**Linked List**

লিঙ্কড লিস্ট (linked list) হলো একটি ডাটা স্ট্রাকচার যেখানে ডাটা গুলোকে একটার পরে আরেকটা, লিঙ্ক আকারে রাখা হয়। ডাটা রাখার জন্য নোড তৈরি করা হয় একাধিক ফিল্ডের সমন্বয়ে। একটা নোড থেকে পরবর্তী নোডে যাওয়ার জন্য লিঙ্ক থাকে। আমরা যদি লিঙ্কড লিস্ট (linked list) তৈরি করি তবে এর চিত্রটা অনেকটা নিচের মতো দেখা যাবে।



এখানে ১,২ ও ৩ প্রত্যেকে একেকটি নোড যেখানে ২টি ফিল্ড আছে। একটি হলো ডাটা (data) যেখানে আমরা নোড ভ্যালু রাখবো এবং দ্বিতীয়টি হলো next যা মূলত একটি পয়েন্টার, এখানে আমরা পরবর্তী নোডের এড্রেস জমা করে রাখবো।

**Data structure: linked list in Bangla**

প্রথমের আমরা আসি, লিঙ্কড লিস্ট (linked list) এর প্রতিটি নোড আমরা কিভাবে গঠন করবো? আমরা যারা C/ C++ জানি তারা স্ট্রাকচারের (struct) নাম শুনেছি নিশ্চয়। স্ট্রাকচার হলো C/ C++ এর একটি ব্যাবস্থা যেখানে আমরা একাধিক ডাটাটাইপের ভারিয়েবল একসাথে করে নতুন একটি ডাটাটাইপ গঠন করতে পারি।

নিচে দেখি নেই নোডের গঠনটি কেমন হবে। আমরা একটি নোড তৈরি করবো যেখানে ডাটা রাখার জন্য একটি ফিল্ড থাকবে এবং এই নোডের পরবর্তী নোডের এড্রেস রাখার জন্য একটি পয়েন্টার থাকবে। নিচের কোডটি খেয়াল করি আমরা।

|  |  |
| --- | --- |
| 1  2  3  4 | struct node{  int data;  node \*next=NULL;  }; |

(Curly brace এর শেষে সেমিকোলন আছে) এখানে node একটি User defined datatype যার মধ্যে আমরা int টাইপ একটি ফিল্ড (data) এবং পরবর্তী নোডের এড্রেস রাখার জন্য একটি node\* পয়েন্টার নিয়েছি।

আমরা একেকটি নোড কিভাবে তৈরি করবো? এটা খুবই সহজ। যারা C++ ব্যবহার করি তারা খুব সহজেই new কিওয়ার্ড ব্যবহার করে কাজটি করে নিতে পারবো নিচের মতো করে।

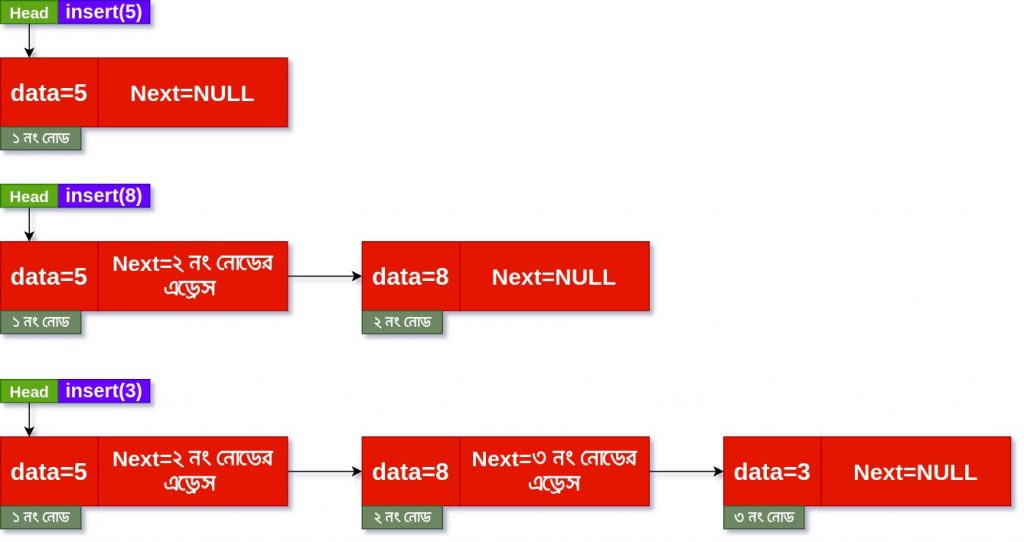
|  |  |
| --- | --- |
| 1 | node \*new\_node=new node; |

