29/01

(8) STACK IMPLEMENTATION USING LINKED LIST

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int data) {
 struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
 if (newNode == NULL) {
     printf("memory allocation failed \n");
     exit(EXIT_FAILURE);
 }
 newNode -> data = data;
 newNode -> next = NULL;
 return newNode;

PUSH
struct Node* push(struct Node* top, int data) {
 struct Node* newNode = createNode(data);
 newNode -> next = top;
 return newNode; }

POP
struct Node* pop(struct Node* top) {
 if (top == NULL) {
 printf("stack underflow. can't pop \n");
 return NULL;
 }
 struct Node* temp = top;
 top = top -> next;
 free(temp);
 return top; }
```

```
void displaystack (struct Node * top) {
  printf ("Stack :");
  while (top != Null){
    printf (" %d", top->data);
    top = top -> next;
  }
  printf ("\n");
}

void freestack (struct Node* top) {
  while (top != NULL) {
    struct Node* temp = top;
    free (top);
  }
}

int main(){
struct Node* top = NULL;
int choice, data;
  do {
    printf ("In Menu \n");
    printf ("1.Push");
    printf ("2 Pop");
    printf ("3. Display");
    printf ("4. Exit");
    printf (" Enter your choice: ");
    scanf ("%d", & choice);

    switch (choice) {
case 1: printf (" Enter data to push");
        scanf ("%d", & data);
        top= push (top, data);
        break;
```

```
case 2: top = pop(top);
        break;
case 3: display stack(top);
        break;
case 4: printf("Existing the program");
        break;
default: printf("Invalid choice");
  }
} while (choice != 4);
  freestack(top);
  return 0;
}
```

Op

```
Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice : 1
Enter data to push : 3
→ Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice : 2
→ Menu
1. Push
2. Pop
3. Display
4. Exit
Stack :
```

```
→ Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice : 4
Exiting the program
```

QUEUE IMPLEMENTATION USING LINKED LIST :

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

struct Node {
  int data;
  struct Node* next;
};

struct Queue {
  struct Node* front;
  struct Node* rear;
};

struct Node* createNode (int data) {
  struct Node* newNode = (struct Node*) malloc (sizeof (struct Node));
  if (newNode == NULL) {
    printf ("Memory allocation failed");
    exit (EXIT_FAILED);
  }
  newNode -> data = data;
  newNode -> next = NULL;
  return newNode;
}

struct Queue* initialize Queue () {
  struct Queue* queue = (struct Queue*) malloc (sizeof (struct Queue));
  if (queue == NULL) {
    printf ("Memory allocation failed \n");
    exit (EXIT_FAILURE);
  }
```

```
queue->front = queue->rear = NULL;
return queue;
}
```

enqueue

```
void enqueue (struct Queue* queue, int data){
    struct Node* newNode= create Node (data);
    if (queue -> rear ==NULL) {
        queue-> front= queue -> rear = new Node;
        return;
    }
    queue -> rear -> next= newNode;
    queue -> rear = newNode;
}
```

dequeue

```
void dequeue (struct Queue* queue){
    if (queue -> front == NULL) {
        printf ("Queue underflow");
        return;
    }
    struct Node* temp = queue -> front;
    queue->front = queue -> front -> next;

    if (queue -> front == NULL) {
        queue -> rear = NULL;
    }
    free (temp);
}
```

display

```
void displayQueue (struct Queue* queue){
    if (queue -> front ==NULL) {
        printf (" Queue is empty \n");
        return;
    }
```

```
struct Node* current = queue -> front;
printf ("Queue");
while (current != NULL) {
    printf ("%d", current -> data);
    current = current -> next;
}
printf ("\n");
}

memory alloc. for q
void freeQueue (struct Queue* queue) {
    while (queue -> front != NULL) {
    struct Node* temp = queue -> front;
    queue -> front = queue -> front -> next;
    free (temp);
    }
    free (queue);
}

int main () {
    Struct Queue* queue = initialize Queue ();
    int data, choice;
    do {
    printf ("\n Menu \n");
    printf ("1. Enqueue \n");
    printf ("2. Dequeue \n");
    printf ("3. Display \n");
    printf ("4. Exit \n");
    printf ("Enter your choice");
    scanf ("%d", &choice);

    switch (choice) {
    case 1: printf ("Enter data to enqueue");
```

```
scanf("%d", &data);
enqueue(queue, data);
break;

case 2: dequeue(queue);
        break;
case 3: displayQueue(queue);
        break;
case 4: printf("Exiting the program\n");
        break;
default: printf("Invalid choice!");
}
} while (choice != 4);
freeQueue(queue);
return 0;
}
```

```
Menu
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter data to enqueue: 3
→ Menu
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
```

```
→ Menu
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue:
→ Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting the program
```

