

সূচিপত্র

ভূমিকা.....	১১
লেখক পরিচিতি.....	১৩
অধ্যায় ০ - শুরুর আগে.....	১৫
০.১ - বইটি কাদের জন্য.....	১৬
০.২ - বইটি কীভাবে পড়তে হবে	১৬
০.৩ - বইটিতে ব্যবহৃত চিহ্নসমূহের ব্যাখ্যা.....	১৭
অধ্যায় ১ - কম্পিউটার মেমোরি (Computer Memory).....	১৮
১.১ - বিট ও বাইট	১৯
১.২ - ভ্যারিয়েবলের অ্যাড্রেস বা ঠিকাবা	২১
১.৩ - বিভিন্ন প্রকারের মেমোরি	২৪
অধ্যায় ২ - পয়েন্টার	২৭
২.১ - নাল পয়েন্টার.....	৩৫
২.২ - স্ট্রিং ও পয়েন্টার.....	৩৭
২.৩ - পয়েন্টারের পয়েন্টার.....	৩৯
অধ্যায় ৩ - ফাইল (File).....	৪২
অধ্যায় ৪ - রিকুর্সন (Recursion).....	৫৩
৪.১ - লোকাল ও গ্লোবাল ভ্যারিয়েবল	৫৩
৪.২ - স্ট্যাটিক ভ্যারিয়েবল	৫৫
৪.৩ - বিভিন্ন প্রকারের মেমোরি	৫৫

কম্পিউটার প্রোগ্রামিং – বিত্তীয় খণ্ড

৮.৮ - রিকার্সন.....	৫৯
অধ্যায় ৫ - বিটওয়াইজ অপারেশন (Bitwise Operation).....	৮৫
অধ্যায় ৬ - স্ট্রাকচার (Structure) ও ইউনিয়ন (Union).....	৯১
৬.১ - স্ট্রাকচার (Structure).....	৯১
৬.২ - ইউনিয়ন (Union).....	১১২
৬.৩ - স্ট্রাকচারের মেমোরি অ্যালাইনমেন্ট.....	১১৪
অধ্যায় ৭ - আরও পয়েন্টার.....	১১৯
৭.১ - পয়েন্টারের হিসাব-নিকাশ.....	১২৬
৭.২ - ভয়েড পয়েন্টার (void pointer).....	১২৯
৭.৩ - ফাংশন পয়েন্টার.....	১৩০
৭.৪ - qsort ও bsearch ফাংশন.....	১৩৩
অধ্যায় ৮ - মজার কিছু প্রোগ্রাম.....	১৩৭
৮.১ - সময় পরিমাপ.....	১৩৭
৮.২ - র্যানডম নম্বর (random number) তৈরি.....	১৩৮
৮.৩ - নিজে হেডার ফাইল তৈরি করা.....	১৪০
অধ্যায় ৯ - বিবিধ.....	১৪২
৯.১ - কনস্ট্যান্ট (constant) ও ম্যাক্রো (Macro).....	১৪২
কনস্ট্যান্ট (constant).....	১৪২
ম্যাক্রো.....	১৪৩
৯.২ - এনিউমেরেশন (enumeration).....	১৪৬
৯.৩ - কমান্ড লাইন আর্গুমেন্ট (Command Line Argument).....	১৫০
৯.৪ - প্রোগ্রাম কম্পাইল হওয়ার ধাপসমূহ.....	১৫৪
৯.৫ - #typedef ও #define নিয়ে কিছু কথা.....	১৫৬

কম্পিউটার প্রোগ্রামিং – বিত্তীয় খণ্ড

৯.৬ - main() ফাংশন ও return ০.....	১৫৮
৯.৭ - lvalue এবং rvalue.....	১৬০
অধ্যায় ১০ - প্রোগ্রাম ডিবাগিং.....	১৬৪
১০.১ - ডিবাগিং কী?.....	১৬৪
১০.২ - সাধারণ ডিবাগিং.....	১৬৯
১০.৩ - কোডব্রকসে ডিবাগিং.....	১৭৪
অধ্যায় ১১ - যেতে হবে বহদুর.....	

একটি বেসরকারি বিশ্ববিদ্যালয়ে শিক্ষকতা দিয়ে কর্মজীবন ওক করলেও পরে একটি দেশি সফটওয়্যার নির্মাতা প্রতিষ্ঠানে কাজ করেন। তারপর যুক্তরাষ্ট্রীভিত্তিক আরেকটি সফটওয়্যার তৈরির প্রতিষ্ঠানে কাজ করার পর নিজেই প্রতিষ্ঠা করেন মুক্ত সফটওয়্যার লিমিটেড নামক সফটওয়্যার তৈরির একটি প্রতিষ্ঠান। অনলাইন কোর্সের মাধ্যমে প্রোগ্রামিং ও সফটওয়্যার তৈরির সফটওয়্যার তৈরির একটি প্রতিষ্ঠান। অনলাইন কোর্সের মাধ্যমে প্রোগ্রামিং ও সফটওয়্যার তৈরির সফটওয়্যার তৈরির একটি প্রতিষ্ঠান। এছাড়া তিনি বাংলাদেশ গণিত নাম বিষয় শিক্ষাদানের জন্য তৈরি করেছেন বিমিক কম্পিউটিং। এছাড়া তিনি বাংলাদেশ গণিত অলিম্পিয়াডে একজন একাডেমিক কাউন্সিলর। বর্তমানে লিড সফটওয়্যার ইঞ্জিনিয়ার হিসেবে কাজ করছেন সিঙ্গাপুরের গ্রাব (Grab) নামক প্রতিষ্ঠানে।

অধ্যায় ০ - শুরুর আগে

কম্পিউটার বলতে আমরা একসময় বুবাতাম একটি বিরাটকায় যন্ত্র, যা থাকবে সারা ঘরজুড়ে। তারপর আমাদের ধারণা হলো, কম্পিউটার এমন একটি যন্ত্র, যা টিভির মতো এবং একটি ছোট টেবিলেই সেটি রাখা সম্ভব (আসলে মনিটরই সবার চেয়ে পড়ত)। এই কম্পিউটারকে মানুষ নাম দিল ডেক্সটপ কম্পিউটার। কয়েক বছর পর দেখা গেল যে, কেউ কেউ এমন কম্পিউটার ব্যবহার করছে, যেটি কী না একটি অফিসে নিয়ে যোরাফেরা করা যায়। এর নাম হলো ল্যাপটপ কম্পিউটার, কারণ কম্পিউটারকে কোলে বসিয়ে কাজ করা যায়। ল্যাপটপ দ্রুতই জনপ্রিয় হলো কম্পিউটার, যেখানে বহুগুণে। সেই সঙ্গে ওজন (তর)-ও কমল। কিন্তু সৃষ্টিশীল মানুষ কি আর এর ক্ষমতাও হৃদি পেল বহুগুণে। সেই সঙ্গে ওজন (তর)-ও কমল। কিন্তু সৃষ্টিশীল মানুষ কি আর থেমে থাকে? মোবাইলফোনগুলো হঠাতে করে কম্পিউটারের কিছু গুণগুণ পেতে শুরু করল, আর তার নাম হয়ে গেল স্মার্টফোন। আমার নিজের প্রথম কম্পিউটারের সিপিইউ ছিল ৪৫০ মেগাহার্টজ। আর এখনকার বেশিরভাগ স্মার্টফোনের প্রসেসর সেই কম্পিউটারের চেয়ে অনেক বেশি শক্তিসম্পন্ন, আর সুতির তো বটেই (মানে মেমোরি বেশি)। তাও যদি মানুষ এবার একটু থামত। কিন্তু না, চলে আসল স্মার্ট ওয়াচ, হ্যাঁ, ঘড়ির ভেতর কম্পিউটার!

এই যে প্রযুক্তি এত দ্রুত গতিতে অগ্রসর হচ্ছে, নিয়ন্ত্রন যন্ত্র তৈরি হচ্ছে, এগুলোর সঙ্গে তাল মিলিয়ে সফটওয়্যার নির্মাতারা কীভাবে সফটওয়্যার বানাচ্ছেন? এত নতুন ল্যাঙ্গুয়েজ, ফ্রেমওয়ার্ক - এসব কীভাবে তারা শিখছে? এখন নতুন কেউ সফটওয়্যার নির্মাণখাতে আসতে চাইলে, তাকে কী কী শিখতে হবে? এসব প্রশ্ন আমাদের শিক্ষার্থীদের মাথায় ঘূরপাক থায়। যারা কম্পিউটার সায়েন্স পড়ছে, যারা ভবিষ্যতে কম্পিউটার সায়েন্স পড়ার জন্য কিছু করছে, যারা কম্পিউটার সায়েন্স পড়ছে না, কিন্তু পেশা হিসেবে সফটওয়্যার প্রকৌশলকে বেছে নিতে আগ্রহী - সবারই একই প্রশ্ন। উত্তরটি কিন্তু খুব সহজ, টেকনোলজির পেছনে না দোড়ে জানের ভিত্তা শুরু করতে হবে। মাটির নিচে ভিত্তি খুব শক্ত ও গভীর হলে যেমন অনেক বড় দালান ওঠানো যায়, তেমনি জানের ভিত্তিটা খুব শক্ত হলে নতুন জিনিস শেখাও খুব সহজ হয়ে যায়। তাই আমি সবসময় বলি, স্কুল-কলেজের পড়ালেখা ঠিকভাবে করতে হবে। পাঠ্যবইয়ের প্রতিটি জিনিস জানা থাকতে হবে (সেটি পরীক্ষায় আসুক আর নাই আসুক)। বাংলা ভাষায় শুন্দিভাবে পড়তে ও লিখতে জানতে হবে। পাটিগণিত, বীজগণিত, জ্যামিতি পড়তে হবে, জানতে হবে, বুবতে হবে। ইংরেজি ভাষাটিও শিখতে হবে, যতদূর পারা যায়। এছাড়া অন্যান্য বিষয়গুলোও পড়তে হবে। এভাবেই বেসিক ভালো হবে। তাহলে প্রোগ্রামিং শিখব কখন?

আমাদের স্কুল পর্যায়ে যেই সিলেবাস রয়েছে, তাতে যষ্ঠ শ্রেণি পর্যন্ত পড়ালেখা কেউ খুব ভালোভাবে করলে সে আসলে প্রোগ্রামিং শেখা শুরু করে দিতে পারে। আর স্কুলের পড়ালেখা তো

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

প্রোগ্রামিং শেখাটা কিন্তু আবার কথেক মাসের কোর্সের মতো নয়। চালিয়ে যেতে হবে বছরের পর বছর। কারও প্রোগ্রামিংয়ের সেদিক যখন খুব ভালো হবে, তখন যেকোনো নতুন প্রোগ্রামিং ভাষা, ফ্রেমওয়ার্ক, টেকনোলজি আয়ত্ত করে নিতে তার তেমন বেগ পেতে হবে না। তাই প্রোগ্রামিংয়ের বেসিক টিক করার পেছনে নিঠার সঙ্গে নিয়মিত অনুশীলন করে যেতে হবে। আর এর জন্য দের উপর হচ্ছে প্রোগ্রামিং সমস্যার সমাধান করা।

ইতিমধ্যে আমার দেখা কম্পিউটার প্রোগ্রামিং ১ম খণ্ড বইটি বাংলাদেশের শিক্ষার্থীদের মধ্যে ব্যাপক জনপ্রিয়। কেবল বিশ্ববিদ্যালয় আর পলিটেকনিকই নয়, স্কুল-কলেজের ছেলেমেয়েরাও বইটি পড়ে প্রোগ্রামিং শিখছে, প্রোগ্রামিং শেখার আগ্রহ পাচ্ছে। বিশ্বাসি আমার জন্য অতুল আনন্দের। সেই সূত্র ধরে আমি ও আরও দুজন সহলেখক মিলে লিখেছি ৫২টি প্রোগ্রামিং সমস্যা ও সমাধান বইটি। যেখানে ৫২টি প্রোগ্রামিং সমস্যা নিয়ে আলোচনা করা হয়েছে এবং সমস্যাগুলো যারা নতুন প্রোগ্রামিং শিখছে, তাদের জন্য উপযোগী। সেই ধারাবাহিকভাবে কম্পিউটার প্রোগ্রামিং ২য় খণ্ড বইটি লিখলাম। বইটিতে প্রোগ্রামিংয়ের একেবারে মৌলিক ধরণগুলোর পরে বেশগুলো জানা উচিত, যেমন পয়েন্টার, ফাইল, রিকর্ণ - এসব জিনিস নিয়ে আলোচনা করা হয়েছে। দুটি বইতেই আমি সি প্রোগ্রামিং ভাষা ব্যবহার করেছি। কারণ কম্পিউটার কীভাবে কাজ করে, এটি বুঝতে হলে প্রচলিত প্রোগ্রামিং ভাষাগুলোর মধ্যে সিই সবচেয়ে ভালো।

০.১ - বইটি কাদের জন্য

যেসব শিক্ষার্থী ইতিমধ্যে আমার কম্পিউটার প্রোগ্রামিং ১ম খণ্ড বইটি শেষ করেছে, এই বইটি মূলত তাদের জন্য। এছাড়া কেউ যদি ৫২টি প্রোগ্রামিং সমস্যা ও সমাধান বইটি পড়ে থাকে এবং সমস্যাগুলো সমাধানের চেষ্টা করে থাকে, তাহলে তার জন্য বইটি পড়া সহজ হবে।

০.২ - বইটি কীভাবে পড়তে হবে

বইতে প্রচুর উদাহরণ আছে, অনেক প্রোগ্রাম লিখে দেওয়া হয়েছে। তোমরা কিন্তু ভুলেও কেবল সেগুলো পড়ে চলে যাবে না, প্রোগ্রামগুলো দেখে দেখে টাইপ করবে, কম্পাইল করবে ও রান করবে। প্রোগ্রামের আউটপুট কেমন এল, কেন এল এগুলো নিজে মাথা খাটিয়ে চিন্তা করবে। বইয়ের ব্যাখ্যা বোার চেষ্টা করবে। বস্তুদেরকে জিজ্ঞাসা করবে, তাদের সঙ্গে আলাপ করবে। তারপরও কেনে জায়গায় খটকা থেকে গেলে অস্তির হবে না, সামনে এগিয়ে যাবে। বইটি যখন বিত্তীয়বার পড়বে, তখন অনেককিছুই পরিকল্পনা হবে। আর যারা বিশ্ববিদ্যালয়ের শিক্ষার্থী, তারা

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

চতুর্থ বর্ষের শেষদিকে বইটি আরেকবার পড়বে, কারণ চাকরির ইন্টারভিউতে কাজে লাগতে পারে।

০.৩ - বইটিতে ব্যবহৃত চিহ্নসমূহের ব্যাখ্যা

০.৩ - বইটিতে ব্যবহৃত চিহ্নসমূহের ব্যাখ্যা
বইতে কিছু কিছু অংশে বিভিন্ন চিহ্ন ব্যবহার করা হয়েছে। চিহ্নগুলোর কোনটি কী বোঝায় তা এখানে আলোচনা করা হলো।

● নোট:

এই জাতীয় বাক্সে সাধারণত বিশেষ কোনো তথ্য দেওয়া থাকবে যা সমস্কে অনেকেরই ভুল ধরাগু থাকে অথবা যা বিশেষভাবে লক্ষ না করলে ভুল হওয়ার সন্তানবন্ধন থাকে।

● টিপস:

এই জাতীয় বাক্সে সাধারণত বিশেষ কোনো তথ্য দেওয়া থাকবে যা সচরাচর ভুলে যাওয়ার সন্তানবন্ধন থাকে।

● ফাংশন প্রোটোটাইপ:

এই জাতীয় বাক্সে দেওয়া থাকবে আমরা নতুন যেসব ফাংশন শিখছি তার বিবরণ।

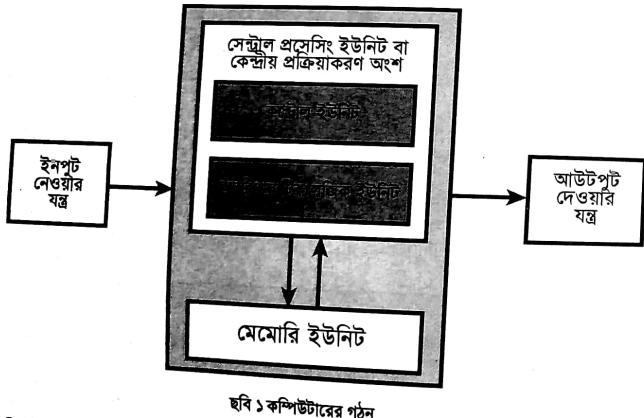
● প্রশ্ন / উত্তর:

কিছু কিছু বিষয়ের ব্যাখ্যা প্রশ্নের আকারে দেওয়া থাকবে।

অধ্যায় ১ – কম্পিউটার মেমোরি (Computer Memory)

আমরা সচরাচর যেসব কম্পিউটার দেখি (যেমন: ডেস্কটপ, ল্যাপটপ), সেসব কম্পিউটারের থাকে –

- ইনপুট মেওয়ার যন্ত্র। যেমন মাউস, কিবোর্ড। এদের কাজ হচ্ছে কম্পিউটারে ইনপুট মেওয়ার ব্যবহাৰ কৰা।
- আউটপুট মেওয়ার যন্ত্র। যেমন মনিটর, প্ৰিন্টাৰ। এদের কাজ হচ্ছে আউটপুট দেখানোৰ ব্যবহাৰ কৰা।
- তথ্য ধাৰণ কৰাৰ যন্ত্ৰ বা স্টোৱেজ ডিভাইস। যেমন পেন ড্রাইভ, হার্ড ডিক্ষন, র্যাম। এদেৱ কাজ হচ্ছে হারীভাবে বা অহারীভাবে তথ্য সংৰক্ষণ কৰা।
- কেন্দ্ৰীয় প্ৰক্ৰিয়াকৰণ অংশ বা সেন্ট্রাল প্ৰসেসিং ইউনিট (Central Processing Unit), সংক্ষেপে সিপিইউ (CPU)। যেমন মাইক্ৰোপ্ৰসেসৱ। এৱ কাজ হচ্ছে বিভিন্ন হিসাৰ-নিকাশ কৰা।



কম্পিউটারের মেমোরি কী দৰকাৰ? প্ৰসেস কৰতে পাৱলৈই তো হয়ে যায়। - এৱকম চিতা তোমোৱা অনেকেই কৰ। কিন্তু কম্পিউটার যে প্ৰসেস কৰবে, কী প্ৰসেস কৰবে? নিচয়ই ডেটা। আৱ কেন জিনিসটি প্ৰসেস কৰতে হবে?