এখানে পয়েন্টার ব্যবহার করার কারণ হলো আমরা যখন বিভিন্ন অপারেশন করবো (insert,delete) তখন আমাদের বিভিন্ন সময় কোন একটি নোডের ফিল্ড আপডেট করতে হবে। যার জন্য ওই নোডের এড্রেসে গিয়ে কাজ করতে হয়। এই জন্য আমাদের পয়েন্টার দিয়ে ওই এড্রেসটিকে পয়েন্ট করে রাখতে হয়।

দ্বিতীয়ত, আমরা যদি একটি ফাংশনের মধ্যে সাধারণ ভেরয়েবল তৈরি করি, তবে ফাংশনটির স্কোপ থেকে বেরনোর সাথে সাথেই ভেরিয়েবলটি ডিলিট হয়ে যাবে কারণ এভাবে ভেরয়েবল তৈরি করলে স্ট্যাক মেমরিতে তৈরি হয়। কিন্তু আমরা যখন new কিওয়ার্ড ব্যবহার করবো তখন এর মেমোরি হীপ (heap) এ তৈরি হবে। যার ফলে ফাংশনের স্কোপ শেষের সাথে সাথে ভেরয়েবলটি ডিলিট হবে না। আরও জানতে এই পোস্টটি পরুন [মে‌মো‌রি ম্যানেজ‌মেন্ট: স্ট্যাক এবং হিপ মে‌মো‌রি।](https://iishanto.com/%e0%a6%b8%e0%a7%8d%e0%a6%9f%e0%a7%8d%e0%a6%af%e0%a6%be%e0%a6%95-%e0%a6%ae%e0%a7%87%e0%a6%ae%e0%a7%87%e0%a6%be%e0%a6%b0%e0%a6%bf-%e0%a6%b9%e0%a6%bf%e0%a6%aa-%e0%a6%ae%e0%a7%87%e0%a6%ae%e0%a7%87/)

## লিঙ্কড লিস্ট এ ডাটা insert করা – Insertion in linked list

লিঙ্কড লিস্ট এ ডেটা রাখতে হলে অবশ্যই একটি নতুন নোড তৈরি করতে হবে। নিচের চিত্রে প্রক্রিয়াটি বর্ণনা করার চেষ্টা করেছি।



প্রথমেই আমদেরকে লিস্টের শুরতে যে নোড থাকবে তার এড্রেসকে কোন একটি ভেরিয়েবলে জমা করে রাখতে হবে। আমরা এর নাম দিলাম head। উপরের চিত্রে খেয়াল করি। প্রথম ধাপে আমরা ৫ ইনসার্ট করেছি। এর জন্য আমদেরকে একটি নতুন নোড তৈরি করতে হবে।

|  |  |
| --- | --- |
| 1  2 | node \*new\_node=new node;  new\_node->data=5; |

যেহেতু এই নোডটি ই আমদের শুরুর নোড, তাই Head নামক পয়েন্টারে এই নোডটির এড্রেস জমা করে রাখবো আমরা। লক্ষকরই, এই নোডের next পয়েন্টার এর মান NULL।

|  |  |
| --- | --- |
| 1  2  3  4 | if(head==NULL){      head=new node;      return;  } |

দ্বিতীয় ধাপে আমরা যখন ৮ ইনসার্ট করেছি তখন একই ভাবে আমরা একটি নতুন নোড তৈরি করেছি। এই নোডটিকে প্রথম নোডের শেষে রাখতে হবে। কিভাবে রাখবো? উপায় হলো প্রথম নোডের next পয়ন্টার দিয়ে এই নোডটিকে পয়েন্ট করে দেয়া। এর জন্য আমরা শুরুতে ১ নং নোড এক্সেস করবো এবং এর next নামক পয়েন্টারে এই নোডের এড্রেস বসাবো। এখানেও লক্ষ করি, নোড ২ এর next পয়েন্টার NULL।

