Assignment-1

Data Structures and Algorithms

Name: Adithya Rao Kalathur
Reg No.: 251100610012
Branch: Computer Science and Engineering (M.E. CSE)
Email: adithya.msismpl2025@learner.manipal.edu

Date: 07-09-2025

Signature: Adithya Rao Kalathur

Question 1 -

Solve by recurrence $T(1) = 1$

- $T(n) = 4 * T(n/3) + n$
- $T(n) = 9 * T(n/3) + n^2$

Answer -

- $T(n) = 4 * T(n/3) + n$

It is a divide and conquer recurrence of the form

$$T(n) = a.T(n/b) + f(n)$$

Here:

$a = 4$

$b = 3$

$f(n) = n$

using Master theorem

if $T(n) = aT(n/b) + n^k$, compare a with b^k

- case 1 : $a < b^k \Rightarrow T(n) = \theta(n^k)$
- case 2 : $a = b^k \Rightarrow T(n) = \theta(n^k \log n)$
- case 3 : $a > b^k \Rightarrow T(n) = \theta(n^{\log_b a})$

Applying the values

$a = 4, b = 3, k = 1$

$b^k = 3^1 = 3$

comparing $a = 4 > 3 = b^k \rightarrow$ case 3

So, $T(n) = \theta(n^{\log_3 4})$

Therefore $\boxed{T(n) = \theta(n^{\log_3 4}) = \theta(n^{1.2619})}$

- $T(n) = 9 * T(n/3) + n^2$

It is of the form

$T(n) = a.T(n/b) + f(n)$

$a = 9, b = 3, f(n) = n^2$

Applying masters theorem

$b^k = 3^2 = 9$

$a = 9$ with $b^k = 9 \rightarrow$ case 2 $(a = b^k)$

Applying case 2 formula

$T(n) = \theta(n^k \log n) = \theta(n^2 \log n)$

Therefore $\boxed{T(n) = \theta(n^2 \log n)}$

Question 2 -

List any four applications of stacks and queues

Answer -

- Stack Applications -

→ Function call Management - Managing function calls and returns in programming languages.

→ Expression Evaluation - Converting infix to postfix notation and evaluating expressions

→ Undo operations - Implementing undo functionality in text editors and applications.

→ Browser History - Managing back button functionality in web browsers.

→ Back tracking algorithms - Eg DFS on graphs

→ Bracket matching / syntax checking - while parsing source code of HTML.

- Queue Applications -

→ Process Scheduling - Managing processes in operating systems (CPU Scheduling)

→ Buffer management - Handling data streams in I/O operations.

→ Breadth - First search - Graph traversal algorithms.

→ Print Queue Management - Managing print jobs in printer spooling.

→ Simulation of real-word lines - Eg: Bank teller queues.

Question 3 -

Write a function to implement union operations and Intersection on 2 instances of SLL - resultant list should contain unique elements.

Ans Answer -

sll_union_intersection.h

```
#ifndef _INCLUDED_SLL_UNION_INTERSECTION_
#define _INCLUDED_SLL_UNION_INTERSECTION_

typedef struct _list_ {
    Node *head;
} List;

List list_new();
List* list_insert_unique(List* list, int32_t data);
List list_union(List *l1, List *l2);
List list_intersection(List *l1, List *l2);
Void list_free(List *list);

#endif
```

sll_union_intersection.c

```
#include <stdlib.h>
#include <assert.h>
#include "sll_union_intersection.h"

List list_new(){
    List l = {NULL};
    return l;
}

List* list_insert_unique(List* list, int32_t data){
    assert(list != NULL);
    Node *cur = list->head;
    while(cur){
```

```
        if (cur -> data == data ) return list;
        cur = cur -> next;
    }
Node * new_node = (Node *) malloc (sizeof (Node));
new_node -> data;
New_node -> next = list -> head ;
list -> head = new_node;
return list;
}
List list_union ( List *l1 , List *l2) {
    assert (l1 != NULL && l2 != NULL);
    List result = list_ new ();
    Node *cur = l1 -> head;
    while (cur){
        list_insert_unique (& result , cur -> data);
        cur = cur -> next;
    }
    cur = l2 -> head ;
    while (cur) {
        list_insert_unique (& result, cur -> data) ;
        cur = cur -> next ;
    }
    return result;
}
List list_intersection (List *l1 , List *l2){
    assert ( l1 != NULL && l2 != NULL);
    List result = list_new ();
    for (Node *c1 = l1 -> head; c1; c1 = (c1 -> next){
        for (Node *c2 = l2 -> head; c2; c2 = (c2 -> next){
```

```
        if (c1->data == c2->data){
            list_insert_unique (result, c1->data);
            break;
        }
    }
  }
  return result;
}
void list_free (List *list){
  assert(list != NULL);
   Node * cur = list->head;
   while (cur){
       Node * tmp = cur;
       cur = cur->next;
       free (tmp);
   }
   list->head = NULL;
}
```

test_sll_union_intersection.c

```
#include <assert.h>
#include "sll_union_intersection.h"
void test_union_and_intersection(){
    List l1 = list_new();
    List l2 = list_new();

  list_insert_unique (&l1, 10);
  list_insert_unique (&l1, 20);
  list_insert_unique (&l1, 30);
  list_insert_unique (&l2, 20);
  list_insert_unique (&l2, 40);
  list_insert_unique(&l2, 50)
```

```
List uni = list_union (&l1, &l2);
List inter = list_intersection (&l1, &l2);
int union_count = 0;
for (Node * cur = uni.head ; cur; cur = cur->next) union_count++;
assert (union_count == 5);

int inter_count = 0;
for (Node * cur = inter.head; cur; cur = cur -> next) inter_count++;
assert (inter_count == 1);
 list_free (&l1)
 list_free (&l2)
  list_free (&uni)
  list_free (&inter)
}
int main (){
    test_union_and_intersection ();
    return 0;
}
```