১.১ – বিট ও বাইট

তোমোৱা তো ইতিমধ্যেই জেনে পিয়েছ যে, কম্পিউটাৰ যে হিসাৰ-নিকাশ কৰে, সেগুলো যত বড় বড় হিসাৰই হোক না কেন, শেষ পৰ্যন্ত কম্পিউটাৰ কেবল শূন্য আৱ এক ব্যবহাৰ কৰে হিসাৰ কৰে (এই হিসাৰ কৰাটাকেই আমোৱা প্ৰসেস কৰা বলছি মাঝে মধ্যে)। তো এই ০ এবং ১-গুলো রাখাৰ জন্য আমাদেৱ জায়গা দৰকাৰ। সেই জায়গাটি দেয় কম্পিউটাৰেৰ মেমোৱি। এই ০ আৱ ১, এগুলো হচ্ছে একেকটি বিট (ইংৰেজিতে bit)। একটি বিট মেকোনো এক রকমেৰ হবে, হয় শূন্য এক (0), না হয় এক (1)। তাহলে একটি বিট দিয়ে দুটি আলাদা জিনিস প্ৰকা৶ কৰা সন্তুষ্ট, যখন (0), না হয় এক (1)। তাহলে একটি বিট দিয়ে দুটি আলাদা জিনিস রাখা যায়? তোমোৱা একটু চিতা কৰ তো। চিতা কৰা শেষ হলে নিচেৰ টেবিলটি দেখো:

১ম বিট	২য় বিট
•	•
০	১
১	০
১	১

তাহলে দুটি বিট পাশাপাশি রাখলে চাৰটি আলাদা জিনিস রাখা যায়: 00, 01, 10 ও 11। এবাৱে তোমোৱা কিছু কাজ কৰতে হবে। খাতা-কলম নিয়ে বসে যাও এবং তিনটি ও চাৰটি বিটেৰ ক্ষেত্ৰে কতগুলো আলাদা জিনিস রাখা সন্তুষ্ট সেটি বেৱ কৰ। উভাৱ হবে, যথাক্রমে 8 ও 16। তাহলে দাঁড়াছে 2, 3 ও 4টি কৰে বিটেৰ ক্ষেত্ৰে যথাক্রমে 4, 8 ও 16টি পৃথক জিনিস রাখা যায়। তাহলে তোমাদেৱ যেহেতু বুদ্ধিশক্তি বেশি, তোমোৱা সহজেই বলে দিতে পাৱবে, 5টি বিটেৰ জন্য 32, 6টি বিটেৰ জন্য 64, 7টি বিটেৰ জন্য 128টি পৃথক জিনিস রাখা সন্তুষ্ট। তাহলে 8টি বিটেৰ জন্য কত? উভাৱ হবে 256। আৱ যাদেৱ সামান্য একটু গাণিতিক বুদ্ধি আছে, তাৱা কিন্তু এতক্ষণে মনে মনে একটি সূত্ৰ বেৱ কৰে ফেলেছ: nটি বিট থাকলে 2^n টি পৃথক জিনিস রাখা সন্তুষ্ট।

বাইট শব্দটি নিচয়ই তোমোৱা শুনেছ? ইংৰেজিতে byte (তবে এই শব্দেৱ মানে কিন্তু কামড় নয়, সেটিৰ ইংৰেজি byte)। আটটি বিট পাশাপাশি রেখে তৈৰি হয় একটি বাইট। তাহলে এক বাইটে কতটি আলাদা জিনিস রাখা যায়? আমি জানি তোমোৱা চাট কৰে বলে দিতে পাৱবে, 256টি।

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

তোমরা সবাই ক্যারেষ্টার টাইপের (char) ভ্যারিয়েবলের সঙ্গে পরিচিত। তাহলে এখন তোমরা নিচ্ছাই বুঝতে পারছ যে, একটি ক্যারেষ্টার টাইপের ভ্যারিয়েবলে 256টি আলাদা জিনিস রাখা সম্ভব। এজনাই আমরা বিজ্ঞ অক্ষর, সংখ্যা, চিহ্ন এগুলোকে ক্যারেষ্টার টাইপের ভ্যারিয়েবলের মধ্যে রাখি। প্রতিটি অক্ষরের একটি মান রয়েছে যাকে সংখ্যা দিয়ে প্রকাশ করা যায়। যেমন a-এর মান 97, b-এর মান 98, c-এর মান 99, এভাবে z-এর মান 122। আবার A-এর মান 65, B-এর মান 66, C-এর মান 67, এভাবে Z-এর মান 90, 0 থেকে 9 পর্যন্ত চিহ্নগুলোর মান যথাক্রমে 48 থেকে 57। এই মানগুলোকে বলে ASCII মান। ASCII হচ্ছে American Standard Code for Information Interchange-এর সংক্ষিপ্তরূপ। ইংরেজি সব অক্ষর, চিহ্ন সবগুলোরই আসকি (ASCII) মান রয়েছে। এখন তোমার মনে প্রশ্ন জাগতে পারে, পৃথিবীর বাকি ভাষাগুলো কী দোষ করল? তাদের সব অক্ষরের জন্য আসকি মান নেই কেন? আসলে 8 বিটে আমরা তো মোটে 256টি জিনিস রাখতে পারি, তাই সবার জ্ঞায়গা দেওয়া সম্ভব হয় নি। অন্য সব ভাষার জন্য রয়েছে ইউনিকোড (Unicode)।

এখন আসো, চট করে কয়েকটি প্রোগ্রাম লিখে ফেলি। তোমরা অবশ্যই কোডগুলো নিজে টাইপ করবে এবং রান করবে। প্রোগ্রামগুলো আমি ব্যাখ্যা করব না, তোমরা নিজেরা চিন্তা করে বুঝে নেবে। আমাদের শিক্ষার্থীদের মূল সমস্যা হচ্ছে অলস মতিষ্ক, চিন্তা করে কিছু বুঝতে চায় না। আমি চাই তোমরা তোমাদের মতিষ্ক একটু খাটোও।

আমাদের প্রথম প্রোগ্রাম:

```
1 #include <stdio.h>
2 int main()
3 {
4     int i;
5
6     for (i = 33; i <= 126; i++) {
7         printf("ASCII code for %c is %d\n", i, i);
8     }
9
10    return 0;
11 }
```

প্রোগ্রাম ১-১

ওপরের প্রোগ্রামে যদি 127-এর চেয়ে বড় আসকি ক্যারেষ্টারগুলো প্রিন্ট করতে চাইতে, সেটি সম্ভব হবে না, কারণ 127 থেকে 255 পর্যন্ত আসকি ক্যারেষ্টারগুলো হচ্ছে নন-প্রিন্টেবল (মানে এগুলো ক্রিনে দেখানো যাবে না, আর দেখালেও হিজিবিজি আউটপুট আসবে)।

২০

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

আমাদের বিত্তীয় প্রোগ্রাম:

```
1 #include <stdio.h>
2
3 int main()
4 {
5     char small_letter, capital_letter;
6
7     printf("Please enter a small letter: ");
8     small_letter = getchar();
9     capital_letter = small_letter - 32;
10    printf("The capital letter is: %c\n", capital_letter);
11
12    return 0;
13 }
```

প্রোগ্রাম ১-২

এখন তোমার কাজ হবে একটি ফাংশন লেখা, যেটি প্যারামিটার হিসেবে একটি ক্যারেষ্টার এবং তাইপের ভ্যারিয়েবল নেবে এবং সেটি যদি কোনো অক্ষর (digit) হয়, তবে 1 রিটার্ন করবে, আর না হলে 0 রিটার্ন করবে। কাজ শেষ হলে চলো, আমরা মেমোরি নিয়ে আলাপ-আলোচনায় কেরত যাই।

ইন্দিজার টাইপের ভ্যারিয়েবলের আকার হচ্ছে চার বাইট, মানে 32 বিট। এই 32 বিটে আমরা 2³² বা 4,294,967,296টি সংখ্যা রাখতে পারি। এখন আমরা যদি কেবল ধনাত্মক সংখ্যা রাখতে চাই, তাহলে 0 থেকে 4,294,967,295 পর্যন্ত রাখা যাবে। এ ধরনের ভ্যারিয়েবলকে বলে unsigned ভ্যারিয়েবল। আর যেসব ভ্যারিয়েবলে ধনাত্মক ও ঋণাত্মক - উভয় প্রকারের সংখ্যা রাখা যায়, তাকে বলে signed ভ্যারিয়েবল, তবে এক্ষেত্রে আলাদাভাবে সেটির উল্লেখ থাকে না। রাখা যায়, তাকে বলে signed ভ্যারিয়েবল, তবে এক্ষেত্রে আলাদাভাবে সেটির উল্লেখ থাকে না। তাহলে আমরা বুঝতে পারলাম কেন int টাইপের ভ্যারিয়েবলে -2,147,483,648 থেকে 2,147,483,647 পর্যন্ত সংখ্যা রাখা যায় আর কেন unsigned int টাইপের ভ্যারিয়েবলে 0 থেকে 4,294,967,295 পর্যন্ত সংখ্যা রাখা যায়।

১.২ - ভ্যারিয়েবলের অ্যাড্রেস বা ঠিকানা

কম্পিউটারের মেমোরিতে স্ক্রিনতম একক হচ্ছে বাইট। পরপর অনেকগুলো বাইট মেমোরিতে সাজানো থাকে। প্রতিটি বাইটের আবার নির্দিষ্ট ঠিকানা বা অ্যাড্রেস (address) রয়েছে। তুমি যখন কোনো ক্যারেষ্টার ভ্যারিয়েবল ডিক্রেয়ার করবে, সেটি মেমোরির একটি বাইট দখল করে

২১

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

ফেলবে। আবার তুমি যদি ইন্টিজার ভ্যারিয়েবল ডিক্লেয়ার কর, তাহলে সেটি মেমোরির পরপর চারটি বাইট দখল করবে। ব্যাপারটি বৈকানোর জন্য নিচের ছবিটির মতো চিন্তা করা যাব। তোমরা দেখতে পাচ্ছি ঘরের আকার হচ্ছে এক বাইট। আবার প্রতিটি ঘরের আলাদা ঠিকানা বা অ্যাড্রেস রয়েছে। ধরে নিই, প্রথম ঘরের অ্যাড্রেস হচ্ছে 100। তাহলে পরের ঘরের অ্যাড্রেস হবে 101, তার পরের ঘরের অ্যাড্রেস হবে 102, এরকম। ছবিটে অ্যাড্রেসগুলো মূল ঘরের নিচে আরেকটি ঘরে লেখা হয়েছে।

ডেটা →	বাইট									
ঠিকানা →	100	101	102	103	104	105	106	107	108	109

এখন তুমি কোনো ভ্যারিয়েবল ডিক্লেয়ার করলে সেটি মেমোরির কোন ঘরটি দখল করবে, তা কিন্তু তোমার হাতে নেই, সেটি নির্ধারণ করবে অপারেটিং সিস্টেম। আবার তুমি যখন কোনো আরে ডিক্লেয়ার কর, আরের উপাদানগুলো কিন্তু মেমোরিতে পরপর জায়গা দখল করে।

এবারে নিচের প্রোগ্রামটি বাট্টপ্ট টাইপ করে রাখ কর।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     char ch1 = 'A', ch2 = 'B';
6     int n1 = 100, n2 = 100000;
7
8     printf("Value of ch1 = %c,\t", ch1);
9     printf("Address of ch1 = %p\n", &ch1);
10
11    printf("Value of ch2 = %c,\t", ch2);
12    printf("Address of ch2 = %p\n", &ch2);
13
14    printf("Value of n1 = %d,\t", n1);
15    printf("Address of n1 = %p\n", &n1);
16
17    printf("Value of n2 = %d,\t", n2);
18    printf("Address of n2 = %p\n", &n2);
19
20
21 }
```

প্রোগ্রাম ১-৩

২২

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

আউটপুট কী? একেক জনের আউটপুট একেক রকম হবে, কারণ মেমোরির কোন জায়গায় কোন ভ্যারিয়েবল থাকবে, তার কোনো ঠিক নেই। এখন তোমরা লক্ষ কর, কোনো ভ্যারিয়েবল যদি n হয়, তবে তার অ্যাড্রেস হচ্ছে &n। এ কারণেই আমরা scanf ফাংশন ব্যবহার করে কোনো ভ্যারিয়েবল ইনপুট নেওয়ার জন্য ভ্যারিয়েবলের নামের আগে & চিহ্ন ব্যবহার করি। কিন্তু আরের বেলাতে সেটি করতে হয় না, কারণ আরের নামটিতেই আরের শুরুর অ্যাড্রেসটি থাকে। আরের বেলাতে সেটি করতে হয় না, কারণ আরের নামটিতেই আরের অ্যাড্রেসটি থাকে। অর্থাৎ কোনো আরে যদি হয় `int ara[10]`, সেখানে ara-এর মান প্রিন্ট করলেই আরের অ্যাড্রেস পাওয়া যাব। আর কোনো ভ্যারিয়েবলের অ্যাড্রেস প্রিন্ট করার জন্য আমরা %p ব্যবহার করতে পারি।

এখন তুমি নিচের প্রোগ্রামটি আগের মতো রাখ করবে।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int ara[5] = {50, 60, 70, 80, 90};
6
7     printf("Value of Array: %d, %d, %d, %d, %d\n", ara[0],
8            ara[1], ara[2], ara[3], ara[4]);
9     printf("Address of ara is %p\n", ara);
10    printf("Address of ara[0] is %p\n", &ara[0]);
11    printf("Address of ara[1] is %p\n", &ara[1]);
12    printf("Address of ara[2] is %p\n", &ara[2]);
13    printf("Address of ara[3] is %p\n", &ara[3]);
14    printf("Address of ara[4] is %p\n", &ara[4]);
15
16    return 0;
17 }
```

প্রোগ্রাম ১-৪

প্রোগ্রামটির প্রথম লাইনে প্রিন্ট হবে আরের উপাদানগুলো। দ্বিতীয় লাইনে প্রিন্ট হবে আরের শুরুর অ্যাড্রেস, মানে মেমোরির যেই জায়গা থেকে আরেটি শুরু হয়েছে, সেই জায়গার ঠিকানা (একেক জনের কম্পিউটারে আউটপুট একেক রকম হবে)। দ্বিতীয় লাইনে যা প্রিন্ট হবে, তৃতীয় লাইনে ঠিক তাই প্রিন্ট হবে, কারণ আরের শুরুর অ্যাড্রেস আর আরের প্রথম উপাদানের অ্যাড্রেস একই জিনিস। চতুর্থ লাইনে আরের দ্বিতীয় উপাদানের অ্যাড্রেস প্রিন্ট হবে, যা প্রথম উপাদানের অ্যাড্রেসের চেয়ে চার বেশি। কারণ প্রথম উপাদান পরপর চারটি বাইট দখল করবে। যদিও তোমার মনে হতে পারে, 50 কেন চার বাইট জায়গা নেবে, এক বাইটেই তো লিখা যাব। কিন্তু যখনই তুমি ডেটা টাইপ int বলে দিবে, তখনই চার বাইট জায়গা দখল হবে যাবে। যদিও