৩ নং ধাপ ও আমরা একই ভাবে সম্পন্ন করেছি। নোড ১ থেকে next এর মাধ্যমে আমরা নোড ২ এ গিয়েছি। নোড ২ এর next পয়েন্টার যেহেতু NULL, তাই আমরা বুঝতে পারি এটিই শেষ নোড, তাই এই next পয়েন্টার এর মাধ্যমে নোড ৩ কে পয়েন্ট করে দিয়েছি।

এককথায় বলতে গেলে, শুরুর নোড (head node) বাদে বাকি নোডগুলো insert করতে আমরা while লুপের মাধ্যমে শেষ নোড খুঁজে বের করবো। তারপর এর next পয়েন্টারে আমরা নতুন নোডকে পয়েন্ট করবো।

## লিঙ্কড লিস্ট এ insert() ফাংশন।

লিঙ্কড লিস্টের সবার শেষে ইনসার্ট করা খুবই সোজাসাপ্টা। আমরা প্রথমে কোড দেখবো। তারপর কোডের ব্যাখ্যা করবো।

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | struct node{  int data;  node \*next=NULL;  };    node \*head=NULL;  void insert(int n){  node \*new\_node=new node;  new\_node->data=n;    if(head==NULL){  head=new\_node;  return;  }    node \*current;  current=head;  while(current->next!=NULL){  current=current->next;  }  current->next=new\_node;    } |

কোডের প্রথমে ১ থেকে ৪ নং লাইনে আমরা node এর স্ট্রাকচার ডিফাইন করেছি। দেখতে পারছি এখানে দুইটি ফিল্ড আছে, একটি ডাটা রাখার জন্য int টাইপ এবং আরেকটি পয়েন্টার পরবর্তী নোডের এড্রেস রাখার জন্য। এর পর আসি ৬ নং লাইনে। আগেই বলেছিলাম, লিঙ্কড লিস্টের শুরু বুঝাতে একটি পয়েন্টার লাগবে, যেখানে আমরা শুরুর নোডটি রাখবো। এই head পয়েন্টারটিই হলো আমদের সেই ভেরিয়েবল।

এর পরে আসে insert() ফাংশন। এই ফাংশন একটি প্যারামিটার নিবে। এই প্যারামিটারের মাধ্যমে আমরা ডাটা পাস করবো।

এখন পরের if স্টেটমেন্ট এ দেখি, যদি head পয়েন্টার নাল বা খালি হয়, তাহলে আমরা কি বুঝতে পারি? বুঝতে পারি আমাদের লিঙ্কড লিস্টে কোন নোড নাই। তাই এই ক্ষেত্রে আমরা পরীক্ষা করেছি head এর মান NULL নাকি। NULL হলে new\_node কে head পয়েন্টার দ্বারা পয়েন্ট করেছি এবং ফাংশন থেকে বেরিয়ে গিয়েছি।

এখন আসা যাক ১৬-১৭ নং লাইনে। এখানে লুপের ভিতর বিবেচ্য নোডের ট্র্যাক রাখার জন্য একটি ভেরিয়েবল নিয়েছি এবং শুরুর মান হিসেবে head নোড ( অন্য কথা নোড ১ ) কে পয়েন্ট করে দিয়েছি।

এবার লুপে আমরা খুঁজেছি লিস্টের শেষ নোড কোনটা। কারণ আমরা লিস্টের শেষ নোডের পরে নতুন নোড রাখবো। আমরা যখন current = শেষ নোডে এসে পরবো তখন লুপ থেমে যাবে। এভাবে current পয়েন্টার আমাদের লিস্টের শেষ নোডকে পয়েন্ট করবে। অতঃপর আমরা current এর next ফিল্ডে নতুন নোড হিসেবে new\_node কে যুক্ত করবো।

শেষ নোড বুঝার উপায়: শেষ নোডের next পয়েন্টার সবসময় NULL থাকবে। যতক্ষণ পর্যন্ত current->next==NULL না হয় ততক্ষণ আমরা লুপ চালিয়ে পরের নোড গুলোতে যেতে থাকবও।

