Date: 29/1/2024

1 All - sort, reverse, concatenation

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
struct Node{
    int data;
    struct Node * next;
};

void insert_by(struct Node ** headRef, int data){
    struct Node * newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode -> data = data;
    newNode -> next = *headRef;
    *headRef = newNode;
}

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

void sort_l(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 rev_ll(struct Node ** headRef) {
    struct Node *prev, *current, *nextNode;
    prev = NULL;
    current = *headRef;
    while (current != NULL) {
      nextNode = current -> next;
      current -> next = prev;
      current = nextNode;
  }
  *headRef = prev;
}
void concatenate_ll(struct Node ** list1, struct Node * list2) {
    if (*list1 == NULL) {
      *list1 = list2;
      return;
    }
    struct Node * temp = *list1;
    while (temp -> next != NULL) {
      temp = temp -> next;
    }
    temp -> next = list2;
  }
```

```
int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int choice;
    int data;
    printf("\n1. insert into list 1 \n");
    printf("2. insert into list 2 \n");
    printf("3. sort ~~insert into~~ list : \n");
    printf("4. reverse list \n");
    printf("5. concatenat lists \n");
    printf("6. printf lists \n");
    printf("0. Exit \n");
    while (1)
    {
        printf("enter your choice : ");
        scanf("%d", &choice);
        switch(choice) {
            case 1: printf("enter data for list 1:? ");
                    scanf("%d", &data);
                    insert_beg(&list1, data);
                    break;
            case 2: printf("enter data for list 2: ");
                    scanf("%d", &data);
                    insert_beg(&list2, data);
                    break;
            case 3: ~~printf~~( sort_l(&list1)
                    printf("list 1 ~~reversed~~ sorted\n");
                    break;
            case 4: rev_l(&list1);
                    printf("list 1 reversed\n");
            case 5: concatinate_l(&list1, list2);
                    printf("lists concatinated \n");
                    break;
```

```
        case 6: printf("list1: ");
                print_l(list1);
                printf("list2: ");
                print_l(list2);
                break;
        case 7: exit(0)
                break;
        default: printf("Invalid choice. Please try again.\n");
    }
}
return 0;
}
```

Output:

1. Insert into list 1
2. Insert into list 2
3. Sort list 1
4. Reverse list 1
5. Concatinate list\n");
6. Print Lists
0. Exit

enter your choice: 1
enter data to insert into list 1: 2
enter your choice: 1
enter data to insert into list 1: 1
enter your choice: 2
enter data to insert into list 2: 3
enter your choice: 2
enter data to insert into list 2: 2
enter your choice: 6
list 1: 1 2
list 2: 2 3

enter your choice : 4
list 1 reversed:
enter your choice: 6
List 1: 2 1
list 2: 2 3
enter your choice: 5
lists concatinated
enter your choice: 6
list 1: 2 1 2 3
list 2: 2 3
enter your choice: 3
list 1 sorted
enter your choice: 6
list 1: 1 2 2 3
list 2: 2 3
enter your choice: 0

2.

a) Stack implementation using singly linked list.

```
Input:
#include <stdio.h>
#include <stdlib.h>
struct node {
int data;
struct node *next;
};
struct node *top=0;
void push(int n) {
struct node *newnode;
newnode = (struct node *) malloc (sizeof (struct node));
newnode -> data = n;
newnode -> next = top;
top = newnod;
}
void display() {
struct node *temp;
temp = top;
if (top == 0) {
    printf ("Underflow.");
  }
else {
   while (top != 0) {
     printf ("%d", temp -> data);
     temp = temp -> next;
  }
}
```

```
void pop() {
    struct node *temp;
    temp = top;
    if (top == 0) {
    printf("Underflow");
    }
    else {
    top = top -> next;
    free(temp);
    }
    }
void main() {
int ch; ch = 6;
printf("1: push, 2: pop, 3: display, 4: exit program\n");
while (ch != 4) {
printf("enter choice: ");
scanf("%d", &ch);
switch(ch) {
case 1: int x;
        printf("enter value: ");
        scanf("%d", &x);
        push(x);
        break;
case 2: pop();
        break;
case 3: display()
        break;
case 4: printf("exiting program.")
        break;
default: printf("invalid input.");
    }
}
```

Output:

1: push 2: pop 3: display 4 exit program

enter choice: 1

enter value: 2

enter choice: 3

2

enter choice: 2

enter choice: 3

Underflow

enter choice: 4

exiting program

b) Queue implementation using single linked list

```
#include <stdio.h>
#include <stdlib.h>
struct node {
int data;
struct node *next;
};
struct node *front = 0;
struct node *rear = 0;
void enqueue(int x) {
struct node *newnod;
newnode = (struct node *) malloc(sizeof(struct node));
newnode -> data = x;
newnode -> next = 0;
if (front == 0 && rear == 0) {
front = rear = newnode;
}
else {
rear->next = newnode;
rear = newnode;
} }
void display() {
struct node *temp;
if (front == 0 && rear == 0) {
        printf("Queue is empty");
}
  else {
        temp = front;
      while (temp != 0) {
        printf("%d", temp -> data);
        temp = temp -> next;
    }
}
```

```
void dequeue()
{
struct node *temp;
temp = front;
if (front == 0 && rear == 0)
{
printf("Underflow");
}
else
{
front = front -> next;
free(temp);
}}
void main()
{
int ch = 6;
while (ch != 0)
{
printf("1 enqueue 2: dequeue 3. display 0: exit program");
scanf("%d", &ch);
switch(ch)
while (ch != 0)
{
case printf("enter choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1: int x;
        printf("enter value: ");
        scanf("%d", &x);
        enqueue(x);
        break;
case 2: dequeue();
        break;
case 3: display();
        break;
case 0: printf("exiting program.");
        break;
default: printf("Invalid output");
         break
}}}
```

Output:

1: enqueue 2: dequeue 3: display 0: exit program

enter choice: 1

enter value: 3

enter choice: 3

~~enter~~ v 3

enter choice: 2

enter choice: 3

Queue is empty

enter choice: 0

exiting program

Ans 29.01.24