২৩

কম্পিউটার প্রোগ্রামিং - বিভিন্ন ধরণ

বিভিন্ন ডেটা টাইপের ভ্যারিয়েবলের সাইজ, কম্পিউটারের আর্কিটেকচারতে ডিম্প রকম হচ্ছে পারে, কিন্তু আমি ধরে নিছি, তোমার কম্পিউটারের ইন্টিজারের সাইজ চার বাইট। আর সেটি সম্ভব না হলে পরের প্রোগ্রামটি চালিয়ে তুমি সত্যাগ্রহ জেনে নিতে পারবে।
সি-তে `sizeof` নামে একটি অপারেটর আছে, যেটি বলে দেয় কোন ভ্যারিয়েবলের সাইজ কত বাইট। নিচের প্রোগ্রামটি হ্যাচ টাইপ কর, তারপরে কম্পাইল ও রান কর। এখন নিজে নিজে চিন্তা করে বুঝে নাও যে `sizeof` কীভাবে ব্যবহার করতে হয়।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int num;
6     char ch;
7     double d_num;
8     float f_num;
9
10    printf("%lu\n", sizeof(int));
11    printf("Size of int: %d\n", sizeof(num));
12    printf("Size of char: %d\n", sizeof(ch));
13    printf("Size of double: %d\n", sizeof(d_num));
14    printf("Size of float: %d\n", sizeof(f_num));
15
16    return 0;
17 }
```

প্রোগ্রাম ১-৫

১.৩ - বিভিন্ন প্রকারের মেমোরি

এখন চলো জেনে নিই, কম্পিউটারে কত ধরনের মেমোরি আছে।

কম্পিউটারের মেমোরিকে মোটা দাগে দুই ভাগে ভাগ করা যায়, অস্থায়ী ও স্থায়ী। ইংরেজিতে বলে ভোল্যাটাইল (volatile) ও নন-ভোল্যাটাইল (non volatile)। যেসব মেমোরিতে কম্পিউটার বক্ষ করার পরেও ডেটা সংরক্ষিত থাকে, তাকে বলে স্থায়ী (non volatile) মেমোরি, যেমন ক্ষেত্রবিশেষে প্রোগ্রাম বক্ষ) করলে হারিয়ে যায়, যেগুলোকে বলে অস্থায়ী মেমোরি, যেমন র্যাম

কম্পিউটার প্রোগ্রামিং - বিভিন্ন ধরণ

(RAM)। কম্পিউটার প্রোগ্রামগুলো ডেটা নিয়ে কাজ করার সময় অস্থায়ী মেমোরি ব্যবহার করে। স্থায়ী মেমোরিগুলো বেশ ধীরগতির হয় বলে সেগুলো ব্যবহার করা হয় না।



ছবি ২: বিভিন্ন ধরনের মেমোরি

কম্পিউটারের প্রসেসরের মধ্যেও কিন্তু মেমোরি আছে, প্রসেসরের সবচেয়ে কাছে থাকে রেজিস্টার, আর তার পরেই থাকে ক্যাশ মেমোরি। সি ল্যাঙ্গুয়েজে আমরা চাইলে কোনো ভ্যারিয়েবলকে রেজিস্টারে রাখার জন্য এভাবে ডিক্লেয়ার করতে পারি: `register int number;`। সেসব ভ্যারিয়েবলকেই আমরা রেজিস্টারে রাখার চেষ্টা করব যেগুলো প্রোগ্রামের ডেটার সবচেয়ে বেশি ব্যবহার করা হয়।

তবে আজকাল রেজিস্টারে এভাবে না রাখলেও চলে, কম্পাইলারগুলো নিজেরাই বুঝে নেয় কোন ভ্যারিয়েবল কোথায় রাখতে হবে। রেজিস্টারের চেয়ে ক্যাশ মেমোরির আকার বড়, যানে বেশি তথ্য ধারণ করতে পারে, তবে গতি একটু কম। তারপরে আসে র্যাম। র্যাম প্রসেসরের বাইরে মাদারবোর্ডে সংযুক্ত থাকে। ক্যাশের তুলনায় র্যামের আকার বেশ বড়, তবে গতিও কম। এখন তোমরা প্রশ্ন করতে পারো, রেজিস্টার আর ক্যাশের আকার আরও বেশি হলে কী সমস্যা ছিল? শুধু শুধু র্যামের ব্যবহার কেন করছি? আসলে রেজিস্টার মেমোরি তৈরিতে খরচ সবচেয়ে বেশি, তারপরে ক্যাশ মেমোরি। র্যাম তাদের তুলনায় বেশ সত্ত্ব। র্যামের পরে আসে ভার্চুয়াল মেমোরি।

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

র্যামে যখন জায়গা হয় না, তখন হার্ডডিস্কের একটা অংশকে কম্পিউটারের অপারেটিং সিস্টেম মেমোরি হিসেবে ব্যবহার করতে দের প্রোগ্রামগুলোকে। সেটি অবশ্যই র্যামের তুলনায় আবেক্ষণিকভাবে ধীর গতির। ছবি-২ এর দিকে একবার চেক কুলিয়ে নাও।

এখানে বিষয়গুলো একটু সহজবোধ্য করে লেখা হয়েছে, তোমরা এগুলো আরও ভালোভাবে শিখতে পারবে যখন তোমরা কম্পিউটার আর্কিটেকচারের ওপর লেখাপড়া করবে। আপাতত মেমোরি নিয়ে এটুকু ধারণা থাকলেই চলবে। আর আমরা এখন প্রোগ্রামিং করার সময় মেমোরি বলতে র্যামকেই বুঝব, বইয়ের বাকি অংশেও সেভাবে লেখা হবে। পরবর্তী অধ্যায়ে আমরা পয়েন্টার নিয়ে আলোচনা করব।

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

অধ্যায় ২ - পয়েন্টার

পয়েন্টার শব্দটি নিচ্ছয়ই তোমরা অনেকবার শুনেছ। অনেকে হয়তো তোমাদের তয় দেখিয়েছে যে, পয়েন্টার খুব কঠিন জিনিস। একবার তো কোনো এক বইতে এমনও লেখা দেখেছিলাম যে, একজন ভালো সি প্রোগ্রামার হচ্ছে সেই ব্যক্তি, যে পয়েন্টার ছাড়া বাকি সব কিছু ভালো বোবে! কি, তয় পেয়ে গেলে? পয়েন্টার মোটেও ভয়ের কোনো বিষয় নয়। তুমি নিশ্চিত থাকতে পারো যে, তুমি এতক্ষণে পয়েন্টার বোঝার জন্য তৈরি হয়ে শিয়েছ এবং এই অধ্যায়টি ঠিকভাবে পড়লে, তুমি হরহামেশাই ব্যবহার করতে পারবে।

আলোচনা করার আগেই আমরা, চল, নিচের প্রোগ্রামটি লিখে কম্পাইল ও রান করি। তোমরা আলোচনা করার আগেই আমরা, চল, নিচের প্রোগ্রামটি লিখে কম্পাইল ও রান করা ও রান অনেকেই হয়তো বইটি পড়ার সময় আমার কথা মতো প্রোগ্রাম টাইপ করা, কম্পাইল করা ও রান করার কাজটি করছ না, তাদের বিষ্ট শেখাটা ঠিকঠাক হবে না। নিজে হাতেকলমে কাজ করা ও সঙ্গে সঙ্গে চিন্তা করে শেখার চাইতে ভালো পদ্ধতি আর নেই।

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int x = 10;
6
7     printf("Value of x is %d\n", x);
8     printf("Address of x is %p\n", &x);
9
10    return 0;
11 }
```

প্রোগ্রাম ২-১

প্রোগ্রামটি রান করলে আউপুট দেখবে এমন :

```
Value of x is 10
Address of x is 0x0060FF0C
```

তাহলে আমরা দেখতে পাচ্ছি x-এর একটি মান আছে, আর আছে অ্যাড্রেস বা ঠিকানা। আমরা আগের অধ্যায়েই ব্যাপারটি জেনেছি। এখন আমরা যদি ভ্যারিয়েবলের মান নিয়ে কাজ করার বদলে তার ঠিকানা নিয়ে কাজ করতাম, তাহলে কেমন হতো? তোমাদের মনে প্রশ্ন আসতে পারে,

কম্পিউটার প্রোগ্রামিং - বিভিন্ন খণ্ড

ভ্যারিয়েবলের ঠিকানা নিয়ে কাজ করার দরকার কী? এতদিন পর্যন্ত তো মান দিয়েই ঠিকঠাক প্রোগ্রাম লিখে এসেছি। আসলে দরকার আছে, কিন্তু বিষয়গুলো ঠিক এখনই বুঝবে না, পরে বুঝবে। আমি এখন বোঝানোর চেষ্টা করলে তোমরা তুল বুঝবে, তাই তুল বোঝার চেয়ে না বোঝা ভালো। আপাতত জেনে রাখো যে, এটি একটি গুরুত্বপূর্ণ বিষয়।

পয়েন্টার হচ্ছে একটি বিশেষ ধরনের ভ্যারিয়েবল যেটি আরেকটি ভ্যারিয়েবলের ঠিকানা রাখতে পারে। ইন্টিজার টাইপের ভ্যারিয়েবলের জন্য ইন্টিজার টাইপের পয়েন্টার ব্যবহার করতে হয়, ক্যারেক্টার টাইপের ভ্যারিয়েবলের জন্য ক্যারেক্টার টাইপের পয়েন্টার, ড্বল বা ফ্লোটের জন্য সেই টাইপের পয়েন্টার।

নিচের প্রোগ্রামটি লিখে কম্পাইল ও রান করে দেখো:

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int x = 10;
6     int *p;
7
8     p = &x;
9
10    printf("*p = %d\n", *p);
11    printf("Value of p is %p\n", p);
12
13    return 0;
14 }
```

প্রোগ্রাম ২-২

প্রোগ্রামটি রান করলে এরকম আউটপুট পাবে:

```

10
Value of p is 0x0060FF08
```

লক্ষ কর, `[int *p]` স্টেমেন্ট ব্যবহার করে আমরা একটি ইন্টিজার টাইপের পয়েন্টার ডিক্রেয়ার করলাম যার নাম `p`। `p`-তে আমরা ইন্টিজার টাইপের ভ্যারিয়েবলের ঠিকানা রাখতে পারব। সেটিই আমরা করেছি `p = &x;` লাইনে। `x` নামক ইন্টিজার ভ্যারিয়েবলের ঠিকানা আমরা `p`-তে রাখলাম। এখন `p`-এর মান প্রিন্ট করলে আমরা `x`-এর ঠিকানা পাব। আর `x`-এর মান পাওয়ার জন্য কী ব্যবহৃত? সেটি হচ্ছে `*p` প্রিন্ট করা। ওপরের উদাহরণটি যদি তোমরা টাইপ করে থাক, তাহলে তোমার কম্পিউটার স্ক্রিনের দিকে আবার লক্ষ কর এবং বিষয়টি তোমার কাছে

কম্পিউটার প্রোগ্রামিং - বিভিন্ন খণ্ড

পরিকার হয়েছে কি না দেখো, আর যারা প্রোগ্রাম নিজে টাইপ করে কম্পাইল ও রান করছ না, তাদের জন্য সমবেদনা রাইল।

● নোট:

যখন আমরা লিখি `int *p`, তখন একে পড়ি এভাবে, "integer pointer p"
যখন আমরা লিখি `*p`, তখন একে পড়ি এভাবে, "content of p"

এখন আমরা একটি পরীক্ষামূলক প্রোগ্রাম লিখব। আমি তো একটু আগে বলেছি যে ভ্যারিয়েবল এটি টাইপের হবে, পয়েন্টারও সে টাইপের হতে হবে। সেটি না হলে কী হবে? বেশি চিন্তা না করে যে টাইপের হবে, পয়েন্টারও সে টাইপের হতে হবে।

প্রোগ্রাম লিখে দেখি।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     double pi = 3.14159265358;
6
7     int *ptr;
8
9     ptr = &pi;
10
11    printf("Value of pi : %lf\n", pi);
12    printf("Value of pi : %lf\n", *ptr);
13
14    return 0;
15 }
```

প্রোগ্রাম ২-৩

প্রোগ্রাম কম্পাইল করতে গেলে কম্পাইলার তোমাকে এই বার্তা দিয়ে সতর্ক করে দিবে :

```
warning: incompatible pointer types assigning to 'int *' from
'double *'
```

অথবা এরকম এরর দিবে :

```
error: cannot convert 'double*' to 'int*' in assignment
```

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

এখন আমরা আরেকটি কাজ করব। একটি ইন্টিজার টাইপের পয়েন্টারে ইন্টিজার ভ্যারিয়েবলের ঠিকানা রাখব, তারপর পয়েন্টারের মান পরিবর্তন করে দিব। তাহলে দেখা যাবে মূল ভ্যারিয়েবলের মান পরিবর্তন হয়ে গিয়েছে।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int x = 10;
6     int *p;
7
8     p = &x;
9
10    printf("Value of x: %d\n", x);
11
12    *p = 20;
13
14    printf("Value of x: %d\n", x);
15    printf("Address of x: %p\n", &x);
16    printf("Value of p: %p\n", p);
17
18    return 0;
19 }
```

প্রোগ্রাম ২-৪

আউটপুট হবে এরকম:

```

Value of x: 10
Value of x: 20
Address of x: 0x0060FF08
Value of p: 0x0060FF08
```

আর হাঁ, আড্রেস কিন্তু তোমার কম্পিউটারে অন্যরকম আসবে, তবে দুটি অ্যাড্রেস সমান হবে।

তো এরকম কেন হলো? x-এর মান কেন পরিবর্তিত হলো? এমনটিই তো হওয়ার কথা। আমি তো p পয়েন্টারে x-এর ঠিকানা রেখেছি আর তারপর ***p = 20;** (পড়ব content of p = 20) লিখেছি, অর্থাৎ আমি p পয়েন্টারে যেই ঠিকানা আছে, সেই ঠিকানায় যেই ভ্যারিয়েবল আছে, সেখানে 20 অ্যাসাইন করেছি। তাই x-এর মান এখন 20। আমি যদি তোমাকে বলি যে, দ্বিমিক কম্পিউটারের অফিসে একটি বই পোছে দিতে, তাহলে তুমি সেটি করতে পারবে কারণ তুমি হয়তো নিজেই সেই অফিস চেন কিংবা তোমার বন্ধুদের কাছে খোঁজখবর নিয়ে অফিসটি চিনে

৩০

কম্পিউটার প্রোগ্রামি-২ - বিত্তীয় খণ্ড

নিতে পারবে। আর আমি যদি তোমাকে বলি ফ্ল্যাট নম্বর w, বাসা নম্বর x, সড়ক নম্বর y, এলাকার নাম z, তাহলেও কিন্তু তুমি সঠিক জায়গায় যেতে পারবে, এটি বিষিকের অফিস কি না সেটি তোমার না জানলেও চলবে।

আরেকটি উদাহরণ দিই। ধরা যাক, তোমার ঘোল নম্বর 25। যখন শিক্ষক তোমার নাম ধরে ডাক দেন, তখন তুমি দাঁড়াও। আবার তোমার ঘোল নম্বর ধরে ডাক দিলেও তুমি দাঁড়িয়ে যাও। তাহলে তুমি যদি কোনো ভ্যারিয়েবল হও, তোমার ঠিকানা হচ্ছে তোমার ঘোল নম্বর। বুঝতে পারলে?

প্রোগ্রামিংয়ে কী করলে কী হয়, সেটি বোঝার সহজ উপায় হচ্ছে নিজে প্রোগ্রাম লিখে রাখ করানো। তাই তোমরা এখন নিচের প্রোগ্রামটি লিখে কম্পাইল ও রাখ করাবে। আউটপুট কী হবে, কেন হবে, সেগুলো আমি আর বিস্তারিত বলে দিলাম না। নিজেরা চিন্তা করে বুঝে নাও।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int x = 10;
6     int *p;
7
8     printf("Value of x: %d\n", x);
9
10    p = &x;
11    *p = 20;
12
13    printf("Value of x: %d\n", x);
14
15    x = 15;
16
17    printf("Value of x: %d\n", x);
18    printf("Value stored at location %p is %d\n", p, *p);
19
20    printf("Address of x: %p\n", &x);
21    printf("Value of p: %p\n", p);
22
23    return 0;
24 }
```

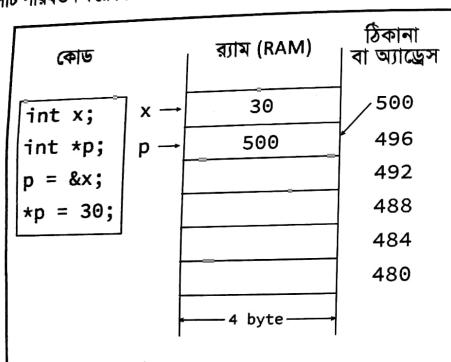
প্রোগ্রাম ২-৫

৩১

কম্পিউটার প্রোগ্রামিং - বিভিন্ন খণ্ড

এখন একটু ভাষার চর্চা করি। পয়েন্টার (pointer) মানে হচ্ছে যে পয়েন্ট (point) করে। পয়েন্ট করা মানে নির্দেশ করা (প্রোজেক্ট থেকে বাংলা অভিধান দেখে নিতে পারো)। আর আমদের সি প্রোগ্রামিং ভাষায় যে পয়েন্টার, সে একটি ভ্যারিয়েবলের ঠিকানা রাখারে এবং তাৰ আগে * টিক্কা দিয়ে সেই ভ্যারিয়েবলের মানও পাওয়া যাবে। p-তে যদি আমি x নামক ভ্যারিয়েবলের ঠিকানা রাখি, তবে x-এর মান x ব্যবহার করেও পাওয়া যাবে, আবার *p ব্যবহার করেও পাওয়া যাবে। *p ব্যবহার করে x-কে একসেস করার পক্ষতিকে বলে ডিরেফারেন্সিং (dereferencing)।

আমি যদি x-এর ঠিকানা p-তে রাখি, তাহলে সেহেতু *p ব্যবহার করে x-কে একসেস করা যায়, *p-তে অন্য কোনো মান আসাইন করলে x-এর মানও পরিবর্তিত হয়ে যাবে। কারণ আমরা তো *p দিয়ে বোঝাচ্ছি p নামক পয়েন্টারে যে ভ্যারিয়েবলের ঠিকানা রাখা আছে, সে ভ্যারিয়েবলের মান। তো সেটি পরিবর্তন করে দিলে তো x-এর মানও পরিবর্তিত হয়ে যাবে, নাকি?



ছবি ৩: পয়েন্টার ব্যবহার করে র্যামের ঠিকানায় তথ্য সেখা

এখন আমরা আরেকটি প্রোগ্রাম লিখব এবং সেটির আচরণ ব্যাখ্যা করব।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int x = 10;
6     int y;
7     int *p;

```

৩২

কম্পিউটার প্রোগ্রামিং - বিভিন্ন খণ্ড

```

8     printf("Value of x: %d\n", x);
9
10    p = &x;
11    y = *p;
12    *p = 15;
13
14    printf("Value of x: %d\n", x);
15    printf("Value of y: %d\n", y);
16    printf("Value of *p: %d\n", *p);
17    printf("Address of x: %p\n", &x);
18    printf("Address of y: %p\n", &y);
19    printf("Value of p: %p\n", p);
20
21
22
23 }

```

প্রোগ্রাম ২-৬

আউটপুট হবে এরকম:

```

Value of x: 10
Value of x: 15
Value of y: 10
Value of *p: 15
Address of x: 0x0060FF08
Address of y: 0x0060FF04
Value of p: 0x0060FF08

```

লক্ষ কর যে, x ও y দুটি পৃথক ভ্যারিয়েবল, তাই এদের ঠিকানাও আলাদা। কিন্তু p-এর মান হচ্ছে x-এর ঠিকানা। তাহলে x-এর যেই মান হবে, *p-এরও সেই একই মান হবে। আবার উল্টোটিও সত্য। তাই আমরা যখন *p = 15 লিখলাম, x এর মানও পরিবর্তিত হয়ে গেল। এদিকে y-এ আমরা রেখেছি *p আর তখন *p-তে আছে 10 (x-এর মান)। তাই পরবর্তী সময়ে *p-তে অন্য মান রাখলেও সেটি y-এর মানে কোনো প্রভাব ফেলবে না। কারণ y-তে 10 আসাইন করা হয়ে গিয়েছে। আর p হচ্ছে x-এর পয়েন্টার, y-এর নয়। ছবি ৪ এর সাহায্যে অ্যাড্রেসগুলো দেখানো হলো।

এখন তোমরা নিচের প্রোগ্রামটি লিখ এবং রান করে দেখো যে কী আউটপুট আসে।

```

1 #include <stdio.h>
2

```

৩৩

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