## লিঙ্কড লিস্ট এ সার্চ করা – Searching in linked list

ধরা যাক আমরা একটি লিস্ট তৈরি করেছি। এখন আমাদেরকে এই লিস্ট থেকে একটি সংখ্যা n খুঁজে বের করতে হবে। আমরা কিভাবে সার্চ করার কাজটি করবো? লিঙ্কড লিস্টে সার্চ করা খুবই সহজ। আমরা কোডটি দেখি আগে,

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | bool search(int n){  node \*current = head; // head পয়েন্টারকে আমরা current নোড বানাবো প্রথমে।  while(current!=NULL){ // লিস্টের শেষ নোড পর্যন্ত সর্বচ্চো আমরা লুপ চালাবো।  if(current->data==n) return true; // যদি current নোড এর data ফিল্ড n এর সমান হয় তবে true রিটার্ন করবো।  current=current->next; // আগের শর্ত পূরণ না হলে current নোড এর next নোড কে current নোড করবো  }  return false; // উপরের লুপে true রিটার্ন না হলে এখানে এসে false রিটার্ন হবে।  } |

উপরের কোডটি খুব সোজা একটি কোড। প্রথমেই ২ নং লাইনে আমরা current=headকরে দিয়েছি। কারণ আমরা head নোড থেকে খোজা শুরু করবো।

তারপর আমরা লুপে ঢুকেছি, যেখানে লুপের শর্ত হলো current!=NULL, অর্থাৎ শেষ নোডে চলে আসলে আমাদের current পয়েন্টার এর মান NULL হবে। লুপের ভেতরে আমরা current পয়েন্টারের data ফিল্ডের মান পরীক্ষা করে দেখছি এর মান n এর সমান হয় কিনা। যদি সমান হয় তবে true রিটার্ন করেছি, অন্যথায় current কে বর্তমান নোডের next নোডে পয়েন্ট করে দিয়েছি।

সব শেষ যদি লুপের মধ্যে true রিটার্ন না হয়, তবে আমরা লুপের বাইরে false রিটার্ন করেছি।

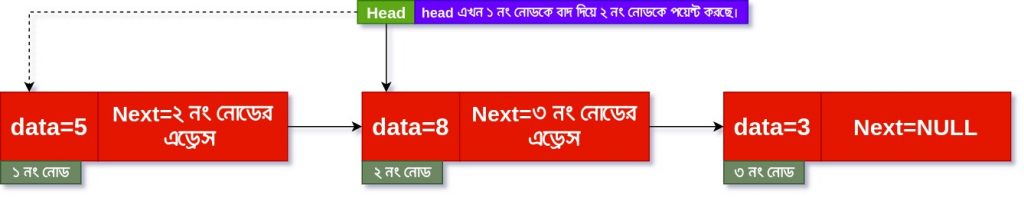
## লিঙ্কড লিস্ট এ একটি নোড ডিলিট করা – Deletion of a node in Linked List

লিঙ্কড লিস্টে ডিলিট অপারেশন করার জন্য আমাদেরকে ৩ টি কেস মাথায় রাখতে হবে। যথাক্রমে, (১) শুরুর একটি নোড ডিলিট করার, (২) শেষের একটি নোড ডিলিট করা এবং (৩) মাঝখানের একটি নোড ডিলিট করা।

কিন্তু এই ৩ টি কেসের যাই হোকনা কেন, আমদেরকে প্রথমে যেই নোড ডিলিট করবো তাকে খুঁজে বের করতে হবে। এর জন্য আমরা সার্চ অপারেশন করবো।

### শুরু থেকে ডিলিট করা বা head যাকে পয়েন্ট করেছে ওই নোড ডিলিট করা।

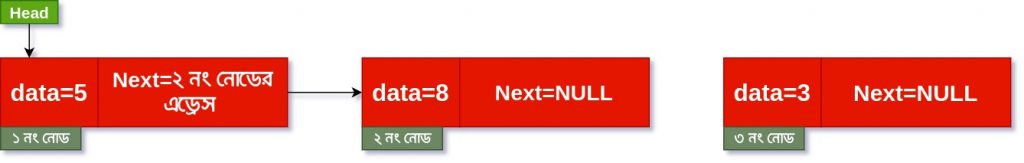
head নোড ডিলিট করতে হলে আমদেরকে শুধু head কে আপডেট করে head=head->next করে দিতে হবে। এতে করে head এর জায়গাকে পরবর্তী নোডের এড্রেস দ্বারা আপডেট করা হবে এবং পুর্বের head এর কোন ট্রেস থাকবে না আমাদের।



