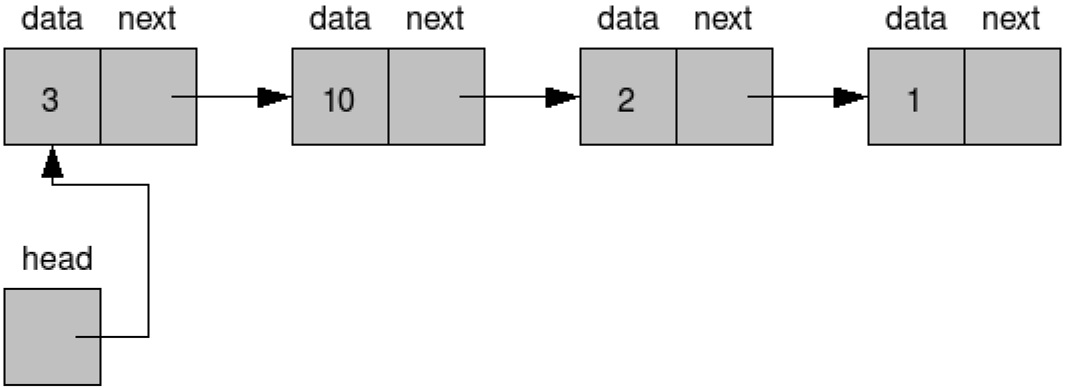
লিংকড লিস্ট হল অনেকটা অ্যারের মত। এটি এমন একটা অ্যারে যেটা যেকোনো ইন্ডেক্স থেকে প্রসারিত বা সংকুচিত হতে পারে! যেমন তোমার ইচ্ছা হল তুমি ১০০ সদস্যের একটা অ্যারের ৫০ তম অবস্থানে নতুন একটা উপাদান দিবা। এটা নরমাল অ্যারে দিয়ে করা অনেক ঝামেলার হলেও লিংকড লিস্ট দিয়ে একেবারেই সোজা। লিংকড লিস্টের আরেকটা বড় সুবিধা হল এটার সাইজ শুরুতে বলে দিতে হয় না! প্রোগ্রাম চলার সাথে সাথে (রানটাইম) এর সাইজ বড় কিংবা ছোট হতে পারে!

লিংকড লিস্ট্ কাজ করে পয়েন্টারের মাধ্যমে। তাই লিংকড লিস্ট নিয়ে কাজ করতে চাইলে অবশ্যই আগে পয়েন্টার সম্পর্কে খুব ভাল আইডিয়া থাকতে হবে। লিংকড লিস্টের সুবিধার কথা বলা হয়েছে, এর অসুবিধাও রয়েছে। প্রথমত, লিংকড লিস্টে কোনো উপাদান র‍্যানডমলি এক্সেস করা যায় না। অর্থাৎ তুমি চাইলেই ২১ তম উপাদানটা সরাসরি ara[20] দিয়ে এক্সেস করতে পারবা না, লুপ চালিয়ে যেতে হবে! দ্বিতীয়ত, লিংকড লিস্টে প্রতিটি উপাদানের জন্য উপাদানটির পাশাপাশি একটি পয়েন্টারও ধরে রাখতে হয়, তাই জায়গার অপচয় হয়।

আর কথা না বাড়িয়ে এবার আমরা কাজ শুরু করে দি।

লিংকড লিস্ট কী

সহজভাবে বললে লিংকড লিস্ট হল কতগুলো নোডের সমষ্টি। প্রতিটি নোডে আছে একটি নির্দিষ্ট উপাদান (আমরা যেটার অ্যারে চাচ্ছি) আর পরবর্তী নোডের অ্যাড্রেস। আর থাকে একটা লোকাল ভ্যারিয়েবলে প্রথম নোডটার অ্যাড্রেস, যেটাকে আমরা বলি হেড!