```

3 int main()
4 {
5     int x = 10, y;
6     int *p, *q;
7
8     p = &x;
9     y = *p;
10    *p = 15;
11    *q = 20;
12
13    printf("Value of x: %d\n", x);
14    printf("Value of y: %d\n", y);
15    printf("Value of *p: %d\n", *p);
16    printf("Value of *q: %d\n", *q);
17
18    return 0;
19 }

```

প্রোগ্রাম ২-৭

কোড	র্যাম (RAM)	ঠিকানা বা অ্যাড্রেস
int x = 10;	x → 15	500
int y;	y → 10	496
int *p;	p → 500	492
p = &x;		488
y = *p;		484
*p = 15;		480
	4 byte	

চিত্র ৪: প্রোগ্রাম ২-৬ এর কোডের ভ্যারিয়েবলের মেমোরি

খুব সন্তুষ্ট তোমরা আউটপুট দেখবে Segmentation fault: 11, যদি লিনাক্স মেশিন হয়। অথবা, যদি উইন্ডোজ মেশিন হয় তাহলে তোমাদের এই প্রোগ্রামটি সন্তুষ্ট ক্র্যাশ করবে। কম্পাইল ঠিকানাক্ষেত্রে করলেও প্রোগ্রাম রান করতে সমস্যা হয়েছে। যখন কোনো প্রোগ্রাম ভুলভাবে মেমোরির কোনো জায়গা একসেস করার চেষ্টা করে, সেটা অপারেটিং সিস্টেম ধরতে

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

পারলে এই এর দেয়। আমরা এখানে ভুল কী করলাম? $*q = 20$; লাইনে আমরা যে q পয়েন্টারকে ডিফারেন্সিং করে সেখানে 20 অ্যাসাইন করছি (মানে q মেই ঘরে পয়েন্ট করা আছে, সেখানে), q-কে তো আমরা কাউকে পয়েন্ট করতে বলিনি। তাই আমাদের যেটি করতে হবে, $q = &y$; লিখতে হবে যদি আমরা চাই যে q, y-কে পয়েন্ট করুক। সুতরাং কোনো পয়েন্টারের কম্পটেট একসেস করতে হলে অবশ্যই আগে একটি ভ্যারিয়েবলের অ্যাড্রেস মেই পয়েন্টারে অ্যাসাইন করতে হবে। আচ্ছা, আমি "খুব সন্তুষ্ট" লিখলাম কেন? কারণ অপারেটিং সিস্টেম যদি বিষয়টি ধরতে না পারে, তাহলে এর নাও দিতে পারে। এজন্যই পয়েন্টারের ব্যবহার ঝুঁকে-শুনে করতে হয়।

এখন তোমাদের জন্য প্রশ্ন, নিচের প্রোগ্রামটির আউটপুট কী হবে?

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int x = 10, y;
6     int *p, *q;
7
8     p = &x;
9     q = &y;
10    y = *p;
11    *p = 15;
12    *q = 20;
13
14    printf("Value of x: %d\n", x);
15    printf("Value of y: %d\n", y);
16    printf("Value of *p: %d\n", *p);
17    printf("Value of *q: %d\n", *q);
18
19    return 0;
20 }

```

প্রোগ্রাম ২-৮

২.১ – নাল পয়েন্টার

সি প্রোগ্রামিং ভাষায়, যদি আমরা একটি পয়েন্টার ডিক্লেয়ার করি, যেটি এই মুহূর্তে কোনো ভ্যারিয়েবলকে পয়েন্ট করছে না, তাহলে আমরা সেই পয়েন্টারে নাল (NULL) অ্যাসাইন করে দিতে পারি: `int *ptr = NULL;`। এখন নিচের প্রোগ্রামটি চালাও।

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int x = 100;
6     int *p = NULL;
7
8     printf("Value of x: %d\n", x);
9     printf("Value of *p: %d\n", *p);
10
11    return 0;
12 }

```

প্রোগ্রাম ২-৯

এখানেও দেখবে Segmentation fault: 11 অথবা উইন্ডোজে প্রোগ্রাম ক্র্যাশ। কারণ আমরা প্রথমে বললাম যে, p কোনো ভ্যারিয়েবলকে পয়েন্ট করছে না, তারপরে আবার p-কে ডিরেফারেন্স করার চেষ্টা করলাম (*p-এর মান প্রিণ্ট করার চেষ্টা করতে গিয়ে)। তাই এই এর ডিরেফারেন্স করার চেষ্টা করলাম (*p-এর মান প্রিণ্ট করার চেষ্টা করতে গিয়ে)। তাই এই এর।

• নোট:

আমরা মধ্যে * চিহ্ন ব্যবহার করে কোনো পয়েন্টারে যেই মেমোরি অ্যাড্রেস লেখা আছে, সেই মেমোরি লোকেশনের মান অ্যাকসেস করতে চাই তখন তাকে বলে ডিরেফারেন্সিং (Dereferencing)। এখানে * কে বলা হয় ডিরেফারেন্সিং অপারেটর (Dereferencing Operator)।

প্রোগ্রামটি নিচের মতো করে লিখলে আর এরর হবে না।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int x = 100;
6     int *p = NULL;
7
8     printf("Value of x: %d\n", x);
9     p = &x;
10    printf("Value of *p: %d\n", *p);
11
12 }

```

৩৬

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

```

12     return 0;
13 }

```

প্রোগ্রাম ২-১০

এখন নিচের প্রোগ্রামটি টাইপ করে রান কর।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int *p = NULL;
6     *p = 100;
7     printf("Value of *p : %d\n", *p);
8
9     return 0;
10 }

```

প্রোগ্রাম ২-১১

এখানেও দেখবে Segmentation fault: 11 অথবা প্রোগ্রাম ক্র্যাশ। কারণ আগের প্রোগ্রামে তো আমরা বলে দিয়েছিলাম যে [p = &x]; কিন্তু এই প্রোগ্রামে আমরা *p-তে 100 রাখার চেষ্টা করেছি, কিন্তু p তো কাউকে পয়েন্ট করছে না, এটি যে নাল পয়েন্টার!

২.২ - স্ট্রিং ও পয়েন্টার

তোমরা যারা আমার কম্পিউটার প্রোগ্রাম ১ম খণ্ড বইটি পড়েছ, তারা ইতিমধ্যেই স্ট্রিংয়ের সঙ্গে পরিচিত। পয়েন্টার ব্যবহার করেও কিন্তু আমরা স্ট্রিংয়ের কাজকর্ম করতে পারি। সেগুলোর কিছু উদাহরণ আমরা দেখব।

প্রথমেই একটি সহজ প্রোগ্রাম লেখা যাক।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     char s[] = "Bangladesh";
6
7     printf("Name of our country : %s\n", s);
8     printf("Address of s: %p\n", s);

```

৩৭

কম্পিউটার প্রোগ্রামিং – বিত্তীয় খণ্ড

```

9     return 0;
10
11 }
```

প্রোগ্রাম ২-১২

এখনে একটি স্ট্রিং ডিক্লেয়ার করেছি এবং সেটি প্রিন্ট করেছি। আর স্ট্রিং হচ্ছে ক্যারেক্টারের আবে আর আবের নামটিই তার আয়াড্রেস বা ঠিকানা নির্দেশ করে, তাই সেটিও প্রিন্ট করেছি। আমার কম্পিউটারের আউটপুট এরকম:

Name of our country : Bangladesh
Address of s: 0x0060FF05

তোমার কম্পিউটার অ্যাড্রেসের জন্য তিনি মানও দেখাতে পারে। তাহলে আমরা তো স্ট্রিংের বদলে একটি ক্যারেক্টারের পয়েন্টার ব্যবহার করেও কাজ করতে পারি। পয়েন্টারের ভেতরে স্ট্রিংের ঠিকানা রেখে দিবেই হবে।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     char s[] = "Bangladesh";
6     char *p;
7
8     p = s;
9
10    printf("Name of our country : %s\n", p);
11
12    return 0;
13 }
```

প্রোগ্রাম ২-১৩

রান করলে আউটপুট আসবে ঠিকঠাক।

তাহলে কী বুবলে? ক্যারেক্টার টাইপের পয়েন্টার দিয়েও কিন্তু স্ট্রিংের কাজ করা যায়। আমরা চাইলে এভাবেও লিখতে পারতাম : `char *p = "Bangladesh";`। এখন তোমার মনে ক্যারেক্টার টাইপের পয়েন্টার দিয়ে যদি স্ট্রিংের কাজ করা যায়, তাহলে শুধু তোমার প্রশ্নের উত্তর দেবে।

৩৮

কম্পিউটার প্রোগ্রামিং – বিত্তীয় খণ্ড

```

1 #include <stdio.h>
2
3 int main()
4 {
5     char c1 = 'A', c2 = 'B', c3 = 'C';
6     char *p1, *p2, *p3;
7     p3 = &c3;
8     p2 = &c2;
9     p1 = &c1;
10
11    printf("%c, %c, %c\n", *p1, *p2, *p3);
12
13    return 0;
14 }
```

প্রোগ্রাম ২-১৪

এটি টাইপ করে কম্পাইল ও রান কর।

২.৩ – পয়েন্টারের পয়েন্টার

পয়েন্টার তো নিজে একটি ভারিয়েবল। তারও নিচয়ই একটি ঠিকানা আছে নাকি? চলো, এর জন্য আমরা আবার কম্পিউটারের শরণাপন্ন হই এবং একটি প্রোগ্রাম লিখে দেখি।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     char c = 'A';
6     char *p;
7
8     p = &c;
9
10    printf("Address of c : %p\n", p);
11    printf("Address of p : %p\n", &p);
12
13    return 0;
14 }
```

প্রোগ্রাম ২-১৫

৩৯

রান করলে আউটপুট আসে এরকম:

Address of c : 0x0060FF0F
Address of p : 0x0060FF08

দুটি প্রথম মেমোরি অ্যাড্রেস। তাহলে পয়েন্টারকেও নিচয়ই পয়েন্টারের তেতর রাখা যাব।
আমরা আমাদের আগের প্রোগ্রামটি একটু পরিবর্তন করি:

```
1 #include <stdio.h>
2
3 int main()
4 {
5     char c = 'A';
6     char *p, **q;
7
8     p = &c;
9     q = &p;
10
11    printf("Value of c : %c\n", c);
12    printf("Value of c : %c\n", *p);
13    printf("Value of c : %c\n", **q);
14
15    return 0;
16 }
```

প্রোগ্রাম ২-১৬

আউটপুট আসবে:

```
Value of c : A
Value of c : A
Value of c : A
```

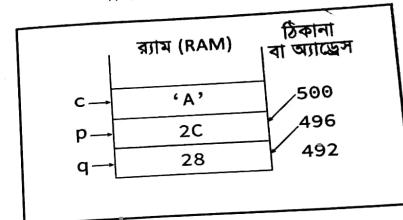
আশা করি, বুঝতে পেরেছ যে কীভাবে আমরা পয়েন্টারের পয়েন্টার ব্যবহার করলাম। বুঝতে না
পারলে কোডটি ভালভাবে দেখো আর ছবিটি ৫ ও দেখতে পারো।

প্রোগ্রামিং হচ্ছে চৰ্চাৰ বিষয়, তাই আমরা এখন আরেকটি প্রোগ্রাম লিখব (প্রায় আগের
প্রোগ্রামটিৰ মতোই, কেবল এক লাইনের পরিবর্তন)। আর তোমরা কিন্তু তোমাদের আগেৰ
ফাইলটি পরিবর্তন কৰে কাজ সেৱে ফেলবে না, প্রতিটি প্রোগ্রাম নতুনভাবে টাইপ কৰবে, তাহলে
শেখা সহজ হয়।

৮০

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড



ছবি ৫: পয়েন্টার ও পয়েন্টারের পয়েন্টার

```
1 #include <stdio.h>
2
3 int main()
4 {
5     char c = 'A';
6     char *p, **q;
7
8     p = &c;
9     q = &p;
10
11    **q = 'B';
12
13    printf("Value of c : %c\n", c);
14    printf("Value of c : %c\n", *p);
15    printf("Value of c : %c\n", **q);
16
17    return 0;
18 }
```

প্রোগ্রাম ২-১৭

প্রোগ্রামটির আউটপুট কী হবে সেটি কোড রান না কৰে বলতে পাৰবে? যদি বলতে পাৰো, তাহলে
পয়েন্টার সম্পর্কিত মৌলিক বিষয়গুলো তোমাৰ দখলে চলে এসেছে। আৱ না পারলে বইটি শুৰু
থেকে এই পৰ্যন্ত আবাৰ পড় এবং আমি যা যা কৰতে বলেছি, সেগুলো ঠিকভাবে অনুসৰণ কৰ।

৮১

অধ্যায় ৩ - ফাইল (File)

আমরা যারা কম্পিউটার ব্যবহার করি, সবাই বিভিন্ন ধরনের ফাইলের সঙ্গে পরিচিত। ফাইলে বিভিন্ন প্রকারের তথ্য সংরক্ষিত থাকে। যেমন ডুকমেন্ট, গান, ভিডিও, সফটওয়্যার ইত্যাদি। এমনকি তোমরা যে কোড লিখ, সেগুলোও তো ফাইলে সংরক্ষণ কর। কম্পিউটারে অনেক প্রকারের ফাইল রয়েছে আর সেগুলো তৈরি করা কিংবা সেগুলোতে কাজ করার জন্য আছে বিভিন্ন রকম সফটওয়্যার। তোমরা যদি খেয়াল করে থাকে, একটি ফাইলের নামের তিনটি অংশ থাকে। ফাইলের নাম, একটি ডট (.) ও তারপরে কর্যকর্তি অক্ষর। ডটের পরে যেই অক্ষরগুলো আছে, তাকে বলে ফাইলের এক্সটেনশন (extension)। যেমন: hello.c নামক ফাইলে hello হচ্ছে ফাইলের নাম, তারপরে ডট আর তারপরে হচ্ছে এক্সটেনশন (c)। এই এক্সটেনশন দিয়ে বোঝা যায় যে, এটি একটি সি প্রোগ্রাম। আবার index.html বা index.htm নামক ফাইলের এক্সটেনশন দেখে বোঝা যায় যে, এটি একটি ইচটিএমএল (HTML) ফাইল। ফাইলের এক্সটেনশন ছাড়াও নাম উপায়ে কম্পিউটার বুঝতে পারে যে, একটি ফাইল কী ধরনের এবং কোন সফটওয়্যার দিয়ে সেটি চালানো বা খোলা যাবে। আর সেই সফটওয়্যারটিও বুঝতে পারে যে ফাইলটি তার জন্য উপযুক্ত কি না। বেশিরভাগ ক্ষেত্রেই ফাইলের এক্সটেনশনই যথেষ্ট নয়, আরও কিছু পদ্ধতি আছে, যেগুলো ঠিক এই বইতে আলোচনা করা সাজে না।

ধরো, আমি এখন একটি প্রোগ্রাম লিখতে চাই। আমি তাহলে কী করব? কোডরক্স চালু করে হয় একটি নতুন ফাইল তৈরি করব কিংবা একটি পুরোনো ফাইল খুলব (ওপেন করব)। তারপরে আমি সেখানে কোড লিখব কিন্তু যতক্ষণ পর্যন্ত না আমি ফাইলটি সেভ করছি, ততক্ষণ পর্যন্ত এটি হার্ডডিক্সে সংরক্ষিত হবে না। আবার পুরোনো ফাইল খোলার বেলায় ফাইলে যেসব কোড ছিল, সেগুলো বিস্তৃ ফাইল খোলার সঙ্গে সঙ্গে কোডরকসের এডিটরে দেখতে পাব, কারণ কোডরকস সেই ফাইলটি ওপেন করার পরে তার সব জিনিস পড়ে (যাকে আমরা রিড করা বলি) ক্রিনে প্রদর্শন করছে। আর কাজ শেষে আমরা নির্দিষ্ট ফাইলটি কিংবা কোডরকস সফটওয়্যারটি বন্ধ করে দিই - উভয়ক্ষেত্রেই ফাইল বন্ধ হয়ে যায় (একে বলে ফাইল ক্লোজ করা)।

আগের অংশটুকু খেয়াল করলে দেখবে যে আমি কিছু ইংরেজি শব্দ বাংলায় লিখেছি, সেগুলো হচ্ছে -

- ✓ ফাইল ওপেন (File Open),
- ✓ ফাইল রিড (File Read),
- ✓ ফাইল রাইট (File Write) এবং
- ✓ ফাইল ক্লোজ (File Close)।

82

আমরা যখন ফাইল নিয়ে কাজ করি, তখন মূলত এসব অপারেশনই করি, সেটি আমরা নিজে করি কিংবা সফটওয়্যার স্বয়ংক্রিয়ভাবে করে। আমরা এখন শিখব প্রোগ্রামিং করে কীভাবে এই কাজগুলো করা যায়।
প্রথমেই কাজ হবে নিচের প্রোগ্রামটি হ্রাস টাইপ করে কম্পাইল ও রান করা। যারা এটি করবে না, তাদের মে কেবল প্রোগ্রামিং শেখাটা অসম্পূর্ণ হবে যাবে তা ই নয়, তারা প্রোগ্রামিংয়ের আনন্দ খেকেও বাধিত হবে।

```
1 #include <stdio.h>
2
3 int main()
4 {
5     FILE *fp;
6     char filename[] = "my_file.txt";
7
8     fp = fopen(filename, "w");
9
10    fprintf(fp, "This is a file created by my program!");
11    fprintf(fp, "I am so happy.");
12
13    fclose(fp);
14
15    return 0;
16 }
```

প্রোগ্রাম ৩-১

প্রোগ্রামটি চালালে তুমি দেখতে পাবে যে তোমার কম্পিউটারে my_file.txt নামক একটি ফাইল তৈরি হয়েছে! তোমার সি প্রোগ্রাম যেখান আছে, সেই ফোল্ডারেই এই ফাইলটি তৈরি হওয়ার কথা, সেখানে খুঁজে না পেলে অন্য কোথাও আছে, তুমি খুঁজে নাও। তারপরে সেটি নেটপ্যাড কিংবা অন্য কোনো টেক্সট এডিটর দিয়ে খুলতে পার। তোমাদের কারও কম্পিউটারে যদি আবার এমন ঠিক করা থাকে যে, অপারেটিং সিস্টেম ফাইলের এক্সটেনশন লুকিয়ে রাখবে (ব্যবহারকারীকে দেখাবে না), তাহলে তোমরা শুধু my_file দেখবে। তোমরা এই সেটিংস পরিবর্তন করে নিতে পার যেন সবসময় ফাইলের এক্সটেনশন দেখা যায়, তাহলে অনেক বামেলা করে যাবে।

এখন প্রোগ্রামটি ব্যাখ্যা করা যাক। মেইন ফাংশনের ভেতরে প্রথম লাইনেই লিখেছি **FILE *fp;**। এর মানে হচ্ছে fp হচ্ছে একটি FILE টাইপের পয়েন্টার। তবে FILE কিন্তু ঠিক ইন্সিজার বা ডবলের মতো ডেটাটাইপ নয়, এটিকে বলে ফাইল হ্যান্ডেল (handle) যা একটি ফাইল ইনপুট-আউটপুটসহ নানাবিধি কাজ করার সুযোগ দেয়। সি প্রোগ্রামে ফাইল নিয়ে কাজ করতে

83

কম্পিউটার প্রোগ্রামিং - বিভাগ ষষ্ঠি

গোলে প্রথমেই ফাইল হ্যান্ডেলের পয়েন্টার ডিক্রেয়ার করা লাগবে। এই পয়েন্টার ছাড়া ফাইল একসেস করা কিংবা ফাইলের তথ্য পড়া, লেখা কিছুই করা যাবে না।
পরের লাইনে আমি একটি স্ট্রিং তৈরি করেছি, যেটি হচ্ছে আমি যেই ফাইলটি তৈরি করতে চাচ্ছি তার নাম।