লিঙ্কড লিস্টে আমাদের সব অপারেশন head থেকে করা লাগে। তাই উপরের চিত্র থেকে বুঝা যাছে আমাদের অপারেশন এখন থেকে ২ নং নোড থেকে শুরু হবে।

### শেষ থেকে কোন নোড ডিলিট করা

শেষ থেকে ডিলিট করতে হলে আমাদের কে শুধু শেষ নোডের আগের নোডের next ফিল্ডে NULL সেট করে দিতে হবে। এতে করে আমাদের লিস্টের শেষ নোডের এড্রেস আর সেভ থাকবে না।



এখানে দেখা যাচ্ছে ২ নং নোডের next ফিল্ড NULL করা হয়েছে। এবং ৩ নং নোডের সাথে ২ নং নোডের সকল সংযোগ বিচ্ছিন্ন। এতে করে ৩ নং নোড আমরা আর যেতে পারব না এবং ২ নং নোড আমাদের শেষ নোড হিসেবে বিবেচ্য হবে।

### মাঝখান থেকে একটি নোড ডিলিট করা

মাঝখান থেকে একটি নোড ডিলিট করতে হলে আমরা ওই নোডের পরের নোডের সাথে ওই নোডের আগের নোডের একটি কানেকশন করে দিবো আগের নোডের next ফিল্ডের মাধ্যমে।



উপরের চিত্রে ২ নং নোড ডিলেট করা হয়েছে। তাই ১ নং নোডের সাথে ৩ নং নোডের সাথে সংযোগ স্থাপন করা হয়েছে। এতে করে ২ নং নোড কে লিস্ট থেকে বাদ দেয়া গিয়েছে।

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | void delete\_node(int n){  node \*current,\*previous;  current=head;  previous=NULL;  while(current!=NULL){  if(current->data==n){ // আমরা যখন সঠিক নোডটি খুঁজে পাবো  if(previous==NULL){ // যখন previous পয়েন্টার NULL হবে, তার মানে ট্রাগেট নোড একটি head নোড  head=current->next; // পরের নোডটিকে head  নোড করে দিলাম  delete[] current; // বর্তমান নোডের মেমরি ফ্রি করে দেয়া হলো  return;  }else if(current->next==NULL){ // যদি আগের শর্ত পূরণ না করে এই শর্ত পূরণ করে তবে এটা সবার শেষের অদ্য  previous->next=NULL; // শেষের নোডের আগের নোডকে NULL করে দিলাম।  delete[] current; // বর্তমান নোডের মেমরি ফ্রি করে দিলাম।  return;  }else{ // প্রথম না শেষ ও না, তার মানে মাঝের নোড  previous->next=current->next; //আগের নোডের সাথে পরের নোডের সম্পর্ক করে দিয়েছি।  delete[] current;  // মেমোরি ফ্রি করে দিলাম/  return;  }  }  previous=current;  current=current->next;  }  } |

আশা করি উপরের তিনটি বেসিক অপারেশন বুঝা গিয়েছে।

উপরের যে insert অপারেশন করেছি, তাকে বলা হয় append. আমরা এছাড়াও, লিস্টের সবার সামনে insert করতে পারব। লিস্টের মাঝেও insert করতে পারব। লিস্টে সবার সামনে insert করতে হলে আপনাদের যা করতে হবে তার কোড নিচের মতো,

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | void insert\_head(int n){  node \*new\_node=new node;  new\_node->data=n;  new\_node->next=head;  head=new\_node;  } |

এখানে আমরা স্বাভাবিক ভাবেই নতুন নোড তৈরি করেছি। তারপর ৪ নং লাইনে নতুন নোডের next ফিল্ডে আগের head নোডকে পাঠিয়ে দিয়েছি এবং নতুন head নোড হিসেবে নতুন নোডকে বসিয়ে দিয়েছি।

