আমরা ইতোমধ্যেই লিংকড লিস্ট সম্পর্কে ধারণা লাভ করেছি। এই পর্বে আমরা দেখবো লিংকড লিস্টেরই একটি বর্ধিত রূপ – যার নাম ডাবলি লিংকড লিস্ট। পার্থক্য হল লিংকড লিস্টে শুধু শুরু থেকে শেষ পর্যন্ত ইটারেট করা যেত। ডাবলি লিংকড লিস্টে NEXT পয়েন্টারের সাথে সাথে একটি PREV পয়েন্টারও রাখা হয়, যার মাধ্যমে আমরা পেছন থেকেও এগোতে পারি।

লিংকড লিস্টের প্রতিটি উপাদানকে বলা যেতে পারে একেকটি লিংক। আর এসব লিংকের সমষ্টিই হল লিংকড লিস্ট। ডাবলি লিংকড লিস্টে যা যা থাকা দরকারঃ

- NEXT পয়েন্টার – যার মাধ্যমে একটি লিংক তার পরবর্তী লিংকের সাথে সংযুক্ত (Linked) থাকবে।
- PREV পয়েন্টার – যার মাধ্যমে একটি লিংক তার পূর্ববর্তী লিংকের সাথে সংযুক্ত (Linked) থাকবে।
- First এবং Last – একটি ডাবলি লিংকড লিস্টের এ দুইটি ভ্যারিয়েবল থাকতে হবে, যারা যথাক্রমে লিস্টটির শুরু এবং শেষ নির্দেশ করে।
- শেষ লিংকটির NEXT পয়েন্টার হবে NULL, কারণ এখানেই লিস্টটির শেষ। একইভাবে প্রথম লিংকটির PREV পয়েন্টার হবে NULL।

একটি ডাবলি লিংকড লিস্টের মূলত নিচের অপারেশনগুলো থাকা দরকার। এর মধ্যে কয়েকটি আমি এখানে দেখাবো, বাকিগুলো তোমার নিজের করে নিতে হবে।

- **Insertion** – লিস্টের শুরুতে নতুন একটি উপাদান (element) যুক্ত করা
- **Deletion** – লিস্টের প্রথম উপাদানটি মুছে দেওয়া
- **Insert Last** – লিস্টের শেষে একটি উপাদান যুক্ত করা
- **Delete Last** – লিস্টের শেষ উপাদানটি মুছে দেওয়া
- **Insert After** – লিস্টের যেকোনো একটি উপাদানের পর নতুন একটি উপাদান যুক্ত করা
- **Delete** – লিস্ট থেকে কোনো একটা পার্টিকুলার ভ্যালুযুক্ত উপাদান মুছে দেওয়া
- **Display forward** – লিস্টটি শুরু থেকে শেষ পর্যন্ত প্রিন্ট করা
- **Display backward** – লিস্টটি শেষ থেকে শুরু পর্যন্ত প্রিন্ট করা

তো কথা না বাড়িয়ে সরাসরি কোডিং-ই শুরু করা যাক। 😀 প্রথমেই আমাদের একটি লিংক স্ট্রাকচার তৈরি করতে হবে, যার থাকবে PREV এবং NEXT পয়েন্টার এবং একটি VALUE।

```
typedef struct node {
   int val;
   struct node *NEXT;
   struct node *PREV;
} node_t;
```

এখানে PREV পয়েন্টারে থাকবে আগের লিংকটির অ্যাড্রেস, এবং NEXT পয়েন্টারে থাকবে পরবর্তী লিংকটির অ্যাড্রেস। এখন আমরা একটি লিংকড লিস্ট তৈরি করবো। এজন্য আমাদের first এবং last নামের দুইটি পয়েন্টার লাগবে। শুরুতে দুইটিই NULL হবে। কারণ, কোনো element-ই তো নেই, first, last আসবে কোথা থেকে। 😛

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

node_t *first, *last, *temp;

void create () {
    first = (node_t*) malloc (sizeof(node_t));
    last = (node_t*) malloc (sizeof (node_t));
    first = last = NULL;
}