তৃতীয় লাইনে আমি ফাইল ওপেন করলাম `fopen()` ফাংশন দিয়ে। এর ভেতরে দুটি প্যারামিটার দিতে হয়। প্রথমটি হচ্ছে ফাইলের নাম, আর দ্বিতীয়টি হচ্ছে মোড (mode)। এই মোড কী জিনিস? মোড দিয়ে আসলে বেরানো হয় যে ফাইলটি ওপেন করার পরে আমি সেটিতে কী করব? বা কী কাজের জন্য ফাইল ওপেন করতে চাচ্ছি? নিচে একটি টেবিল দিলাম:

মোড	বর্ণনা
w	ফাইল লেখার উদ্দেশ্যে একটি ফাইল খোলা। যদি ফাইলটি না থাকে, তাহলে সেটি তৈরি করে নেয়।
r	ফাইল পড়ার জন্য খোলা। অবশ্যই ফাইলটি কম্পিউটারে থাকতে হবে।
a	Append শব্দটি মনে রাখতে হবে। এর অর্থ হচ্ছে যোগ করা বা মুক্ত করা। এই মোডে ফাইল খোলার অর্থ হচ্ছে ফাইলটি খোলার আগে যা আছে, তা থাকবে আর সেগুলোর পর থেকে ফাইলে লেখা শুরু হবে। ফাইল না থাকলে তৈরি করে নিবে।
w+	লেখা ও পড়া। তবে এটি ব্যবহার করার সময় সতর্ক থাকতে হবে। ফাইলে কোনো কিছু থাকলে সেগুলো মুছে যেতে পারে।
r+	পড়া ও লেখা।
a+	লেখা ও পড়া। ফাইল পড়া শুরু হবে প্রথম থেকে তবে লেখার সময় শেষ থেকে লেখা শুরু, আগের কোনো কিছু মুছবে না।

আর `fopen()` ফাংশনটি একটি ফাইল পয়েন্টার রিটার্ন করে, তাই সেই ফাইল পয়েন্টারের মান আমি `fp`-তে অ্যাসাইন করে দিলাম: `fp = fopen(...);`

ফাংশন `fopen`

প্রোটোটাইপ:

```
FILE *fopen ( const char *restrict filename, const char
```

88

কম্পিউটার প্রোগ্রামিং - বিভাগ ষষ্ঠি

*restrict mode);

ইনপুট:

const char *restrict filename: একটি স্ট্রিং যা ফাইলের নাম নির্দেশ করে।
const char *restrict mode: একটি স্ট্রিং যা মোড নির্দেশ করে।

রিটার্ন ভ্যালু:

FILE *: ফাইল পয়েন্টার বা ফাইল হ্যান্ডেল রিটার্ন করে।

কাজ:

এই ফাংশনটি একটি ফাইল এর নাম `filename` ও মোড `mode` ইনপুট নেয় ও ফাইলটি খুলে একটি ফাইল হ্যান্ডেল রিটার্ন করে।

তারপর আমি `fprintf()` ফাংশন ব্যবহার করে একটি স্ট্রিং প্রিন্ট করলাম। তবে এই ফাংশনের প্রথম প্যারামিটারে কিন্তু ফাইল পয়েন্টারটি দিতে হবে, নইলে এটি বুঝবে না যে কোন ফাইলে লিখতে হবে।

সবশেষে ফাইলটি বন্ধ করতে হবে, আর সেটি করার জন্য `fclose()` ফাংশন ব্যবহার করতে হবে। প্যারামিটার হিসেবে অবশ্যই কোন ফাইলটি বন্ধ করতে চাচ্ছি, সেটির পয়েন্টার পাঠাতে হবে। তোমরা অনেক সময় দেখবে যে `fclose()` ফাংশন ব্যবহার না করলেও অসুবিধা হচ্ছে না, এর কারণ হচ্ছে অনেক সময় অপারেটিং সিস্টেম তোমার হয়ে এই কাজটি করে দেয়। কিন্তু যদি এই আশায় `fclose()` ফাংশন ব্যবহার করা বন্ধ করে দাও, তাহলে ভবিষ্যতে বিপদে পড়বে।

৫ ফাংশন `fprintf`

প্রোটোটাইপ:

```
int fprintf ( FILE * stream, const char * format, ... );
```

ইনপুট:

FILE * stream: একটি ফাইল হ্যান্ডেল

const char * format: একটি ফরম্যাটেড স্ট্রিং

... : ফরম্যাট স্ট্রিংয়ে অবস্থিত যতটি ফরম্যাট স্পেসিফিকেশন আছে, ততটি ইনপুট

রিটার্ন ভ্যালু:

int: ঠিক কর্তৃত ক্যারেক্টার লিখা হলো, সেই সংখ্যাটি রিটার্ন করে

85

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

কাজ:
এই কাশনটি stream ফাইল হ্যান্ডেলকে আউটপুট স্ট্রিম হিসেবে বিবেচনা করে, `format` স্থিতে নির্দেশিত ফরম্যাটে `printf` ফাংশনের মতো ফরম্যাট স্পেসিফিকেশনলোকে পরবর্তী ভ্যারিয়েবলের মান দিয়ে প্রতিস্থাপন করে ফাইলে ডেটা রাইট করে এবং কটটি ক্যারেক্টার রাইট করা হলো সেই সংখ্যা রিটার্ন করে।

ফাইল একবার বন্ধ করে দিলে কিন্তু আর সেটিতে লেখা বা পড়া যাবে না। তা করতে সেজে ফাইলটি আবার খুলতে হবে (`fopen()` ব্যবহার করে)। নিচের প্রোগ্রামটি চালাও :

```

1 #include <stdio.h>
2
3 int main()
4 {
5     FILE *fp;
6     char filename[] = "my_file.txt";
7     fp = fopen(filename, "w");
8
9     fprintf(fp, "This is a file created by my program! ");
10    fprintf(fp, " I am so happy.\n");
11    fclose(fp);
12    fprintf(fp, " Second line.\n");
13
14    return 0;
15 }
```

প্রোগ্রাম ৩-২

আউটপুট ফাইলে (এক্ষেত্রে `my_file.txt`) `Second line` লেখা হয় নি, কারণ তার আগেই ফাইলটি বন্ধ করে দিয়েছি। আমরা চাইলে ফাইলটি আবার খুলতে পারি। নিচের কোডে আমি কাজটি করে দেখালাম:

```

1 #include <stdio.h>
2
3 int main()
4 {
5     FILE *fp;
6     char filename[] = "my_file.txt";
7     fp = fopen(filename, "w");
8 }
```

৪৬

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

```

9     fprintf(fp, "This is a file created by my program! ");
10    fprintf(fp, " I am so happy.\n");
11    fclose(fp);
12
13    fp = fopen(filename, "a");
14    fprintf(fp, "Second line.\n");
15    fclose(fp);
16
17    return 0;
18 }
```

প্রোগ্রাম ৩-৩

এখন খেয়াল কর যে, দ্বিতীয়বার ফাইল খোলার সময় আমি অ্যাপেন্ড মোড ব্যবহার করেছি, `fp = fopen(filename, "a");`। এখন তোমাদের কাজ হবে অ্যাপেন্ড মোডের বদলে রাইট মোড দিয়ে প্রোগ্রামটি লেখা। এছাড়া অন্যান্য মোডগুলোও ব্যবহার করে দেখো, তাহলে তোমাদের পক্ষে বুঝতে সহজ হবে, আর প্রোগ্রামিং শেখার উপায়ই হচ্ছে হাতেকলমে শেখা।

৫ ফাংশন `fclose`

প্রোটোটাইপ:

```
int fclose ( FILE * stream );
```

ইনপুট:

`FILE * stream` : একটি ফাইল হ্যান্ডেল

রিটার্ন ভ্যালু:

`int` : ফাইলটি সফলভাবে বন্ধ হলে 0 রিটার্ন করে, নাহলে EOF রিটার্ন করে

কাজ:

`stream` এ নির্দেশিত ফাইলটি বন্ধ করে।

ইতিমধ্যে তোমরা ফাইলে কোনো কিছু লেখার জন্য `fprintf` ফাংশন ব্যবহার করে ফেলেছ। এখন ফাইল থেকে ইনপুট নিতে চাইলে কী করতে হবে? `scanf`-এর মতো `fscanf` ফাংশন ব্যবহার করা যায়। এখন আমরা একটি প্রোগ্রাম লিখব, যেটি একটি ফাইল থেকে দুটি সংখ্যা পড়বে আর আরেকটি ফাইলে তাদের যোগফল প্রিণ্ট করবে। তোমার সি প্রোগ্রামটি যেখানে লিখবে, সেই ফোল্ডারে `in.txt` নামে একটি ফাইল তৈরি কর আর সেখানে পরপর দুটি পৃথক

৪৭

কম্পিউটার প্রোগ্রামিং - বিস্তীর্ণ খণ্ড

লাইনে দুটি পূর্ণসংখ্যা লিখ। এবাবে নিচের প্রোগ্রামটি লিখে চালিয়ে দেখো। আর যারা উইন্ডোজ ব্যবহার কর, তারা ফাইলের নামের জায়গায় পুরো পথ (path) ব্যবহার করতে পার এভাবে
 C:\programs\practice\cpbook2\in.txt (যদি তোমার প্রোগ্রামটি সি প্রোগ্রাম কোডের ভেতরে programs ক্ষেত্রের ভেতরে practice ক্ষেত্রের ভেতরে cpbook2 ক্ষেত্রের থাকে।)

```

1 #include <stdio.h>
2
3 int main()
4 {
5     FILE *fp_in, *fp_out;
6     char *input_file = "in.txt";
7     char *output_file = "out.txt";
8     int num1, num2, sum;
9
10    fp_in = fopen(input_file, "r");
11    fp_out = fopen(output_file, "w");
12
13    fscanf(fp_in, "%d", &num1);
14    fscanf(fp_in, "%d", &num2);
15    sum = num1 + num2;
16    printf("%d %d %d\n", num1, num2, sum);
17    fprintf(fp_out, "%d\n", sum);
18
19    fclose(fp_in);
20    fclose(fp_out);
21
22    return 0;
23 }
```

প্রোগ্রাম ৩-৪

তোমরা দেখবে যে out.txt নামের একটি ফাইল তৈরি হয়েছে এবং সেখানে যোগফল রাখা আছে। একটি ব্যাপার লক্ষ কর, আমাদেরকে এখানে fscanf ফাংশন পরপর দুবার ব্যবহার করতে হচ্ছে। এর কারণ হচ্ছে fscanf ফাংশনটি কেবল একটি স্ট্রিং ইনপুট নিতে পারে আর সেখানেও কোনো স্পেস বা নিউলাইন থাকা চলবে না। কোনো হোয়াইটস্পেস (whitespace) ক্ষারেষ্টার পেলেই সে ইনপুট নেওয়া বন্ধ করে দেয়।

কম্পিউটার প্রোগ্রামিং - বিস্তীর্ণ খণ্ড

৩ ফাংশন fscanf

প্রোটোটাইপ:

```
int fscanf ( FILE * stream, const char * format, ... );
```

ইনপুট:

FILE * stream : একটি ফাইল হ্যান্ডেল
 const char * format : একটি ফরম্যাটেড স্ট্রিং
 ... : ফরম্যাট স্ট্রিংয়ে যতটি ফরম্যাট স্পেসিফিকায়ার আছে, ততটি ভ্যারিয়েবলের ঠিকানা

রিটার্ন ভ্যালু:

int : কতটি ইনপুট পড়া হলো, সেই সংখ্যা রিটার্ন করে

কাজ:

scanf ফাংশনের মতোই stream নামক ফাইলকে ইনপুট স্ট্রিম হিসেবে বিবেচনা করে ইনপুট নেয়, format স্ট্রিংয়ে দেওয়া ফরম্যাটে। নির্ধারিত ফরম্যাট স্পেসিফিকায়ার অনুযায়ী ইনপুট নিয়ে পরবর্তী ভ্যারিয়েবলের ঠিকানায় সংরক্ষণ করা হয়। মোট কতটি ভ্যারিয়েবল ইনপুট নেওয়া হলো সেই সংখ্যাটি রিটার্ন করা হয়।

এখন আমরা যদি ইনপুট ফাইলে সংখ্যা দুটি পাশাপাশি রাখতে চাইতাম (এক বা একাধিক স্পেস দিয়ে আলাদা কর), তাহলে আমাদের কী করতে হবে? আমাদের পুরো লাইনটি একটি স্ট্রিং আকারে পড়তে হবে আর তারপরে সেই স্ট্রিং থেকে আবার ইন্টিজার দুটি পড়তে হবে। পুরো লাইন পড়ার জন্য fgets আর একটি স্ট্রিং থেকে দুটি ইন্টিজার পড়ার জন্য sscanf ফাংশন ব্যবহার করতে হবে। নিচের কোড যদি ভালোভাবে খেয়াল কর, তাহলে বিষয়টি পরিষ্কার হবে। আর হ্যাঁ, কেবল ভালোভাবে খেয়াল করলেই হবে না, প্রোগ্রাম টাইপ করে কম্পাইল ও রান করতে হবে।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     FILE *fp_in, *fp_out;
6     char *input_file = "in.txt";
7     char *output_file = "out.txt";
```

```

8   char line[80];
9   int num1, num2, sum;
10
11   fp_in = fopen(input_file, "r");
12   fp_out = fopen(output_file, "w");
13
14   fgets(line, 80, fp_in);
15   printf("Line: %s\n", line);
16
17   sscanf(line, "%d %d", &num1, &num2);
18
19   sum = num1 + num2;
20   printf("%d %d\n", num1, num2, sum);
21   fprintf(fp_out, "%d\n", sum);
22
23   fclose(fp_in);
24   fclose(fp_out);
25
26
27 }

```

প্রোগ্রাম ৩-৫

`fgets()` ফাংশনের প্রথম প্যারামিটারে, যেই স্ট্রিংয়ে তুমি ইনপুট রাখবে, তার নাম দিতে হয়। দ্বিতীয় প্যারামিটারে দিতে হয় স্ট্রিংয়ের সর্বোচ্চ দৈর্ঘ্য, এক্ষেত্রে আমি 80 দিয়েছি। তাহলে সে পায়, তাহলে সে ইনপুট নেওয়া বক্স করবে। আর নিউলাইন পেলে কিন্তু নিউলাইনটিও সে স্ট্রিংয়ের ভেতরে দিয়ে দিবে। তৃতীয় প্যারামিটার হচ্ছে, যেই ফাইল থেকে পড়তে হবে, সেই ফাইলের হাল্ডেলের পমেন্টার, আমাদের ক্ষেত্রে `fp_in`।

এই ফাংশন `fgets`

প্রোটোটাইপ:

`char * fgets (char * str, int num, FILE * stream)`

ইনপুট:

`char * str` : ইনপুট স্ট্রিং থেকে ডেটা পড়ে যেই স্ট্রিংয়ে রাখা হবে তার পমেন্টার
`int num` : সর্বমোট কতটি ক্যারেক্টার পড়া হবে
`FILE * stream` : যেই ফাইল থেকে ডেটা পড়া হবে তার হাল্ডেল

৫০

রিটার্ন ভ্যালু:

`char *` : ঠিকঠাক মতো পড়া শেষ হলে যেই স্ট্রিংয়ে ডেটা রাখা হলো তার পমেন্টার রিটার্ন করে। এছাড়া কোনো কারণে ডেটা পড়া না গেলে নাল পমেন্টার রিটার্ন করে।

কাজ:

`stream` ফাইলকে ইনপুট স্ট্রিম হিসেবে বিবেচনা করে `num-1` সংখ্যক ক্যারেক্টার পড়বে। `num-1` সংখ্যক ক্যারেক্টার পড়ার আগেই EOF বা নিউলাইন পেলে পেলে পড়া বন্ধ করবে ও `str` রিটার্ন করবে। ফাইল পড়তে কোনো error হলে নাল পমেন্টার রিটার্ন করবে।

আর `sscanf()` ফাংশনের প্রথম প্যারামিটার হচ্ছে যেই স্ট্রিং থেকে পড়তে হবে, তার নাম। দ্বিতীয় প্যারামিটার হচ্ছে ফরম্যাট স্ট্রিং, আমাদের ক্ষেত্রে `"%d %d"`, যেহেতু আমরা দুটি ইন্টজার ইনপুট নিতে চাচ্ছি। তারপরে কতটি প্যারামিটার লিখতে হবে, সেটি নির্ভর করবে ফরম্যাট স্ট্রিংয়ে আমরা কতটি উপাদানের কথা বলেছি, তার ওপর। আমরা দুটি ইন্টজার দুটি ভ্যারিয়েবলে রাখার জন্য `&num1, &num2` ফাংশনের প্যারামিটারে ব্যবহার করেছি (মানে ফাংশনের প্যারামিটারে পাঠিয়েছি)।

এই ফাংশন `sscanf`

প্রোটোটাইপ:

`int sscanf (const char * s, const char * format, ...)`

ইনপুট:

`const char * s` : উৎস স্ট্রিং`const char * format` : একটি ফরম্যাটেড স্ট্রিং`... :` ফরম্যাট স্ট্রিংয়ে যতটি ফরম্যাট স্পেসিফিকার আছে, ততটি ভ্যারিয়েবলের ঠিকানা

রিটার্ন ভ্যালু:

`int` : কতটি ভ্যারিয়েবল পড়া হলো সেই সংখ্যা রিটার্ন করে। কোনো ত্রুটি হলে EOF রিটার্ন করে।

কাজ:

৫১

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

S ট্রিচ থেকে scanf ফাংশনের মতো format স্ট্রিং অন্যান্য ডেটা পড়ে পরবর্তী
অ্যাক্টিভেটেড স্টাকসাঞ্চলোতে সংরক্ষণ করে। কট্টি ভারিয়েবল পড়া হলো সেই সংরক্ষণ
রিটার্ন করে। কোনো ডেটা পড়া না দিলে EOF রিটার্ন করে।

প্রোগ্রামটি ঠিক্কাক কাজ করবে। তবে ক্ষিণে যখন আউটপুট দিবে, তখন দেখবে দুটি লাইনের
মাঝে একটি ফাঁক লাইন আছে। কারণ কী? এটি তোমাদের জন্য প্রশ্ন, খুঁজে বের কর কর।

ফাইলের প্রাথমিক কাজগুলো যোটায়ুটি শিখে ফেলেছ। এখন তুমি চাইলে সি প্রোগ্রামিং
ল্যাঙ্গুয়েজের কোনো একটি বই থেকে বা ইন্টারনেটে খোঁজ করে ফাইল সম্পর্কিত আরও অনেক
ফাংশনের ব্যবহার শিখে নিতে পারো। তবে আমি যতটুকু দেখিয়েছি, ততটুকু দিয়ে তোমাদের
জন্য একটি ছেট কাজ দিই। একটি টেক্সট ফাইল তৈরি করও, যেখানে রোল নম্বর (একটি
ইংজিনের), তারপরে একটি স্পেস, তারপরে একটি পূর্ণসংখ্যা (0 থেকে 100-এর তেতরে), যা ওই
রোল নম্বরের প্রাণ একটি পরীক্ষার নম্ব। একরম দশজন শিক্ষার্থীর জন্য দশটি লাইন
তৈরি কর। ধরা যাক, এটি গণিত পরীক্ষার ফল, তাই ফাইলটি math.txt নামে সেভ কর। এবাবে
ওই দশজন শিক্ষার্থীর বাংলা ও ইংরেজি পরীক্ষার ফলের জন্যও অনুরূপ ফাইল তৈরি কর এবং
সেগুলোকে যথাক্রমে bangla.txt ও english.txt নাম দিয়ে সেভ কর। এখন তুমি একটি
প্রোগ্রাম তৈরি করবে, যেটি ওই তিনটি ফাইল থেকে পরীক্ষার রেজাল্ট পড়ে প্রতিটি শিক্ষার্থীর গড়
নম্বর বের করে আরেকটি ফাইলে সেভ করবে। আর এই ফাইলেও প্রথমে রোল নম্বর, তারপরে
একটি স্পেস, তারপরে গড় নম্বর থাকবে। তোমাদের সবারই এই প্রোগ্রামটি লিখতে পারা উচিত।
কেউ ৩০ মিনিটে করবে, কারণ তিনি ঘন্টা লাগবে, কারণ ও বা তিনি দিন। কত সময় লাগল, সেটি
এখন কোনো ব্যাপার নয়, নিজে নিজে প্রোগ্রামটি লিখতে পারাটাই গুরুত্বপূর্ণ।

এতক্ষণ তোমার কেবল টেক্সট ফাইল নিয়ে কাজ করেছ। এখন তোমাদের দেখাব বাইনারি
ফাইলের কাজ। ধরা যাক, তুমি একটি প্রোগ্রাম লিখবে যেটি একটি ইমেইজ ফাইলের কপি তৈরি
করবে। এটি কিন্তু টেক্সট ফাইল নয়। বাইনারি ফাইল নিয়ে কাজ করার সময় ফাইল খোলার সময়
আমরা যে মোড ব্যবহার করি, সেটিতে একটু পরিবর্তন করতে হবে। r, w, a এর বদলে লিখতে
হবে rb, wb, ab।

এখন তোমার কাজ হচ্ছে একটি ইমেজ ফাইল (ছবি) খুঁজে বের করে সেটি তুমি যেই ফোন্ডারে সি
প্রোগ্রামগুলো লিখছ, সেখানে নিয়ে আসা (কপি-পেস্ট করে)। এরপর নিচের প্রোগ্রামটি টাইপ
করে কম্পাইল ও রান করবে। তবে টাইপ করার সময় ফাইলের নাম বদলে দিতে হবে। আমি
image1.jpg লিখেছি কারণ আমার কম্পিউটারে আমি এই ফাইলটি নিয়ে কাজ করব, তোমার
স্ক্রেনে ফাইলের নাম অন্যকিছুও হতে পারে, যেমন photo.png ইত্যাদি। তুমি সেই অন্যান্য
প্রোগ্রামে ফাইলের নাম দিবে।

৫২

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     FILE *fp_in, *fp_out;
7     char *input_file = "image1.jpg";
8     char *output_file = "image2.jpg";
9     int ch;
10
11    fp_in = fopen(input_file, "rb");
12    if (fp_in == NULL) {
13        perror("File opening failed");
14        return EXIT_FAILURE;
15    }
16
17    fp_out = fopen(output_file, "wb");
18
19    while (1) {
20        ch = fgetc(fp_in);
21        if (ch == EOF) {
22            break;
23        }
24        fputc(ch, fp_out);
25    }
26
27    fclose(fp_in);
28    fclose(fp_out);
29
30    return 0;
31 }
```