এখন আমরা যদি কোন একটি নোডের পরে একটি নতুন নোড insert করতে চাই তবে আমাদেরকে প্রথমে ওই নোডে যেতে হবে। ধরা যাক এই নোডটি x এই নোডের পরের নোডটি y. এখন আমরা x এবং y এর মাঝে আরেকটি নোড z বসাতে চাই। এটা করার জন্য আমাদেরকে সার্চের মাধ্যমে প্রথমে x নোডে যেতে হবে। তারপর x নোডের next দ্বারা z কে পয়েন্ট করাতে হবে। তারপর z এর next ফিল্ড দ্বারা y কে পয়েন্ট করাতে হবে। এতেই আমাদের কাজ হয়ে যাবে। এই কোডটা আপনারা নিজে অনুশীলন করুন। বুঝতে সমস্যা হলে কমেন্ট করুন সবাই।

# মে‌মো‌রি ম্যানেজ‌মেন্ট: স্ট্যাক এবং হিপ মে‌মো‌রি।

প্রোগ্রাম লেখার সময় আমা‌দের ভ্যারিয়েবল নি‌য়েও কাজ কর‌তে হয়। এজন্য আমা‌দের দরকার হ‌লো মে‌মো‌রি ম্যানেজ‌মেন্ট সি‌স্টেম। আমরা যে টাই‌পেরই **ভ্যারিয়েবল** নেই না কেন, তা আমা‌দের সি‌স্টেম কোন না কোন ভা‌বে তা ম্যানেজ ক‌রে। আমরা এক‌টি ভ্যারি‌য়েবল যখন ডি‌ক্লেয়ার ক‌রি তখন তা **স্ট্যাক** মে‌মো‌রি অথবা **হিপ** মে‌মো‌রি‌তে জমা হয়। আজ‌কের আর্টিকেলে আমরা জান‌বো **স্ট্যাক এবং হিপ মে‌মো‌রির পার্থক্য** নি‌য়ে।

## স্ট্যাক কি?

আমরা প্রথ‌মে জে‌নে নেই স্ট্যাক কি? এটা হ‌লো ক‌ম্পিউটার মে‌মো‌রির একটা বি‌শেষ জায়গা যা কোন ফাংশন দ্বারা (**main()** ফাংশন সহ) তৈ‌রি টে‌ম্পোর‌রি বা অস্থায়ী ভ্যা‌রি‌য়েবল কে জমা ক‌রে রা‌খে। স্ট্যাক হ‌লো এক‌টি **LIFO (Last In First Out)** ডাটা স্ট্রাকচার। মা‌নে এখা‌নে যা‌কে সবার শে‌ষে রাখা হ‌বে, বের করার সময় তা‌কে প্রধান্য দেয়া হ‌বে।

স্ট্যাক মো‌মো‌রি য‌থেস্ট অপ‌টিমাইজড্। CPU খুব দ্রুত এখা‌নে কাজ কর‌তে পা‌রে। প্র‌তিবার যখন ফাংশ‌নে একটা ভ্যারি‌য়েবল ডি‌ক্লেয়ার করা হয় তখন এ‌কে স্ট্যা‌কে পুশ করা হয়। যখন ফাংশনটির কাজ শেষ হ‌য়ে যায় তখন যতগু‌লো ভ্যারি‌য়েবল স্ট্যা‌কে পুশ করা হই‌ছি‌লো তা‌দের ফ্রি ক‌রে দেওয়া হয়। যখন একটা স্ট্যাক ভ্যারি‌য়েবল ‌ফ্রি হয় তখন ঐ স্পেসটা আ‌রেক‌টি স্ট্যাক ভ্যা‌রি‌য়েবল এর জন্য প্রস্তুত হ‌য়ে যায়।

স্ট্যা‌ক এ ভ্যা‌রি‌য়েবল স্টোর করার সু‌বিধা হ‌লো, মে‌মো‌রিটা আ‌গে থে‌কেই রে‌ডি করা থা‌কে। নি‌জের হা‌তে মে‌মো‌রি Allocate ,বা Free কর‌তে হয় না। CPU স্ট্যাক মে‌মো‌রি‌তে খুব দ্রুত রি‌ডিং এবং রাই‌টিং অপা‌রেশন চালা‌তে পা‌রে।

