04 Linked Lists

Problem: Large Arrays!

```
#define MAX_MEMBERS 100
```

```
typedef struct SocialNet {
    Person members[MAX_MEMBERS];
    int size;
} SocialNet;
```

Linked List: A array that grows according to needs

Linked List: Code

```
typedef struct Node {
    Person data;
    struct Node* next;
} Node;

typedef Node* LinkedList;

Node third = {
    {"Alice", 22},
    NULL
};
Node second = {
    {"Bob", 26},
    &third
};
Node first = {
    {"Charlie", 20},
    &second
};
```

```
LinkedList L = &first;
```

Size of a Liniked List

```
int size(LinkedList l) {
    int s = 0;
    while (l != NULL) {
        l = l->next;
        s ++;
    }
    return s;
}
```

A recursive solution

```
int size(LinkedList l) {
    return l==NULL? 0: size(l->next) + 1;
}
```

Printing elements of a linked list

```
void print_list(LinkedList l) {
    while (l != NULL) {
        printf("%s\t\t%d\n",l->data.name, l->data.age);
        l = l->next;
    }
}
```

Find the element at the ith position

```
Person* element_at(int pos, LinkedList l) {
    int s = 0;
    while (l != NULL) {
        if (s == pos) return &(l->data);
        l = l->next;
        s ++;
    }
    return NULL;
}
```

A recursive solution

```
Person* element_at_recursive(int pos, LinkedList l) {
    // TODO
    if (l==NULL) return NULL;
    if (pos == 0)   {return &(l->data);}
    else { return element_at(pos-1, l->next); }

    // return pos == 0 ? &(l->data): element_at(pos-1, l->next);
}
```

Append element to end of the list

```
LinkedList append(Person p, LinkedList l) {
```

```
    // Node D = {{"Raj", 18}, NULL}; Local Variable! Will not work.
    Node* D = (Node *) malloc(sizeof(Node));
    D->data = p;
    D->next = NULL;
    if (l == NULL) return D; // if l is empty just return D.
    while (l->next != NULL) {
        l = l->next;
    }
    l->next = D;
    return l;
}
```

Full code

```
#include "stdio.h"
#include "stdlib.h"
#define MAX_NAME_LEN 100

typedef struct Person {
    char name[MAX_NAME_LEN];
    int age;
} Person;

typedef struct Node {
    Person data;
    struct Node* next;
} Node;

typedef Node* LinkedList;
```

```c
void print_list(LinkedList l) {
    printf("-----------------------------------\n");
    while (l != NULL) {
        printf("%s\t\t%d\n",l->data.name, l->data.age);
        l = l->next;
    }
    printf("-----------------------------------\n");
}

int size(LinkedList l) {
    int s = 0;
    while (l != NULL) {
        l = l->next;
        s ++;
    }
    return s;
    // Simpler recursive solution
    //     return l==NULL? 0: size(l->next) + 1;
}

Person* element_at(int pos, LinkedList l) {
    int s = 0;
    while (l != NULL) {
        if (s == pos) return &(l->data);
        l = l->next;
        s ++;
    }
    return NULL;
}

Person* element_at_recursive(int pos, LinkedList l) {
    // TODO
    if (l==NULL) return NULL;
```

```
    if (pos == 0)   {return &(l->data);}
    else { return element_at(pos-1, l->next); }

    // return pos == 0 ? &(l->data): element_at(pos-1, l->next);
}

LinkedList append(Person p, LinkedList l) {

    // Node D = {{"Raj", 18}, NULL};
    Node* D = (Node *) malloc(sizeof(Node));
    D->data = p;
    D->next = NULL;
    if (l == NULL) return D;
    while (l->next != NULL) {
        l = l->next;
    }
    l->next = D;
    return l;
}

int main() {
    Node third = {
        {"Alice", 22},
        NULL
    };
    Node second = {
        {"Bob", 26},
        &third
    };
    Node first = {
        {"Charlie", 20},
        &second
    };
```

```
    Person D = {"Raj", 18};

    LinkedList l = &first;
    printf("Size of the list is %d\n", size(l));
    print_list(l);
    printf("Element at 1st position: %s\n", element_at(1,l)->name);
    printf("Element at 2nd position: %s\n", element_at(2,l)->name);
    append(D, l);
    printf("List after appending\n");
    print_list(l);
    return 0;
}
```

HW: Insert element at a position in the list

```
LinkedList insert(Person p, int pos, LinkedList l) {
    // TODO
}
```

HW: Concatenate 2 lists

```
LinkedList concat(LinkedList l1, LinkedList l2) {
    // TODO
}
```

HW: Reverse a list

```
LinkedList reverse(LinkedList l) {
    // TODO
}
```