প্রোগ্রাম ৩-৬

এখনে আমি দুটি ফাইলই বাইনারি মোডে ওপেন করেছি (একটি রিড বাইনারি, আরেকটি রাইট
বাইনারি)। তবে প্রথম ফাইলটি, যেটি ইমপুট ফাইল, সেটি ওপেন করার সময় একটি পরীক্ষা
করেছি। যদি fopen ফাংশন NULL পয়েন্টার রিটার্ন করে, তার অর্থ হচ্ছে ফাইলটি ওপেন করার
সময় কোনো ঝামেলা হয়েছে। তাই fp_in NULL কি না, সেটি পরীক্ষা করেছি এবং শর্ত যদি
সত্য হয়, তাহলে perror("File opening failed"); স্টেমেন্ট দিয়ে এরয়াটি প্রিন্ট
করেছি এবং মেইন ফাংশন থেকে রিটার্ন করেছি। আর রিটার্ন করার সময় 0 রিটার্ন না করে
EXIT_FAILURE রিটার্ন করেছি।

৫৩

৫৫ ফাংশন perror

প্রোটোটাইপ:
void perror (const char * str)

ইনপুট:
const char * str : একটি স্ট্রিং যা এর মেসেজের আগে প্রিন্ট হবে

রিটার্ন ভ্যালু:
void : কোনো মান রিটার্ন করে না

কাজ:
একটি অর্থবোধক এর মেসেজ প্রিন্ট করা।

তোমরা চাইলে তোমার ইনপুট ফাইলের নাম পরিবর্তন করে প্রোগ্রাম চালিয়ে পরীক্ষা করে দেখতে পারো যে ক্ষেত্রে কী এর প্রিন্ট হচ্ছে। আমার ক্ষেত্রে প্রিন্ট হচ্ছে:

File opening failed: No such file or directory

EXIT_FAILURE ব্যবহার করার কারণে আমাদেরকে stdlib.h হেডার ফাইলটি শুরুতে ইনক্লুড করতে হচ্ছে। নইলে তোমার কম্পাইলার এটিকে চিনবে না এবং প্রোগ্রাম কম্পাইল হবে না। ফাইল ওপেন করার সময় এরকম পরীক্ষা করা বেশ ভালো অভ্যাস, নানান বামেলা থেকে বাঁচা যায়। যেমন, ভুল এক্সটেনশন থাকা, এক্সটেনশন না থাকা, কোন্তোরের ভেতর ফাইল না থাকা, কোন্তোরাই না থাকা, ফাইলটি হ্যাতো অন্য কোনো প্রোগ্রাম খুলে রেখেছে আর সে কারণে খুলতে না পারা ইত্যাদি। তোমরা আগের সবগুলো প্রোগ্রামে এই কাজটি করতে পারো। আমি শুরুতে জটিলতা এড়ানোর জন্য কাজটি করিনি।

১ প্রশ্ন:

EXIT_FAILURE কী?

২ উত্তর:

সি প্রোগ্রামিংয়ে stdlib.h হেডার ফাইল দুটি ম্যাক্রো ডিফাইন করা আছে
এভাবে:

```
#define EXIT_SUCCESS /*implementation defined*/
#define EXIT_FAILURE /*implementation defined*/
```

মেইন ফাংশন থেকে বের হয়ে যাওয়ার সময় এই দুটি ম্যাক্রো রিটার্ন করা যায়। প্রোগ্রামটি সফলভাবে চলেছে কি না তা বোবানোর জন্য। যাতাবিকভাবেই EXIT_SUCCESS দিয়ে সফল এক্সিটিশন ও EXIT_FAILURE দিয়ে অসাফল এক্সিটিশন বোবায়। এদেরকে exit ফাংশনের অর্ডারে হিসেবেও ব্যবহার করা যায়।
বাইয়ের শেষ অধ্যায়ে ম্যাক্রো নিয়ে আলোচনা করা হচ্ছে।

৫৬ ফাংশন fgetc

প্রোটোটাইপ:
int fgetc (FILE * stream)

ইনপুট:

FILE * stream : যেই ফাইল থেকে ডেটা পড়া হবে তার হ্যান্ডেল

রিটার্ন ভ্যালু:

int : একটি ক্যারেক্টোরকে ইন্টিজারে টাইপকাস্ট করে রিটার্ন করে, অথবা EOF রিটার্ন করে।

কাজ:

stream থেকে পরবর্তী ক্যারেক্টোরটি unsigned char হিসেবে পড়ে ও ইন্টিজারে টাইপকাস্ট করে রিটার্ন করে। সেই সঙ্গে ফাইল পজিশন ইন্ডিকেটরকে এককর সামনে এগিয়ে দেয়।

প্রতিবারই ইনপুট নেওয়ার পরে পরীক্ষা করে দেখছি যে, ch-এর মান EOF-এর সমান কি না। যদি সমান হয়, তাহলে বুঝতে হবে যে, আমরা ফাইলের শেষে পৌছে গিয়েছি। আর তখন আমরা লুপ থেকে বেক করব, নইলে লুপটি চলতেই থাকবে, কারণ লুপের ভেতর শর্ত সবসময় সত্য - while(1)।

কম্পিউটার প্রোগ্রামিং - বিতীয় খণ্ড

● প্রশ্ন:

EOF কী?

● উত্তর:

EOF কথাটির মানে হচ্ছে End Of File। আমরা যখন কোনো উৎস বা সোর্স থেকে ডেটা ইনপুট নেই, প্রোগ্রামিং ভাষাতে সেই উৎস বা সোর্সকে বলা হয় স্ট্রিম (stream) বা ফাইল (file)। যখন কোনো ডেটা সোর্স থেকে আর কোনো ডেটা ইনপুট নেওয়া যায় না (এটি অনেক কাবণে হতে পারে, যেমন আর কোনো ডেটা নেই, ডিভাইস অথবা নেটওর্ক সংযোগ বিচ্ছিন্ন হওয়া ইত্যাদি), তখন সেই অবস্থা বোঝানোর জন্য একটি সিগন্যাল তৈরি করা হয়। সেটিকেই EOF বা End of File বলে।

আমরা যখন টার্মিনালে প্রোগ্রাম চালাই তখন ইনপুট কখনো আসলে "শেষ" হয় না যদি না টার্মিনালটিই বন্ধ হয়ে যায়। কিন্তু, ইনপুট শেষ হয়েছে, এমনটি বোঝানোর প্রয়োজন পড়তে পারে। তাই বিভিন্ন অপারেটিং সিস্টেমে একটি বিশেষ কিসিকোয়েল্স রিজার্ভ রাখা হয় EOF বোঝানোর জন্য। যেমন, ইউনিক্স (Unix) অপারেটিং সিস্টেমে Ctrl+D এই দুই কি একসঙ্গে চাপলে টার্মিনালের ড্রাইভার একটি EOF সিগন্যাল তৈরি করে। আবার উইন্ডোজ অপারেটিং সিস্টেমে Ctrl+Z দিয়ে এই বিষয়টি বোঝানো হয়। সি প্রোগ্রামিংয়ের ক্ষেত্রে stdio.h হেডের ফাইলে EOF নামে একটি ধ্রুবক ডিফাইন করা আছে যার মান -1। সি ভাষায় আমরা ইনপুট নেওয়ার জন্য scanf(), getc(), getchar(), fscanf() ইত্যাদি ফাংশন ব্যবহার করি। যখন এই ফাংশনগুলো ইনপুট নিতে নিতে ফাইলের শেষে চলে যায় বা টার্মিনাল থেকে EOF সিগন্যাল দেওয়া হয় (Ctrl+Z বা Ctrl+D ব্যবহার করে), তখন ফাংশনগুলো এই EOF মানটি রিটার্ন করে। যার অর্থ যেই উৎস (স্ট্রিম বা ফাইল) থেকে সে ডেটা ইনপুট নিচ্ছে, সেখানে আর কোনো ডেটা নেই।

যেয়াল রাখতে হবে যে, EOF নিজে কিন্তু কোনো ইনপুট নয়, এটি নিজে কোনো ক্যারেক্টার টাইপের ডেটা নয়। এর কোনো আসকি বা ইউনিকোড মান নেই। এটি একটি অবস্থা বা কন্ডিশন, যা বোঝায় যে উৎস থেকে ডেটা পড়া শেষ, এখান থেকে আর কোনো ডেটা পড়া যাবে না। আর বিভিন্ন প্রোগ্রামিং ভাষায় এই অবস্থাটি প্রোগ্রামিং লজিকে ব্যবহারের জন্য বিভিন্ন সিস্টেম ডিফাইনড কনস্ট্যান্ট ব্যবহার করে, যেমন সি প্রোগ্রামিংয়ের EOF (যার মান লাইব্রেরিতে ডিফাইন করা -1 হিসেবে)। তাই, আমি

কম্পিউটার প্রোগ্রামিং - বিতীয় খণ্ড

যদি কিবোর্ড থেকে -1 ইনপুট দেই, তাহলে, সেটি EOF বোঝাবে না, কারণ -1 বোঝাবে।

প্রতিবার ইনপুট নেওয়ার পরেই আমরা আবার ch-এর মানটি fputc ফাংশনের সাহায্যে আউটপুট ফাইলে লিখে দিচ্ছি। আর সব কাজ শেষে আমরা ফাইল দুটি বন্ধ করে দিচ্ছি। তুমি যদি প্রোগ্রামটি টিকঠাক চালিয়ে দেখো, তাহলে দেখবে যে তুমি যেই ইমেইজ ফাইলটি ইনপুট দিয়েছ, আউটপুট ফাইলে ছবিহীন সেই রকম ইমেজ তৈরি হয়েছে।

* ফাংশন fputc

প্রোটোটাইপ:

```
int fputc ( int char, FILE * stream)
```

ইনপুট:

```
int char : যে ক্যারেক্টারটি ফাইলে লিখতে হবে তার ইন্টিজার রূপ  
FILE * stream : যেই ফাইলে ডেটা লেখা হবে তার হ্যান্ডেল
```

রিটার্ন ভ্যালু:

```
int : যে ক্যারেক্টারটি লেখা হবে সেটিই রিটার্ন করে। কোনো এরর হলে EOF রিটার্ন করে।
```

কাজ:

```
stream ফাইলের ফাইল ইন্ডিকেটর পজিশনে char ক্যারেক্টারটি লিখে এবং  
ইন্ডিকেটরকে একক্ষরণ সামনে এগিয়ে দেয়।
```

কোনো ফাইলে কাজ করার সময় (পড়া বা লেখা), আমরা চাইলে কোনো নির্দিষ্ট জায়গা থেকে সেটি শুরু করতে পারি। fseek ফাংশন ব্যবহার করে আমরা ফাইল পজিশন ইন্ডিকেটর (File Position Indicator) এর মান পরিবর্তন করতে পারি। পরবর্তী প্রোগ্রাম লেখার আগে তোমাদেরকে in.txt নামে একটি ফাইল তৈরি করতে হবে। আমার ফাইলের ক্রিমশ্ট দিলাম (ছবি ৬)।

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড



ছবি ৬ একটি টেক্সট ফাইলের ছবি

এবারে নিচের প্রোগ্রামটি লিখে কম্পাইল ও রান কর:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     FILE *fp = fopen("in.txt", "r");
7     if (fp == NULL) {
8         perror("Can't open file");
9         return EXIT_FAILURE;
10    }
11
12    int ch;
13
14    ch = fgetc(fp);
15    printf("%c\n", (char)ch);
16    ch = fgetc(fp);
17    printf("%c\n", (char)ch);
18
19    fseek(fp, 0, 0);
20    ch = fgetc(fp);
21    printf("%c\n", (char)ch);
22
23    fclose(fp);
24
25    return 0;
26 }
```

প্রোগ্রাম ৩-৭
৫৮

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

আউটপুট আসবে এমন:

A
B
A

এখন দেখো, প্রথম দুটি ক্যারেক্টর প্রিণ্ট করার পরে আমি fseek ফাংশন দিয়ে ফাইল পজিশন ইন্ডিকেটর থেকে নির্দেশ করে ফাইলের ঠিক কোন জায়গা থেকে পড়া বা লেখা শুরু হবে। পরিবর্তন করে একেবারে শুরুতে নিয়ে দিয়েছি। ফাংশনের প্রথম প্যারামিটার হচ্ছে ফাইল পয়েন্টার, বিত্তীয় প্যারামিটার হচ্ছে কত ঘর সামনে আগামনে, আর তৃতীয় প্যারামিটার হচ্ছে, বিত্তীয় প্যারামিটারের যত ঘর সামনে আগামনের কথা বলা হয়েছে, সেটি কোন জায়গা থেকে শুরু করতে হবে। আমরা দ্বিতীয় ও তৃতীয় উভয় প্যারামিটারের মান 0 দিয়েছি। তাহলে 0 থেকে শুরু করে 0 ঘর এগোলে কিন্তু ফাইল পজিশন ইন্ডিকেটরের মান 0-ই হবে, অর্থাৎ একেবারে শুরু থেকে। তাই তোমরা তৃতীয় আউটপুটে আবার A দেখছ।

* ফাংশন fseek

প্রোটোটাইপ:

