enter data: ");
                    scanf (" %d", &data);
                    insert beg (data);
                    break;
            case 2: printf (" Enter data: ");
                    scanf ("%d", &data);
                    insert end ( &list 2, data);
                    break;
            case 3: printf ("Enter data: ");
                    scanf (" %d", &data);
                    printf ("Enter position: ");
                    scanf (" %d", &position);
                    insert at middle (data, position);
                    break;
```

```
        case 4: sortList();
                break;
        case 5: reverseList();
                break;
        case 6: insertAtEnd(list2, 2);
                insertAtEnd(list2, 4);
                concatenate(list, list2);
                break;
        case 0: printf("Exiting...");
                exit(0);
                break;
        default: printf("Invalid choice\n");
    }
    printf("Current List: ");
    printList();
}
    return 0;
}
```

Output:

1. Insert at Beginning 2. Insert at End 3. Insert at Middle 4. Sort 5. Reverse 6. Concatenate 0. Exit.
Enter your choice: 1
Enter data: 2
Current list: 2
Enter your choice: 1
Enter data: 3
Current list: 3 2
Enter your choice: 1
Enter data: 4
Current list: 4 3 2
Enter your choice: 4
List sorted successfully.
Current list: 2 3 4
Enter your choice: 5
List reversed successfully.
Current list: 4 3 2
Enter your choice: 6
Lists concatenated successfully.
Current list: 4 3 2 2 4