int main() {
    create();
    return 0;
}
```

বুঝার ক্ষেত্রে অনেক সাহায্য করে একটা জিনিষ। সেটা হল লিস্ট প্রিন্ট করার ফাংশন আগেই লিখে রাখা। আমরা যেহেতু আগের পর্বেই NEXT ব্যবহার করে লিংকড লিস্ট প্রিন্ট করা শিখেছি, তাই এখন আর আলাদা করে সেটা লিখছি না। আশা করি নিজেই লিখে নিতে পারবে। না পারলে এই পর্বের শেষে পুরো কোডের লিংক দেওয়া আছে। সেখান থেকে দেখে নিও। ☺

লিস্টে কোনো নতুন উপাদান যোগ করা

আমাদের লিস্টে আমরা নতুন উপাদান দুইটি ভিন্ন জায়গায় যোগ করতে পারি, শুরুতে কিংবা শেষে। দুই ক্ষেত্রেই আমাদের প্রথমে একটি লিংক তৈরি করে নিতে হবে যে ভ্যালু দেওয়া হয়েছে তা দিয়ে। আমি শুধু শুরুতে অ্যাড করার ব্যাপারটা কোড করে দেখাবো।

- এজন্য আমাদের নতুন তৈরি লিংকের NEXT হিসেবে দিতে হবে আগে সেভ করে রাখা first।
- নতুন লিংকের PREV হিসেবে দিতে হবে NULL। কারণ এর আগে আর কোনো উপাদান নেই।
- আগে যেটা first ছিল, তার PREV-এ দিতে হবে নতুন লিংকের অ্যাড্রেস। (যদি লিংকড লিস্টটা empty না হয়। সেটা first-এর মান NULL কিনা চেক করলেই বোঝা যাবে! আর যদি লিংকড লিস্ট empty হয় সে ক্ষেত্রে নতুন লিংকটি হবে last-এরও অ্যাড্রেস)
- সবশেষে first-এর নতুন অ্যাড্রেস হবে আমাদের নতুন লিংকটি। খেয়াল কর, লিস্ট যদি empty থাকে এই অপারেশনের আগে, তাহলে অপারেশন শেষে first এবং last দু'টির মান একই হয়। কারণ নতুন ক্ষেত্রে উপাদান তো একটাই। তাই first, last-ও সেম! এইটা খেয়াল করা খুব গুরুত্বপূর্ণ। কারণ অনেক ক্ষেত্রেই এই সহজ জিনিসটাতে ভুল হয়ে যায়!

```
void insertFirst (int num) {
   temp = (node_t*) malloc (sizeof (node_t));
   temp->val = num;
   temp->NEXT = first;
   temp->PREV = NULL;
   // empty na hole aager first element tar notun PREV hobe bortoman temp
   if (first!=NULL) first->PREV = temp;
   else last = temp; // empty hole last link ta hobe ei temp tai
   first = temp;
}
```

সহজ না? 😀 এবার তোমার কাজ হবে শেষে insert করার কাজটি করে ফেলা। চেষ্টা করেও না পারলে এখানে কোড দেখতে পারো। ☺

কোনো উপাদান মুছে দেওয়া

নতুন উপাদান যোগ করার মত এ কাজটিও মূলত দুই জায়গায় করা যায়। শুরুতে এবং শেষে। আমি শেষ উপাদান ডিলিট করে দেখাবো। তোমার কাজ হবে প্রথম উপাদান ডিলিট করা।

এ কাজটি অনেক সহজ।

- last-কে ফ্রী করে দিতে হবে। এটা করা গুরুত্বপূর্ণ। কারণ না হলে এই মেমরিটা অন্য কেউ এক্সেস করতে পারে না। এজন্য last-এর PREV অ্যাড্রেসটা temp-এ সেভ করে রাখা দরকার আগে।
- last-এর আগের লিংকটির (যেটার অ্যাড্রেস temp-এ সেভ করা আছে) NEXT-কে NULL করে দিতে হবে।
- নতুন last হিসেবে দিতে হবে temp-এ সেভ থাকা অ্যাড্রেসটা!

তবে একটা সমস্যা আছে। আমরা তো একটা empty লিংকড লিস্ট থেকে কিছু মুছতে পারবো না। তাই আগেই চেক করে নিতে হবে লিস্টটা empty কিনা। বারবার একই জিনিষ চেক করা লাগতেছে। একটা ফাংশনই লিখে রাখা যাক এর জন্য!

আরও একটা সমস্যা আছে। যদি লিস্টে শুধু একটি উপাদানই থাকে, সেক্ষেত্রে last-এর ভ্যালু NULL করে দিতে হবে। এটা চেক করা যায় head->next NULL নাকি সেটার মাধ্যমে।

```
int popLast () {
   int toBeReturned=0;
   if (!isEmpty()) {
      toBeReturned = last->val;
      temp = (node_t*) malloc (sizeof (node_t));
      temp = last->PREV;

      if (first->NEXT == NULL) {
         first = last = NULL;
      }

      else {
         free(last);
         temp->NEXT = NULL;
         last = temp;
      }
```

```
    }
    return toBeReturned;
}
```

খেয়াল কর, লিস্টটা যদি আগে থেকে empty থাকে, তাহলে ফাংশনটা রিটার্ন করবে 0। এবার popFirst-এর কোড লিখে ফেল তো। 😀 কোডটা এখানে আছে।

মৌলিক আলোচনা মূলত এখানেই শেষ। আমরা যা শিখেছি, তা দিয়ে Stack, Queue ইমপ্লিমেন্ট করে ফেলতে পারবো! তোমার জন্য কিছু হোমওয়ার্ক থাকবেঃ

- লিস্টটা রিভার্স অর্ডারে প্রিন্ট করা। অর্থাৎ, শেষ থেকে শুরু পর্যন্ত।
- প্রথম উপাদান শো করা।
- শেষ উপাদান শো করা।
- কোনো একটা value ইনপুট নিয়ে, লিস্টে সে value-টা যতবার পাওয়া যায়, ততবার ডিলিট করা।
- লিস্টের n-তম উপাদানের পর নতুন একটি উপাদান যোগ করা।

আমার পুরো কোডটি পাবে এই লিংকে। ☺