```
int fseek ( FILE * stream, long int offset, int whence)
```

ইনপুট:

FILE * stream : একটি ফাইল হ্যান্ডেল
long int offset : whence থেকে কত বাইট পরে তা নির্দেশ করে
int whence : যেই অবস্থানের সাথে offset যোগ করা হবে তার মান

রিটুর্ন ভ্যালু:

সফল হলে 0 রিটুর্ন করে, অন্যথায়, non-zero মান রিটুর্ন করে।

কাজ:

একটি ফাইল স্ট্রিম stream ইনপুট নেয় এবং whence থেকে offset ঘর সামনে
তার ফাইল পজিশন ইন্ডিকেটর সেট করে।

আমরা যদি fseek() ফাংশনটি এভাবে কল করতাম : `fseek(fp, 0, SEEK_SET);`
তাহলেও তোমরা একই আউটপুট দেখতে। SEEK_SET মানে হচ্ছে ফাইলের শুরু, SEEK_CUR
মানে হচ্ছে ফাইল পজিশন ইন্ডিকেটরের বর্তমান অবস্থান, আর SEEK_END মানে হচ্ছে

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

ফাইলের একেবারে শেষ পজিশন। এগুলো stdio.h হেডার ফাইলে ডিফাইন করা আছে। তাহলে তোমাদের এখন কাজ হবে প্রথমে ওপরের প্রোগ্রামটি পরিবর্তন করে `fseek(fp, 0, SEEK_SET)` ব্যবহার করে রান করে আউটপুট দেখো। তারপরের কাজ হচ্ছে সেই লাইনটির পরিবর্তে `fseek(fp, 0, SEEK_CUR)` ব্যবহার করে আবার প্রোগ্রামটি কম্পাইল ও রান কর। আউটপুট দেখে নিচ্যই বুঝতে পারছ যে আসলে ঘটনা কী ঘটছে।

• নোট:

`SEEK_SET` মানে ফাইলের শুরুর অবস্থান

`SEEK_CUR` মানে ফাইল পজিশন ইন্ডিকেটরের বর্তমান অবস্থান

`SEEK_END` মানে ফাইলের শেষ অবস্থান

এখন আমরা যদি চাই, ওপরের প্রোগ্রামে দুটি ক্যারেক্টার পড়ে প্রিন্ট করার পরে, তিনটি ক্যারেক্টার বাদ দিয়ে তার পরের ক্যারেক্টার পড়বে (মানে C, D, E বাদ দিয়ে একেবার F), তাহলে আমাদের তিনটি ক্যারেক্টারের জন্য যত ঘর জায়গা দরকার তত ঘর যোগ করতে হবে SEEK_CUR-এর সঙ্গে। একটি ক্যারেক্টারের সাইজ হচ্ছে `sizeof(char)`, তাহলে তিনটি ক্যারেক্টারের সাইজ হবে `sizeof(char) * 3`। তাহলে আমরা `fseek` ফাংশনটি কল করব এভাবে: `fseek(fp, sizeof(char) * 3, SEEK_CUR)`। তোমরা কাজটি কম্পিউটারে করে দেশো।

এখন তোমাকে যদি বলা হয় যে, তুমি যেই ইমেজ ফাইলটি তৈরি করলে, সেটির সাইজ কত, তা একটি প্রোগ্রাম লিখে বের করতে হবে, তাহলে কীভাবে করবে? এই কাজ করার জন্য আমাদেরকে আরেকটি ফাংশন সম্পর্কে জানতে হবে, `fsize`। বাইনারি ফাইলের ফেল্টে ফাংশনটি ইনপুট হিসেবে ফাইল পয়েন্টার নেয় আর ফাইল পজিশন ইন্ডিকেটর ফাইলের শুরু থেকে কত বাইট দূরে আছে, সেটি রিটার্ন করে। তাহলে আমরা যদি ফাইল পজিশন ইন্ডিকেটর একেবারে শেষে নিয়ে যাই, আর তারপরে `fsize` ফাংশন কল করি, তাহলেই ফাইলটি কত বাইট, সেটি জানা যাবে।

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     FILE *fp_in;
7     char *input_file = "image1.jpg";
8 }
```

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

```

9     fp_in = fopen(input_file, "rb");
10    if (fp_in == NULL) {
11        perror("file opening failed");
12        return EXIT_FAILURE;
13    }
14    fseek(fp_in, 0, SEEK_END);
15
16    printf("File Size: %ld bytes\n", ftell(fp_in));
17    printf("File Size (KiloByte): %ld\n", ftell(fp_in)/1024);
18
19    fclose(fp_in);
20
21    return 0;
22
23
24 }
```

প্রোগ্রাম ৩-৮

আর `ftell()` ফাংশনটির রিটার্ন টাইপ হচ্ছে long int, তাই প্রিন্ট করার জন্য `%ld` ব্যবহার করেছি।

• ফাংশন `ftell`

প্রোটোটাইপ:

`long int ftell (FILE * stream)`

ইনপুট:

`FILE * stream` : একটি ফাইল হ্যান্ডেল

রিটার্ন ভালু:

`long int` : ফাইল পজিশন ইন্ডিকেটর রিটার্ন করে

কাজ:

একটি `stream` ইনপুট নেয় ও তার বর্তমান ফাইল পজিশন ইন্ডিকেটর রিটার্ন করে।

বাইনারি ফাইলের জন্য আরো দুটি গুরুত্বপূর্ণ ফাংশন আছে, `fread` ও `fwrite`। তবে আমরা এখন পর্যন্ত কম্পিউটার প্রোগ্রামিং যতদূর শিখেছি, সেই জ্ঞান দিয়ে ফাঁশনগুলো বোঝা একটু

কঠিনই হবে, তাই আগতত আর আলোচনা করলাম না। পরে কোনো এক সময় তোমরা ইন্টারনেট থেকে ফাংশন দুটির ব্যবহার শিখে নেবে। এখন আরেকটি মজার কাজ করে এই অধ্যায়ের ইতি টানব। আমরা একটি ফাইল মোছার প্রোগ্রাম লিখব। এর জন্য stdio.h হেডার ফাইলে remove নামে একটি ফাংশন রয়েছে। এর ভেতরে প্যারামিটার হিসেবে ফাইলের নাম দিতে হয় আর ফাইলটি যদি ঠিকঠাক মোছা যায়, তাহলে ফাংশনটি ০ রিটুর্ন করে, নইলে অন্য কোনো মান রিটুর্ন করে। আমি কোডটি লিখে দিচ্ছি:

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int return_value;
6     char *filename = "image1.jpg";
7
8     return_value = remove(filename);
9
10    if (return_value != 0) {
11        perror("File Remove Failed");
12        return 1;
13    }
14
15    printf("%s removed successfully\n", filename);
16
17    // now try to remove again
18    return_value = remove(filename);
19
20    if (return_value != 0) {
21        perror("File Remove Failed");
22        return 1;
23    }
24
25    printf("%s removed successfully\n", filename);
26
27    return 0;
28 }
```

প্রোগ্রাম ৩-৯

তোমরা মনোযোগ দিয়ে কোডটি দেখ, তারপরে টাইপ করে কম্পাইল ও রান কর। আর কোনো বিষয় না বুঝলে একটু চিন্তা কর।
পরবর্তী অধ্যায়ে আমরা জানব, কম্পিউটার প্রোগ্রামিংয়ের খুব মজার এবং অভ্যন্তরীণ রিকার্শন।

অধ্যায় ৪ – রিকার্শন (Recursion)

৪.১ – লোকাল ও গ্লোবাল ভ্যারিয়েবল

রিকার্শন সম্পর্কে জানার আগে, চল, আমরা লোকাল ও গ্লোবাল ভ্যারিয়েবল সম্পর্কে একটু জেনে নিই। তুমি যখন একটি প্রোগ্রাম লিখ, তখন তার ভেতরে মেইন ফাংশনসহ আরও বিভিন্ন ফাংশন লিখ। এসব ফাংশনের ভেতরে যেসব ভ্যারিয়েবল ডিক্লেয়ার কর, সেগুলো হচ্ছে লোকাল ভ্যারিয়েবল। আর ফাংশনের বাইরেও ভ্যারিয়েবল ডিক্লেয়ার করা যায়, সেগুলোকে বলে গ্লোবাল ভ্যারিয়েবল। আবার কোনো ফাংশনের প্যারামিটারে যেসব ভ্যারিয়েবল পাঠানো হয়, সেগুলোও লোকাল ভ্যারিয়েবল। আমরা এখন কয়েকটি প্রোগ্রামের সাহায্যে গ্লোবাল ও লোকাল ভ্যারিয়েবলের পার্থক্য দেখব।

```

1 #include <stdio.h>
2
3 int x;
4
5 int main()
6 {
7     int y;
8
9     printf("x = %d, y = %d\n", x, y);
10
11    return 0;
12 }
```

প্রোগ্রাম ৪-১

ওপরের প্রোগ্রামটি লিখে কম্পাইল ও রান করলে দেখবে x-এর মান ০ প্রিন্ট হচ্ছে আর y-এর মান হিজিবিজি (গারবেজ - garbage) প্রিন্ট হচ্ছে। এখানে x হচ্ছে একটি গ্লোবাল ভ্যারিয়েবল, তাই এটি মেইন ফাংশনে ডিক্লেয়ার করা না হলেও মেইন ফাংশন থেকে একসেস করা যাচ্ছে। আর y হচ্ছে লোকাল ভ্যারিয়েবল। দুটি ভ্যারিয়েবলের কোনোটিতেই আমরা কোনো মান অ্যাসাইন করিনি। গ্লোবাল ভ্যারিয়েবল ডিক্লেয়ার করার সময় কোনো মান অ্যাসাইন না করলে ব্যবহৃত হবে ০ অ্যাসাইন হয়, কিন্তু লোকাল ভ্যারিয়েবলের ক্ষেত্রে সেটি হয় না। এজনাই এরকম আউটপুট।

এখন তোমরা নিচের কোডটি টাইপ করে প্রথমে বোকার চেষ্টা করবে যে, কী ঘটনা ঘটছে এবং আউটপুট কী হতে পারে। তারপরে সেটি কম্পাইল ও রান করে দেখবে যে তোমার প্রজ্ঞাশিত আউটপুট আসছে কি না। প্রয়োজন হলে কাগজ-কলমের সাহায্য নাও।

```

1 #include <stdio.h>
2
3 int x = 1;
4
5 void myfnc(int y)
6 {
7     y = y * 2;
8     x = x + 10;
9     printf("myfnc, x = %d, y = %d\n", x, y);
10 }
11
12 int main()
13 {
14     int y = 5;
15
16     x = 10;
17
18     myfnc(y);
19
20     printf("main, x = %d, y = %d\n", x, y);
21
22     return 0;
23 }
```

প্রোগ্রাম 8-২

এখনে আমরা একটি প্লোকাল ভ্যারিয়েবল *x* ডিক্লেয়ার করেছি এবং তার মধ্যে একটি ভ্যালুও অ্যাসাইন করে দিয়েছি। মেইন ফাংশনের ভেতরে প্রথমে *y* নামে একটি ভ্যারিয়েবল ডিক্লেয়ার করে তার মধ্যে 5 অ্যাসাইন করলাম। আবার *x* এর মধ্যে 10 অ্যাসাইন করলাম। তারপর *myfnc()* ফাংশনটি কল করলাম।

এখন প্রোগ্রামটি ওই ফাংশনের ভেতরে যাবে। শুরুতেই সে *y* নামে আরেকটি ভ্যারিয়েবল তৈরি করবে, যেটি ওই ফাংশনের জন্য একটি প্লোকাল ভ্যারিয়েবল, আর প্যারামিটার হিসেবে মেইন ফাংশন থেকে যে *y* পাঠাচ্ছি তার মানটি কপি করে তার নিজস্ব *y* ভ্যারিয়েবলে রাখবে। দেখো, এখনে কিন্তু দুটি *y* আলাদা, তারা দুটি আলাদা ফাংশনের প্লোকাল ভ্যারিয়েবল। একটি ফাংশনের প্লোকাল ভ্যারিয়েবল আরেকটি ফাংশনের ভেতর থেকে একসেস করা যায় না।

তারপর আমি *myfnc*-তে *y*-কে 2 দিয়ে গুণ করে ঘণকস্ব আবার *y*-কে আলাদেন করলাম। এখন *printf* ফাংশনটি কাজ করবে এবং *myfnc, x = 20, y = 10* প্রিণ্ট হবে। তারপরে ফাংশন থেকে বিটুর্স করে আবার ফাংশনে প্রোগ্রাম চল আসবে। এখন তোমার প্রোগ্রাম *myfnc* ফাংশনের প্লোকাল ভ্যারিয়েবল *y*-কে আবার চিনতে পারবে না, বা মনে রাখবে না। সে চিনবে *main* ফাংশনের প্লোকাল ভ্যারিয়েবল *y*-কে, তাই এর মান 5 প্রিণ্ট হবে। কিন্তু *x* যেহেতু প্লোকাল ভ্যারিয়েবল, তাই মেইন ফাংশনে *x*-এর মান 20 প্রিণ্ট হবে, কারণ *myfnc* ফাংশনে আমরা *x*-এর মান 10 বাড়িয়ে দিয়ে 20 করেছি। আশা করি তোমরা বুবাতে পেরেছ।

আর কোনো প্রোগ্রামে যদি একটি প্লোকাল ভ্যারিয়েবল থাকে আর একই নামে কোনো ফাংশনে একটি প্লোকাল ভ্যারিয়েবল থাকে, তখন কিন্তু ওই ফাংশন আর প্লোকাল ভ্যারিয়েবলটিকে চিনবে না, প্লোকাল ভ্যারিয়েবলকে চিনবে। এখন তোমাদের জন্য কাজ হচ্ছে, একটি প্রোগ্রাম লিখা, যেটি কেউ রান করলে এই ব্যাপারটি পরিষ্কারভাবে বুবাতে পারে।

8.2 - স্ট্যাটিক ভ্যারিয়েবল

এখন আমরা জন্যে স্ট্যাটিক ভ্যারিয়েবল সম্পর্কে। স্ট্যাটিক ভ্যারিয়েবল দুভাবে লেখা যায়, প্লোকাল ক্ষেপে (ফাংশনের বাইরে) আর ফাংশন ক্ষেপে (ফাংশনের ভেতরে)। প্রথমে চলো দেখি, স্ট্যাটিক প্লোকাল ভ্যারিয়েবল কী জিনিস। নিচের প্রোগ্রামটি লক্ষ কর:

```

1 #include <stdio.h>
2
3 int a;
4 static int b;
5
6 void func()
7 {
8     a = a+1;
9     b = b+1;
10 }
11
12 int main()
13 {
14     func();
15     printf("a = %d\n", a);
16     printf("b = %d\n", b);
17 }
```

কম্পিউটার প্রোগ্রামিং – দ্বিতীয় খণ্ড

প্রোগ্রাম 8-3

19 return 0;

20 }

21 }

প্রোগ্রাম 8-3

এই প্রোগ্রামের আউটপুট হবে এরকম:

a = 1
b = 1

এখনে a এবং b দুটি ভ্যারিয়েবলই main ফাংশনের বাইরে প্লোবাল স্পেসে ডিক্লেয়ার করা হয়েছে। আর তাই এদেরকে ইনিশিয়ালাইজ না করা সত্ত্বেও এদের মান হয়ে গেছে 0। এরপর func ফাংশনের মধ্যে আমরা এদের মান 1 করে বাড়িয়ে দিলাম। যেহেতু, এরা প্লোবাল ভ্যারিয়েবল তাই func বা main দুটি ফাংশনের মধ্যে থেকেই ভ্যারিয়েবল দুটিকে একসেস করা যায়। তাহলে এদের মধ্যে পার্থক্য কী? যদি তুমি বড় কোনো প্রজেক্ট কর, যেখানে একাধিক সি ফাইল আছে, তখন প্লোবাল ভ্যারিয়েবলগুলোকে যেকোনো ফাইলের যেকোনো ফাংশন থেকে একসেস করা যাবে। আর স্ট্যাটিক প্লোবাল ভ্যারিয়েবলগুলোকে শুধুমাত্র যেই ফাইলে ডিক্লেয়ার করা হয়েছে ওই ফাইলের ফাংশনগুলোর মধ্যেই একসেস করা যাবে। এটিই শুধু পার্থক্য।

আরেক ধরনের স্ট্যাটিক ভ্যারিয়েবল আছে যেগুলো ফাংশন ক্ষেত্রে থাকে। সেটা কী তা ব্যাখ্যা করার আছেই চলো একটা কোড করে দেখে নেই:

