Vallisha.m , IBM19CS177 , 7-12-2020

1. Sorting Linked List

```
Void  sort (node* start){

        int flag , i;
        node * ptr1;
        node * ptr2;
        ptr2 = NULL;
        if (start == NULL)
            return;
        do
        {
            flag = 0;
            ptr1 = start;
            while (ptr1 → next != ptr2)
            {
                if (ptr → value > ptr1 → next → value)
                {
                    swap(ptr1, ptr1 → next);
                    flag = 1;
                }
            }
```

```c
            ptr1 = ptr1 ->next;
    }

    ptr2 = ptr1;
  } while ( flag );
}

void swap (node* a, node *b ) {
    int temp = a ->value;
    a -> value = b -> value;
       b -> value = temp;

  }
```

2. Reversing linked list

```c
void reverse() {
    if (head == NULL) {
        printf("Linked list is empty ");
        return;
    }
    if (head -> next == NULL) {
        printf("Reversed");
        return;
    }
    node* tmp;
    node* current = head -> next;
       node* previous = head;
    while (current != NULL) {
        tmp = current -> next;
        current -> next = previous;
        previous = current;
```

```
              current = tmp;
      }
      head -> next = NULL;
      head = previous;
 }    printf ("Reversed");
      return;
 }


3.  Merging in ascending order


    //Recursive implementation
    // called initially as merge (head 1, head 2, head 3),
    // head1 & head 2 are head pointer to two linked list
    //head3 is phead pointer of merged list
    //alternatively merge can be called as
    // merge (head1, head 2, NULL);


    void   merge (node* curr1, no node* curr2,
                     node * prev){

     int   flag 1 = (curr1 == NULL);
     int   flag 3 = (curr2 == NULL);
      if ( flag1 && flag3)
            return;
    node * newNode = (node*) malloc( sizeof(node));
    newNode -> next = NULL;
     if (prev == NULL){
         ████ Sort (head1); // algorithm in part 1
            sort (head 2);// algoritm in part 1
```

```
        head3 = newNode;
    }
    int flag2=1, flag4=1;
    if (!flag1 && !flag3) //both curr1 & curr2 not null
        flag2 = curr1->value >= curr2->value;
    }

    if (flag1)
            flag4 =0;
    if ( flag3)
            flag2 =0;

    if ( flag1 || flag2){

        newNode -> value = curr2 -> value
        curr2 = curr2 -> next;
    }
    else if( flag3 || flag4){
        newNode -> value = curr1 -> value;
        curr1 = curr1 -> next;
    }
    if (prev != NULL)
            prev -> next = newNode;
    prev = newNode;
    merge ( curr1, curr2, prev);
    }
```

```
            current = tmp;
        }
        head -> next = NULL;
        head = previous;
    }   printf ("Reversed");
        return;
}
```

3. Merging in ascending order

```
//Recursive implementation
// called initially as merge(head1, head2, head3);
// head1 & head2 are head pointer to two linked list
//head3 is head pointer of merged list
//alternatively merge can be called as
// merge (head1, head2, NULL);

void    merge (node* curr1, node* curr2,
                        node* prev){

int    flag1 = (curr1 == NULL);
int    flag3 = (curr2 == NULL);
 if ( flag1 && flag3)
            return;
node * newNode = (node*) malloc( sizeof(node));
newNode -> next = NULL;
 if (prev == NULL){
        sort (head1); // algorithm in part 1
            sort (head2); // algorithm in part 1
```

4. Implementing a stack with linked list

```c
void push (int value){

    node* ptr = (node*) malloc (sizeof(node));
    ptr → val = value;
    if(head == NULL){
        head = ptr;
        head → next = NULL;
    }
    else {
        ptr → next =
```

4. Implementing a stack with linear linked list

```c
void push (int value){

    node* ptr = (node*) malloc (sizeof(node));
    ptr → value = value;
    ptr → next = head;
    head = ptr;
}
void pop (){
    if (head == NULL){
        printf("List is empty");
        return;
    }
```

```c
        node* tmp = head->next;
        free(head);
        head = tmp;
    }
}
```

5. Implementing a Queue with a Linked List

```c
void enqueue(int input){

    node* ptr = (node*) malloc(sizeof(node));
    ptr->next = NULL;
    ptr->value = input;
    if( front == NULL && rear == NULL)
    {
        front = rear = ptr;
    }
    else {
        ptr->next = front;
        front = ptr;
    }
}
void dequeue(){

    if (front == NULL && rear == NULL){
        printf(" Queue is empty ");
        return;
    }
```

```c
    if( front → next == NULL)
    {
        free (front);
        front = rear = NULL;
        return;
    }

    node* ptr = front;
    while((ptr → next) → next != NULL)
        ptr = ptr → next;
    free (ptr → next);
    ptr → next = NULL;
}


// For all the above funutions, to display

void display (node *head){

    if (head == NULL){
        printf ("Linked List empty");
        return;
    }
    printf (" Linked list contains : ");
    node * tmp = head;
    while ( tmp != NULL){
        printf ("%d", tmp → value );
        tmp = tmp → next;
    }
}
```

//We use This structure for node;

```c
typedef struct node{
    int value;
    struct node* next;
}node;
```