Question 4 -

Design data structure to store student data in a single linked list. Write a function to insert and delete an element in the list. Write a function to deallocate all the memory.

student_list.h

```
#ifndef _INCLUDED_STUDENT_LIST_
#define _INCLUDED_STUDENT_LIST_
# include <stdint.h>
typedef struct _student_ {
        int32_t rollno;
```

```
    char name[30];
     struct _student_ * next;
  } Student;

typedef struct_ student _list_ {
     Student *head;
   } StudentList;

StudentList student_list_new();
StudentList *student_insert(StudentList *list, int32_t roll, const char
                                                    *name);
StudentList *student_delete(StudentList *list, int32_t roll);
void student_list_free(StudentList *list);
#endif
```

student_list.c

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "student_list.h"

StudentList student_list_new(){
    StudentList sl = {NULL};
     return sl;
}

StudentList *student_insert(StudentList *list, int32_t roll, const char *name){
    assert(list != NULL);
    Student *new_stu = (student *)malloc(sizeof(student));
     new_stu->rollno = roll;
     strncpy(new_stu->name, name, 29);
```

```
new_stu->score[29] = '\0';
new_stu->next = list->head;
list->head = new_stu;
return list;
}
StudentList *student_delete(StudentList *list, int32_t roll){
  assert(list!=NULL);
  Student *cur = list->head, *prev = NULL;
  while(cur){
    if(cur->rollno==roll){
      if(prev) prev->next = cur->next;
      else list->head = cur->next;
      free(cur);
      break;
    }
    prev = cur;
    cur = cur->next;
  }
  return list;
}
void student_list_free(StudentList *list){
    assert(list!=NULL);
    Student *cur = list->head;
    while(cur){
      Student *tmp = cur;
      cur = cur->next;
      free(tmp);
    }
    list->head = NULL;
}
```

test_student_list.c

```
#include <assert.h>
#include "student_list.h"
void test_student_list(){
    StudentList sl = student_list_new();
    student_insert(&sl, 1, "Adithya");
    student_insert(&sl, 2, "Rohan");
    student_insert(&sl, 3, "sohan");
    assert(sl.head != NULL);
    student_delete(&sl, 2);
    for(Student *cur = sl.head; cur; cur = cur->next){
        assert(cur->rollno != 2);
    }
    student_list_free(&sl);
    assert(sl.head == NULL);
}
int main(){
    test_student_list();
    return 0;
}
```

Question 5 -

Reverse a given Linked List, display Nth node from the end.

sll_reverse_nth.h

```
#ifndef _INCLUDED_SLL_REVERSE_NTH_
#define _INCLUDED_SLL_REVERSE_NTH_

typedef struct _snode_{
    int32_t data;
```

```
    struct _rnode_ *next;
} RNode;

typedef struct _rlist_ {
    RNode *head;
} RList;

RList rlist_new();
RList *rlist_insert(RList *list, int32_t data);
void rlist_reverse(RList *list);
int32_t rlist_nth_from_end(RList *list, int32_t n);
void rlist_free(RList *list);
#endif
```

sll_reverse_nth.c

```
#include <stdlib.h>
#include <assert.h>
#include "sll_reverse_nth.h"

RList rlist_new(){
    RList l = {NULL};
    return l;
}

RList *rlist_insert(RList *list, int32_t data){
    assert(list != NULL);
    RNode *node = (RNode *) malloc(sizeof(RNode));
    node->data = data;
    node->next = list->head;
    list->head = node;
    return list;
}

void rlist_reverse(RList *list){
```

```
    assert (list != NULL);
    RNode *prev = NULL, *cur = list->head, *next = NULL;
    while (cur) {
        next = cur->next;
        cur->next = prev;
        prev = cur;
        cur = next;
    }
    list->head = prev;
}
int32_t rlist_nth_from_end(RList * list, int32_t n) {
    assert (list != NULL);
    RNode * fast = list->head, * slow = list->head;
    for (int i=0; i<n; i++) {
        if (!fast) return -1;
        fast = fast->next;
    }
    while (fast) {
        fast = fast->next;
        slow = slow->next;
    }
    return slow ? slow->data : -1;
}
void rlist_free (RList *list) {
        assert (list != NULL);
        RNode *cur = list->head;
        while (cur) {
            RNode * tmp = cur;
            cur = cur->next;
            free (tmp);
        }
        list->head = NULL;
}
```

test_sll_reverse_nth.c

```
#include <assert.h>
#include "sll_reverse_nth.h"
void test_reverse_and_nth(){
    RList l = rlist_new();

    rlist_insert(&l, 10);
    rlist_insert(&l, 20);
    rlist_insert(&l, 30);

    rlist_reverse(&l);

    assert(rlist_nth_from_end(&l, 1) == 30);
    assert(rlist_nth_from_end(&l, 2) == 20);
    assert(rlist_nth_from_end(&l, 3) == 10);
    rlist_free(&l);
    assert(l.head == NULL);
}
int main(){
    test_reverse_and_nth();
    return 0;
}
```