যখন একটা ফাংশন এর কাজ শেষ হ‌য়ে যায়, তখন ঐ ফাংশন দ্বারা তৈ‌রি সব ভ্যা‌রি‌য়েবল অটোমেটিক পপ আউট হ‌য়ে যায়।মা‌নে দ্ড়ায় স্ট্যাক মে‌মো‌রি লোকাল প্রকৃ‌তির। এটা প্রোগ্রা‌মিং এ ভ্যা‌রি‌য়েবল স্কো‌পিং অথবা লোকাল ভ্যা‌রি‌য়েবল ক‌ন্সে‌প্টের এর সা‌থে সামন্জস্যপূর্ন।

C প্রোগ্রা‌মিং এ একটা সাধারন ভূল, যে কোন ফংশ‌নে একটা স্ট্যাক ভ্যা‌রি‌য়েবল তৈ‌রি ক‌রে ফাংশন‌টির কাজ শেষ হওয়ার পর তা আবার এক্সেস করার চেস্টা ক‌রা।

স্ট্যাক মে‌মো‌রির আ‌রেকটা গুরুত্বপূর্ন বিষয় হ‌লো, এখা‌নে ভ্যা‌রি‌য়েবল এর সাইজ এর এক‌টি নির্দিষ্ট লি‌মিট আ‌ছে। এর বে‌শি স্ট্যাক এ স্টোর করা যায় না। **এই লি‌মিট OS ভে‌দে আলাদা হ‌য়ে থাকে**।

### Code Example:

Stack Memory

C

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | #include <stdio.h>  int \*numberPointer(){      int stackVariable=9;//pushing a variable in stack      return &stackVariable; //returning the address will occure a warning.  }  int main() {      numberPointer();      return 0;  } |

উপ‌রের numberPointer() ফাংশনটি ভিত‌রে Allocate করা স্ট্যাক ভ্যা‌রি‌য়েবল, “stackVariable” এর Address return কর‌বে। যে‌হেতু “stackVariable” না‌মের ভ্যা‌রি‌য়েবলটা ফাংশনটির ভিত‌রে স্ট্যা‌টিক্যা‌লি ডি‌ক্লেয়ার করা হ‌য়ে‌ছে (লাইন ৩) তাই এই ভ্যা‌রি‌য়েবলটা স্ট্যাক মে‌মো‌রি‌তে পুশ হ‌বে, এবং প্রোগ্রামটি Exit করার সা‌থে সা‌থে ফ্রি হ‌য়ে যা‌বে। য‌দি এখন আমরা কোডটা রান করাই ত‌বে নি‌চের ম‌তো একটা ওয়ার্নিং দেখ‌তে পা‌বো। কারন ফাংশনটা যে এড্রেস রিটার্ন কর‌বে সেই এড্রেসটা রিটার্ন কর‌ার পরই ফ্রি হ‌য়ে যা‌বে।

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | In function 'numberPointer':    4:12: warning: function returns address of local variable [-Wreturn-local-addr]         return &stackVariable; //returning the address will occure a warning.                ^ |

**এখন একটা Overview করা যাক:**

* ফাংশ‌নে ভ্যা‌রি‌য়েবল পুশ বা পপ করার উপর নির্ভর ক‌রে স্ট্যাক বর্ধিত বা সংকুচিত হ‌তে পা‌রে।
* ‌নি‌জে নি‌জে মে‌মো‌রি ম্যানেজ‌মেন্ট করার কোন প্র‌য়োজন নেই। ভ্যা‌রি‌য়েবল নি‌জে নি‌জেই Allocate এবং ফ্রি হ‌য়ে যা‌বে।
* স্ট্যাক এর সাইজ লি‌মিট র‌য়ে‌ছে।
* যে ফাংশনটি স্ট্যাক ভ্যারি‌য়েবল তৈ‌রি ক‌রে, ঐ ফাংশ‌নের লাইফটাইম যতক্ষন স্ট্যা‌কে ভ্যা‌রি‌য়েবল এর অবস্থান ততক্ষন।