শেষ নোডটার অ্যাড্রেস হিসেবে দিতে হয় NULL, তাহলে বুঝা যায় যে, লিংকড লিস্টটা ওখানেই শেষ। তাহলে আমরা এখানে একটা লিংকড লিস্ট দেখতে পাচ্ছি, যাকে অ্যারে হিসেবে লিখলে হত এমন: {3, 10, 2, 1}

তো এখন দেখা যাক, একটা নোড কীভাবে ডিফাইন করবো। আমরা আগেই বলেছি একটা নোডে থাকে দুইটা জিনিস। প্রথমত, আমরা যেটার লিস্ট বানাতে চাচ্ছি, আর একটা অ্যাড্রেস। তাহলে এটা হবে একটা স্ট্রাকচার।

```
typedef struct node {
    int val;
    struct node * next;
} node_t;
```

খেয়াল করে দেখ, আমরা স্ট্রাকচারটার মধ্যেই এই স্ট্রাকচার টাইপের একটা পয়েন্টার ডিক্লেয়ার করে দিয়েছি! এখন আমরা এই নোডটা ব্যবহার করতে পারি। এবার আমাদেরকে জানতে হবে আমাদের লিংকড লিস্টটা শুরু কোথা থেকে হবে, অর্থাৎ লিংকড লিস্টের ‘হেড’-এর অ্যাড্রেস কোথায়।

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
    int val;
    struct node * next;
} node_t;
int main() {
    head = (struct node*) malloc(sizeof(node_t));
    node_t *head = NULL;
    return 0;
}
```

আমরা এখানে malloc ব্যবহার করে হেডের অ্যাড্রেস লোকেট করে দিয়েছি। পরবর্তী নোডগুলোতেও আমাদেরকে একইভাবে malloc ব্যবহার করতে হবে, নাহলে পুরো কোডটাই ক্র্যাশ করবে!
এবার আমরা আমাদের লিংকড লিস্টে ভ্যারিয়েবল রাখা শুরু করবো।

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
    int val;
    struct node * next;
} node_t;

int main() {
    head = (struct node*) malloc(sizeof(node_t));
    node_t *head = NULL;

    head->val = 7; // head er address e jei node ta ache tar val er man rakhlam 7
    head->next = NULL; // porer address hishebe rakhlam NULL. orthat ekhanei shes

    return 0;
}
```

এভাবে আমরা আমাদের লিংকড লিস্টে প্রথম উপাদানটি রাখলাম, যেটা হল 7।

এখন আমরা এই লিংকড লিস্টটা ইটারেট করবো, অর্থাৎ শুরু থেকে শেষ পর্যন্ত যাব

কাজটা খুবই সোজা। আমরা যতক্ষণ পর্যন্ত না কোনো নোডের next-এর মান NULL না হচ্ছে ততক্ষণ পর্যন্ত একটা লুপ চালিয়ে দিব! এজন্য আমরা একটি ফাংশন বানিয়ে ফেলি, যার প্যারামিটার হিসেবে আমরা পাঠাবো হেড পয়েন্টার ভ্যারিয়েবলটিকে।

```
void print_list(node_t * head) {
    node_t * current = head;

    while (current != NULL) {
        printf("%d\n", current->val);
        current = current->next;
    }
}
```

এই ফাংশনে আমরা প্রথমে current অ্যাড্রেসে থাকা নোডের val-এর মান প্রিন্ট করছি। এরপর current-এর মান বদলে করে দিচ্ছি current-এ থাকা নোডটির অ্যাড্রেসটি। আমাদের বর্তমান কোডটি আছে এই লিংকে।

এবার আমরা আমাদের লিংকড লিস্টের শেষে ভ্যারিয়েবল অ্যাড করার ফাংশন লিখবো

- এক্ষেত্রেও আমাদের ফাংশনের প্যারামিটার হিসেবে হেড পাঠাতে হবে।
- এরপর আমাদেরকে লুপ চালাতে হবে যতক্ষণ না NULL না পাই।
- NULL পেয়ে গেলে সেখানে একটা নতুন নোডের অ্যাড্রেস অ্যালোকেট করতে হবে।
- নতুন অ্যালোকেট করা নোডটিতে ভ্যালু রাখতে হবে এবং সেখানে অ্যাড্রেস হিসেবে দিতে হবে NULL!