Sort, Reverse and concatenate linked lists

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

Struct Node {
  int data;
  struct Node* next;
  };

void insertAtBeginning( struct Node* headRef , int data) {
  struct Node* newNode = (struct Node*) malloc (sizeof (struct No
  newNode -> data = data;
  newNode -> next = *headRef ;
  *headRef = newNode;

void printList (struct Node* head) {
  while (head != NULL) {
    printf (" %d", head->data );
    head = head -> next;
  }

  printf ("\n");

void sortList (struct Node* head) {
  struct Node* current, *NextNode ;
  int temp ;
  current = *head ;
  while ( current != NULL) {
    nextNode = current -> next
    while (nextNode != NULL) {
      if (current -> data > nextNode -> data) {
        temp = current -> data;
```

```
current->data = nextNode->data;
nextNode->data = temp;
}
nextNode = nextNode->next;
}
current = current->next;
}
}

void reverseList (struct Node* headRef) {
  structNode *prev, *current, *nextNode;
  prev=NULL;
  current=*headRef;
while (current != NULL) {
  nextNode = current->next;
  current->next= prev;
  prev= current;
  current = nextNode;
}
*headRef = prev;

void concatenateLists (struct Node* list1, structNode* list 2) {
  if (*list1 ==NULL) {
    *list1= list2;
    return;
  }
  struct Node* temp = *list1;
  while (temp->next != NULL) {
    temp= temp->next;
  }
  temp->next = list 2
}
```

```
int main () {
struct Node* list1 = NULL;
struct Node* list2 = NULL;
int choice;
int data;

while (1)
{
printf("\n 1. Insert into List 1 \n");
printf("\n 2. Insert into list 2\n");
printf("3. Sort list \n");
printf("4. Reverse list \n");
printf(" 5. Concatenate list \n");
printf(" 6. Display lists \n");
Printf(" 0. Exit \n");
print (" Enter your choice");
scanf ("%d", &choice");

switch( choice) {
case 1: prung (" Enter data to insert into list 1:");
        scanf ("%d", &data);
        Insert AtBeginning (&list1, data);
        break;
case 2: prung (" Enterdata to insert into List2");
        scanf (" %d", &data);
        InsertAtBeginning(&list2, data);
        break;
case 3: sortList (&list1);
        print ("List 1 sorted");
        break;
case 4: reverseList(&list1);
        printf (" List 1 reversed");
        break;
```

O/P:

```
case 5: concatenate lists (&list1, list 2);
        printf ("list concantenated \n");
        break;

case 6: printf ("List 1:");
        printlist (list1);
        printf ("List 2:");
        printlist (List 2);
        break;
case 7: exit (0);
        break;
default : printf ("Invalid choice");
  }
 }
 return 0;
}
```

O/P:

```
        Menu:
1. Insert into list 1
2. Instert into list 2
3. Sort list 1
4. Reverse list 2
5 Concalenate list
6. Display lists
0. Exit
Enter your choice: 1
Enter data to insert in List 1: 4
Menu.
1. Insert into list 1
2. Insert into list 2
3. Sort list 1
4. Reverse list 2
5. Concalenate lists
6. Display lists
0. Exit
Enter your choice: 1
Enter data to insert List 1: 3
Menu.
1. Insert into list 1
2. Insert into list 2
3. Sort
4. Reverse
5. Concalenate
6. Display
0. Exit
Enter your choice: 3
List 1 sorted.
```

→

5/02/24

Menu

1.
2.
3.
4.
5.
6.
0.

Enter your choice : 4 (6,7)
List 2 reversed. (7,6)

Menu.

1.
2.
3.
4.
5.
6.
0.

Enter your choice : 5
Lists concatenated

Menu

1.
2.
3.
4.
5.
6.
0.

Enter your choice : 6
list 1 : 4 3 7 6

29/01/24