## ‌হিপ মে‌মো‌রি কি?

‌হিপ হ‌লো ক‌ম্পিউটার মে‌মো‌রির এক‌টি জায়গা যেখা‌নে আমা‌দের জন্য আ‌গে থে‌কেই মে‌মো‌রি ম্যানেজ করা থা‌কে না, বা CPU নি‌জেও অটো‌মে‌টিক্যা‌লি ম্যানেজ ক‌রে না। এ‌টি মে‌মো‌রির এক‌টি বিশাল অঞ্চল জু‌রে থা‌কে। হিপ (**Heap**) এ মে‌মো‌রি Allocation করার জন্য আমরা **malloc** বা **calloc** না‌মের C এর Built in functions ব্যবহার ক‌রে থা‌কি। একবার আপ‌নি হিপ এ মে‌মো‌রি Allocation কর‌লে একে কাজ শে‌ষে ফ্রি ( **free()** ) করার দ্বা‌য়িত্বটাও আপনার ঘা‌ড়ে। কাজটা ঠিক ম‌তো না কর‌তে পার‌লে আপনার প্রোগ্রা‌মে মে‌মো‌রি লিক এর ম‌তো সিচু‌য়েশন তৈ‌রি হ‌বে। মেমো‌রি লিক(Memory leak) একটা সি‌রিয়াস সমস্যা মূলত সেই ধর‌নের সফটওয়্যার এর জন্য যা অনেক সময় ধ‌রে রান ক‌রে।

হিপ‌ে মূলত কোন সাইজ লি‌মিট নেই, যেমনটা স্ট্যাক এর ক্ষে‌ত্রে দেখা যায়। হিপ মে‌মো‌রি‌তে Read এবং Write অপা‌রেশন একটু স্লো। কারন হিপ মে‌মো‌রি Access কর‌তে পয়েন্টার ব্যবহার কর‌তে হয়। স্ট্যা‌কের বিপরী‌তে, হিপ মে‌মো‌রি‌তে তৈ‌রি করা ভ্যা‌রি‌য়েবল প্রোগ্রাম এর যে‌কোন জায়গা থে‌কেই Access করা যায়।

### Code Example:

C

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #include <stdio.h>  #include <stdlib.h>  int \*numberPointer(){      int \*heapVariable = (int\*) malloc(sizeof(int));      return heapVariable;  }  int main() {      numberPointer();      return 0;  } |

উপ‌রে যে কে‌া‌ডের উদাহরণ দেয়া হ‌য়ে‌ছে, তা রান করা‌লে কোন ধর‌নের **Warning** দি‌বে না। কারন এখা‌নে আমরা যে মে‌মো‌রিটা Allocate ক‌রে‌ছি তা হি‌পে Allocate হ‌য়ে‌ছে (লাইন ৪)। তাই ফাংশন‌টি য‌দি Exit হ‌য়েও যায় ত‌বুও আমাদের কা‌ছে address টা Available থাক‌বে।

## স্ট্যাক মেমো‌রি বনাম হিপ মে‌মো‌রি

### স্ট্যাক মে‌মো‌রি

* খুব দ্রুত **এক্সেস** করা যায়।
* ‌নি‌জে নি‌জে ভ্যা‌রি‌য়েবল ফ্রি করার দরকার নেই।
* CPU এর দাড়াই স্পেস ম্যানেজ করা হয়।
* শুধু মাত্র লোকাল ভে‌রি‌য়েবল এর জন্য
* **স্ট্যাক সাই‌জে লি‌মিট আছে।**
* **ভ্যা‌রি‌য়েবল রিসাইজ করা যায় না।**

### হিপ মে‌মো‌রি

* ভ্যা‌রি‌য়েবল গু‌লো‌কে গ্লোবা‌লি এক্সেস করা যায়। (**কোন ভা‌বে এড্রেস পাস ক‌রে হ‌লেও**)
* ‌মে‌মো‌রি সাই‌জে লি‌মিট নেই।
* স্ট্যাক এর চে‌য়ে স্লো।
* ‌স্পেস এর সর্বোত্তম  ব্যবহার এর কোন গেরা‌ন্টি নেই 😉
* ‌নি‌জে নি‌জেই ম্যানেজ কর‌তে হ‌বে।
* **ভ্যারিয়েবল গু‌লো realloc() ব্যবহার ক‌রে resize করা যায়।**