খুবই সহজ চারটি কাজ, আমরা কোডটি লিখে ফেলি ঝটপট।

```
void push_pichone(node_t * head) {
    printf("Enter the integer: ");
    int n;
    scanf("%d", &n);
    node_t * current = head;
    // list er shes element ta khuje ber korbo
    while (current->next != NULL) {
        current = current->next;
    }

    // ebar amra variable add korbo
    current->next = malloc(sizeof(node_t)); // malloc use kora khub e important
    current->next->val = n;
    current->next->next = NULL;
}
```

আমাদের বর্তমান কোডটি আছে এই লিংকে।

এবার আমরা লিস্টে সবার শুরুতে ভ্যারিয়েবল রাখবো

- এজন্য আমাদেরকে প্রথমে একটি নতুন নোড বানিয়ে সেখানে প্রয়োজনীয় মানগুলো রাখতে হবে।

- এরপর আমরা নতুন নোডের নেক্সট অ্যাড্রেস হিসেবে দিব আমাদের পুরনো হেডটিকে।
- এরপর হেডের অ্যাড্রেস বদলে করে দিব আমাদের নতুন নোডের অ্যাড্রেসটি!
- তবে আমরা যেহেতু হেডটির ভ্যালু চেঞ্জ করে দিব, তাই এক্ষেত্রে প্যারামিটার হিসেবে হেড না পাঠিয়ে পাঠাবো হেডের অ্যাড্রেস!
  খুব সহজ কাজ, কোড লিখে ফেলি!

```
void push_shamne(node_t ** head) {
    printf("Enter the integer: ");
    int n;
    scanf("%d", &n);

    node_t * new_node;
    new_node = malloc(sizeof(node_t));

    new_node->val = n;
    new_node->next = *head;
    *head = new_node;
}
```

পুরো কোডটি আছে এই লিংকে।

লিস্টের প্রথম উপাদানটি সরিয়ে দেওয়া

- এজন্য আমাদেরকে প্রথম উপাদানে যে নেক্সট অ্যাড্রেসটা আছে, সেটাকে একটা ভ্যারিয়েবলে রাখতে হবে।
- হেডে যে নোডটা আছে, সে অ্যাড্রেসটা ফ্রী করে দিব।
- নতুন হেড হিসেবে আমাদের স্টোর করা ভ্যারিয়েবলটি সেট করবো।
- আমরা যে ভ্যারিয়েবলটি উড়িয়ে দিয়েছি, সেটা রিটার্ন করে দিব। :p
  এবার কোড লিখে ফেলি ঝটপট!

```
int pop_shamne(node_t ** head) {
    int ret = -1;
    node_t * next_node = NULL;
    // eta korchi jaate
    if (*head == NULL) {
        return -1;
    }

    next_node = (*head)->next; // head variable er next address ta rakhtichi save kore
    ret = (*head)->val; // value ta save kore rakhtichi jaate pore return korte pari
    free(*head); // memory free kore dicchi
```

```
    *head = next_node; // head hishebe shei purono store kora address ta set kortichi

    return ret;
}
```

পুরো কোডটি আছে এখানে।
বাড়ির কাজ!

- লিস্টের শেষ উপাদানটি সরিয়ে দেওয়ার ফাংশন লিখতে হবে।
- একটি ইন্টিজার ইনপুট নিয়ে সেটা যত বার আছে, তত বার রিমুভ করতে হবে এবং সবশেষে কতটি ইন্টিজার রিমুভ করা হয়েছে, সেটা রিটার্ন করতে হবে।
- লিস্টের n তম উপাদান রিমুভ করার ফাংশন লিখতে হবে।

মূলত স্ট্যাক, কিউ, গ্রাফ ইত্যাদি ডাটা স্ট্রাকচার ইমপ্লিমেন্ট করার জন্য লিংকড লিস্টের দরকার পড়ে।