```

1 #include <stdio.h>
2
3 void func()
4 {
5     int a = 10;
6     static int s = 10;
7
8     a = a+2;
9     s = s+2;
10
11    printf("a = %d, s = %d\n", a, s);
12 }
13
14 int main()
15 {
16     func();
17     func();
18     func();

```

৬৬

কম্পিউটার প্রোগ্রামিং – দ্বিতীয় খণ্ড

প্রোগ্রাম 8-8

19 return 0;

20 }

21 }

এই প্রোগ্রামটির আউটপুট হবে এরকম:

a = 12, s = 12
a = 12, s = 14
a = 12, s = 16

লক্ষ কর, আমরা func ফাংশনে দুটি ইন্টিজার a ও s ডিক্লেয়ার করলাম। একটি সাধারণ ইন্টিজার ভ্যারিয়েবল, অন্যটি একটি স্ট্যাটিক ইন্টিজার ভ্যারিয়েবল। দুটিতেই মান অ্যাসাইন করলাম 10। তারপর ভ্যারিয়েবল দুটি সঙ্গে 2 যোগ করে প্রিন্ট করলাম। এখন main ফাংশন থেকে তিনিইরার func ফাংশনটি কল করলাম। a ভ্যারিয়েবলটি প্রিন্ট করল বিস্তৃt s ভ্যারিয়েবলটি যথাক্রমে 12, 14 ও 16 প্রিন্ট করল। এটিই হচ্ছে স্ট্যাটিক ভ্যারিয়েবলের বৈশিষ্ট্য। প্রথমবার যখন ফাংশন (যে ফাংশনে স্ট্যাটিক ভ্যারিয়েবলটি ডিক্লেয়ার করা হয়েছে) কল করা হয়, তখন সেই ভ্যারিয়েবলে যেই মান অ্যাসাইন করা হয়, এরপর ফাংশন থেকে বের হয়ে গেলেও ওই ভ্যারিয়েবলটি মুছে ফেলা হয় না। ওটা মেমোরিতে থেকে যাব, যতক্ষণ প্রোগ্রাম চলতে থাকে।

মজার ব্যাপার হচ্ছে ওই ভ্যারিয়েবলটিকে যেহেতু একটি ফাংশনের মধ্যে ডিক্লেয়ার করা হয়েছে তাই, ওই ফাংশনের বাইরে অন্য কোনো ফাংশনের মধ্যে তাকে আর একসেস করা যাবে না, শুধুমাত্র ওই ফাংশনটিকেই যদি আবার কল করা হয় তবেই কেবল ওই ভ্যারিয়েবলকে একসেস করা যাবে। সুতরাং দ্বিতীয়বার যখন আমরা func ফাংশনটিকে কল করি তখন s ভ্যারিয়েবলের মান 12 ই থেকে যায় এবং তার সঙ্গে 2 যোগ করে 14 প্রিন্ট করে।

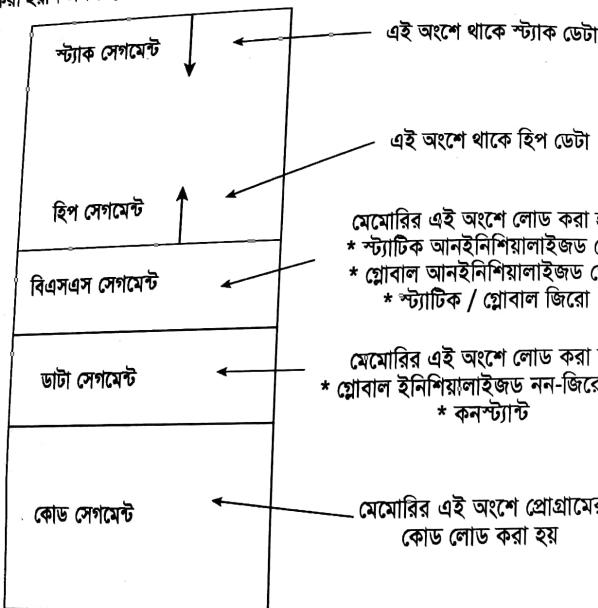
৮.৩ – বিভিন্ন প্রকারের মেমোরি

এখন আমরা মেমোরি সম্পর্কে আরেকটু জানব। প্রোগ্রাম চালানোর সময় অপারেটিং সিস্টেম মেমোরিকে কয়েকটি ভাগে ভাগ করে। সেগুলো হচ্ছে –

- ১) কোড সেগমেন্ট (code segment): প্রোগ্রামের যেসব অংশ বারবার এক্সিকিউট হয়, সেগুলো এখানে থাকে।

২) ডেটা সেগমেন্ট (data segment): এখানে থাকে ইনিশিয়ালাইজড (initialized) ডেটা অর্থাৎ, যেসব ফ্লোবাল ভ্যারিয়েবল ডিক্রেয়ার করার সঙ্গে সঙ্গে একটি মান আসাইন করা হয়, সেগুলো। এছাড়া কোনো কনস্ট্যাট বা প্রক্রিয়া মান থাকলে সেগুলোও এখানে লোড হয়। কনস্ট্যাট বিষয়ে আলোচনা করা হয়েছে অধ্যায় ১।

ডেটা সেগমেন্টেরই আরেকটি অংশ রয়েছে যাকে আনইনিশিয়ালাইজড (un-initialized) ডেটা সেগমেন্ট বা বিএসএস সেগমেন্ট (BSS segment) বলে। এই অংশে থাকে ইনিশিয়ালাইজেশন করা হয়নি এমন ফ্লোবাল ও স্ট্যাটিক ভ্যারিয়েবল।



ছবি ৭: প্রোগ্রামের বিভিন্ন প্রকার মেমোরি

৩) স্ট্যাক সেগমেন্ট (stack segment): লোকাল ভ্যারিয়েবল ও ফাংশনের ভেতরে যেসব আর্গুমেন্ট পাঠানো হয়, সেগুলো এখানে থাকে।

৪) হিপ সেগমেন্ট (heap segment): পয়েন্টার দিয়ে মেমোরি আলোকেশন করলে প্রোগ্রাম হিপ মেমোরি ব্যবহার করে। বিষয়টি তোমরা এখন বুঝতে পারার কথা নয়, তাই এটি নিয়ে আরও চিহ্নিত হয়ে না। বইটি যখন দ্বিতীয়বার পড়বে, তোমরা তখন একটি বুঝতে পারবে।

ওপরের অংশটুকু আমি একটি সহজ করে লিখলাম, এখন তোমাদের যতটুকু জানা দরকার ততটুকু। তোমরা যখন ভবিষ্যতে প্রোগ্রামিং নিয়ে আরও লেখাপড়া করবে, তখন আরও বিস্তারিত জানতে পারবে।

৮.৮ - রিকার্সন

এখন নিচের প্রোগ্রামটি হ্যাচ টাইপ করে কম্পাইল ও রান কর।

```

1 #include <stdio.h>
2
3 void recurse()
4 {
5     char *s = "Hurrey, I am learning recursion!";
6     printf("%s\n", s);
7     recurse();
8 }
9
10 int main()
11 {
12     recurse();
13     return 0;
14 }
```

প্রোগ্রাম ৮-৫

প্রোগ্রামটি কিছুক্ষণ চলে তারপরে বদ্ধ হয়ে যাবে। যদিও প্রোগ্রামটি আজীবন চলার কথা, কিন্তু কিছুক্ষণ চলেই সে তার জন্য বরাদ্দ সব মেমোরি (স্ট্যাক সেগমেন্ট-এ) খরচ করে ফেলবে। একটি ফাংশন থেকে যখন আরেকটি ফাংশন কল করা হয়, তখন কম্পিউটার যে ফাংশন থেকে কল করা হচ্ছে, সেই ফাংশনের ভ্যারিয়েবল এবং ফাংশনের কোন লাইন পর্যন্ত প্রোগ্রাম রান হয়েছিল, এসব তথ্য স্ট্যাক মেমোরিতে রাখে। তাই বারবার কল হতে থাকলে স্ট্যাকের জন্য বরাদ্দ সব জায়গা শেষ হয়ে যায়, তাই প্রোগ্রামটি বদ্ধ হয়ে যায়, একে ক্র্যাশ (crash) করা বলে।

কম্পিউটার প্রোগ্রামিং - বিজীয় খণ্ড

একটি খাতার কথা চিন্তা কর। প্রথম পৃষ্ঠা খুলে দেখলে সেখানে লেখা আছে, পরের পৃষ্ঠা টেক্সন। পরের পৃষ্ঠায় শিয়ে দেখলে সেখানেও একই কথা লেখা। এভাবে পাতা উচ্চাতে উচ্চাতে পৃষ্ঠায় শিয়ে দেখলে একই কথা লেখা, কিন্তু তারপরে খাতায় আব পৃষ্ঠা নেই। খাতা শেষ।

রিকার্শন হচ্ছে একটি ফাংশন নিজেই নিজেকে কল করা। তবে কল করার কাজটি একটু বুঝে শুনে করতে হবে। কখন রিকার্শন বক্ষ হবে, সেটি প্রোগ্রামের ভেতরে বলে দিতে হবে। নিচের প্রোগ্রামটি টাইপ করে কম্পাইল ও রান কর:

```

1 #include <stdio.h>
2
3 void recurse(int count)
4 {
5     if (count == 5) {
6         return;
7     }
8     printf("I am learning recursion.\n");
9     recurse(count+1);
10    return;
11 }
12
13 int main()
14 {
15     recurse(1);
16     return 0;
17 }
```

প্রোগ্রাম ৪-৬

I am learning recursion বাক্যটি কতবার প্রিন্ট হচ্ছে? এখানে আসলে কী ঘটে সেটি যদি তুমি বুঝতে চাও, তাহলে একটি খাতা নিয়ে তোমার বসতে হবে। কিংবা আলাদা আলাদা কতগুলো কাগজ হলেও চলবে। সেগুলো জোগাড় করে এখন আমি যা বলব, তা ঠিকঠাক অনুসরণ করতে হবে।

ফাংশন: main()
লাইন নম্বর: 1

ছবি ৮: প্রোগ্রামের বর্তমান অবস্থা কাগজে লেখ

৭০

কম্পিউটার প্রোগ্রামিং - বিজীয় খণ্ড

প্রথমে প্রোগ্রাম রান করা শুরু হবে মেইন ফাংশন থেকে। তাই একটি কাগজে main লিখে রাখ। তারপরে লিখতে হবে প্রোগ্রামটি এখন যত নম্বর লাইন এক্সিকিউট করবে সেই সংখ্যাটি। তাহলে তোমার লিখতে হবে ১ (ছবি ৮)।

এর অর্থ হচ্ছে আমি main ফাংশনের ১ নম্বর লাইনে রয়েছি এখন। তারপরে কী হবে? recurse ফাংশনটি কল হবে। এই কাজটি করার সময় কম্পিউটার মেইন ফাংশনের সব তথ্য স্ট্যাক মেমোরিতে জমা রাখবে। তুমি তাই কাগজটি একদিকে সরিয়ে রাখ। এখন প্রোগ্রাম recurse ফাংশনের ভেতরে চুকে গেল। সেখানে একটি ভ্যারিয়েবল তৈরি হবে, যার নাম count এবং প্রথমে এর মান হবে ১। কারণ মেইন ফাংশন থেকে ১ পাঠানো হয়েছে। তারপরে শর্ত পরীক্ষা হবে যে count এর মান ৫ এর সমান কি না। যেহেতু শর্তটি মিথ্যা, তাই ব্লকের ভেতরে না চুকে তার পরের লাইনে চলে যাবে এবং I am learning recursion. বাক্যটি স্ক্রিন প্রিন্ট হবে। তারপর প্রোগ্রামটি পরবর্তী লাইনে যাবে এবং recurse ফাংশনটি আবার কল করবে। যেহেতু বর্তমান ফাংশন থেকে বের হয়ে আরেকটি ফাংশন কল করা হচ্ছে (হোক না একই ফাংশন), কম্পিউটার করবে কী, recurse ফাংশনের সকল তথ্য স্ট্যাক মেমোরিতে জমা রাখবে। তুমি একটি কাগজে লিখে ফেল (ছবি ৯)।

ফাংশন: recurse()
লাইন নম্বর: 5
Count: 1

ছবি ৯: প্রোগ্রামের বর্তমান অবস্থা কাগজে লেখ

অর্থাৎ, আমি recurse ফাংশনের ৫ নম্বর লাইনে রয়েছি এবং এখানে count ভ্যারিয়েবলের মান ১ এবারে কাগজটি আগের কাগজের ওপরে রেখে দাও। ওপরে কেন রাখতে হবে? কারণ স্ট্যাক মানে হচ্ছে তুপ আর সেখানে আমরা কোনো কিছু রাখার সময় একটির ওপর আরেকটি রাখব। আর তোলার সময় সবচেয়ে ওপরের জিনিসটি তুলব।

এখন তোমার প্রোগ্রামটি আবার recurse ফাংশনের ভেতরে চুকবে। count-এর মান হবে ২, কারণ recurse থেকে কল করার সময় পাঠানো হয়েছিল count + 1। আবারও শর্ত পরীক্ষা হবে, শর্ত মিথ্যা হবে। চার নম্বর লাইনের প্রিন্ট স্টেটমেন্টটি চলবে। তারপর পাঁচ নম্বর লাইনে recurse ফাংশনটি কল হবে। আগেই মতোই কম্পিউটার বর্তমান recurse ফাংশনের সব তথ্য স্ট্যাকে জমা করবে। তুমি ও কর। একটি কাগজে লিখে সেটি আগের কাগজগুলোর ওপরে রাখ।



ছবি ১০: বইয়ের সূপ বা স্ট্যাক

এভাবে তুমি কাজ করতে থাক এবং কাগজ একটির ওপর আরেকটি রাখতে থাক যতক্ষণ পর্যন্ত না চুকবে আর সেখানে যেই return আছে, সেটি এক্সিকিউট হবে। তখন ইকাবের ভেতরে প্রোগ্রামটি করবে? যে তাকে কল করেছিল, তার কাছেই। তুমি তাহলে তোমার কাগজের সূপ থেকে সবচেয়ে করে না থাক।

ফাংশন: recurse()
লাইন নম্বর: 5
Count: 4

ছবি ১১: স্ট্যাকের সর্বশেষ অবস্থা

কাগজটি আর সেটি সূপে রাখবে না, অন্য কোথাও সরিয়ে রাখবে। এখন, ফাংশনের পরবর্তী লাইন এক্সিকিউট হবে। পরবর্তী লাইনও হচ্ছে return; - তাহলে ফাংশন থেকে রিটার্ন করবে, যে তাকে কল করেছিল তার কাছে। তাই তুমি সূপের সবচেয়ে ওপরে যেই কাগজটি আছে, সেটি সরাও। সেখানে দেখবে লেখা আছে ছবি ১২ এর কথাগুলো।

ফাংশন: recurse()
লাইন নম্বর: 5
Count: 3

ছবি ১২: একবার রিটার্ন করার পর স্ট্যাকের বর্তমান অবস্থা

এখন এই ফাংশনের ষষ্ঠ লাইনের রিটার্ন স্টেটমেন্ট এক্সিকিউট হবে ফাংশনটি সেখান থেকে কল করা হয়েছিল, সেখানে তোমার প্রোগ্রাম ফেরত যাবে। এভাবে করতে থাকলে দেখবে একসময় তোমার সূপে কেবল একটি কাগজ আছে, সেটি হচ্ছে main ফাংশনের কাগজ। তারপর সেটি তুল ফেলবে। তারা মানে তোমার প্রোগ্রাম আবার মেইন ফাংশনের ভেতরে চলে পোল। এখন সেই ফাংশনের পরের লাইন অর্থাৎ return 0; এক্সিকিউট হবে আর প্রোগ্রামটি বন্ধ হবে। আপা করি রিকার্শন তোমার কাছে পরিষ্কার হয়ে গেছে। আর পরিষ্কার না হলেও চিন্তার কিছু নেই। আমরা এখন আরও কয়েকটি প্রোগ্রাম লিখব। তুমি যদি প্রয়োজন মনে কর, তাহলে আগের মতো কাগজ-কলম নিয়ে বসতে পারো, বুবাতে সহজ হবে।

এখন নিচের প্রোগ্রামটি দেখে বল যে এখানে কী কাজ হচ্ছে? তারপর প্রোগ্রামটি লিখে কম্পাইল ও রান কর।

```
1 #include <stdio.h>
2
3 void recurse(int count)
4 {
5     if (count > 5) {
6         return;
7     }
8     printf("Count = %d\n", count);
9     recurse(count+1);
10 }
11
12 int main()
13 {
14     recurse(1);
15     return 0;
16 }
```

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় ধরণ

এখন একটি কথা মনে করিয়ে দিই। কোনো ফাংশনের রিটার্ন টাইপ যদি void হয়, তবে ফাংশনের শেষ লাইনে return; না লিখলেও চলে। আপনা আপনি সেটি রিটার্ন করে। void মানে কম্পাইলার তার কাছে থেকে কোনো কিছু আশা করে না, সে ঠিকঠাক ফেরত আসলেই খুশি!

এবারে আরেকটি প্রোগ্রাম লিখব। তোমরা প্রথমে এটির কোড দেখো। আগের প্রোগ্রামের সঙ্গে এর পার্থক্য কী, সেটি বোবার চেষ্টা কর। প্রয়োজন হলে খাতা-কলম ব্যবহার কর, বুবাতে অনেক সহজ হবে। আর সবশেষে প্রোগ্রামটি টাইপ করে কম্পাইল ও রান কর।

```

1 #include <stdio.h>
2
3 void recurse(int count)
4 {
5     if (count > 5) {
6         return;
7     }
8     recurse(count+1);
9     printf("Count = %d\n", count);
10 }
11
12 int main()
13 {
14     recurse(1);
15     return 0;
16 }
```

প্রোগ্রাম 8-8

আমি এখনে প্রোগ্রামগুলো ব্যাখ্যা করছি না, আর আউটপুট কী হবে, সেটিও বলে দিচ্ছি না। তোমাদের নিজের প্রথমে চিন্তা করতে হবে। পরে প্রোগ্রাম লিখে দেখতে হবে আউটপুট কী আসে। আমার মতে রিকার্শন শেখার জন্য এটিই সেরা উপায়।

এখন আরেকটি প্রোগ্রাম। এটিও তোমরা আগে চিন্তা করবে, কাগজ-কলমে আউটপুট বের করার চেষ্টা করবে। আর সবশেষে প্রোগ্রামটি টাইপ করে কম্পাইল ও রান করবে।

```

1 #include <stdio.h>
2
3 void recurse(int count)
4 {
5     if (count > 5) {
6         return;
7     }
```

98

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় ধরণ

```

8     printf("Count : %d\n", count);
9     recurse(count+1);
10    printf("Count = %d\n", count);
11 }
12 int main()
13 {
14     recurse(1);
15     return 0;
16 }
```

প্রোগ্রাম 8-9

এবারে চলো, এই কোডটিকে আমরা আরো একটু পরিবর্তন করি। আমরা দেখেছি, ফাংশন কোপে আমরা স্ট্যাটিক ভ্যারিয়েবল ব্যবহার করতে পারি। তাহলে একটি রিকার্শন ফাংশনের মধ্যে স্ট্যাটিক ভ্যারিয়েবল ব্যবহার করেই ফাংশনটি কতবার চলবে তা নিয়ন্ত্রণ করতে পারি। সেক্ষেত্রে আর count নামক ভ্যারিয়েবলটি আমাদের ফাংশনের প্যারামিটার হিসেবে পাঠাতে হবে না বাববাব। নিচের কোডটি লক্ষ কর:

```

1 #include <stdio.h>
2
3 void recurse()
4 {
5     static int count = 1;
6     if (count > 5) {
7         return;
8     }
9     printf("Count = %d\n", count);
10    count = count + 1;
11    recurse();
12 }
13
14 int main()
15 {
16     recurse();
17
18     return 0;
19 }
```

প্রোগ্রাম 8-10

তুমি যদি এভাবে ঠিকঠাকভাবে এসে থাকো, রিকার্শনের রহস্য তোমার কাছে স্টেডিয়ামের ফ্লাউইটের আলোর মতো পরিষ্কার হয়ে যাওয়ার কথা। যদি কোনো সমস্যা থাকে তাহলে আপনের অংশগুলো আবার পড়। তাতেও কাজ না হলে কোনো বন্ধু বা শিক্ষকের সাহায্য নাও। আগের অংশ না বুঝলে আমি এরপর যেসব প্রোগ্রাম দেখাব, সেগুলো বুঝতে পারবে না।

আরেকটি কথা বলা প্রয়োজন। রিকার্শন মানে যে কেবল ফাংশন নিজেই নিজেকে কল করা, তা নয়। যদি এমন হয় যে একটি ফাংশন A থেকে আরেকটি ফাংশন B কল করা হচ্ছে, B থেকে C কল করা হচ্ছে, C আবার A-কে কল করছে - তাহলে এটিও রিকার্শনের মধ্যেই পরে। তোমরা একটি প্রোগ্রাম লিখে দেখতে পারো যে এরকম করলে কী হয়।

আমরা এখন রিকার্শন ব্যবহার করে আরও কিছু প্রোগ্রাম লিখব। ভুলে গেলে চলবে না এ প্রোগ্রাম সেখার একমাত্র উপায় হচ্ছে প্রোগ্রামিং করা। আর আমার লক্ষ্য তোমাদেরকে আন্তর্জাতিক মানের প্রোগ্রামার হতে সহায়তা করা, আর সেটি হতে না পারলেও সমস্যা নেই, তোমরা যদি প্রোগ্রামিংয়ের সৌন্দর্য বুঝতে পারো, তাতেই আমার আনন্দ!

তোমরা যারা ৫২টি প্রোগ্রামিং সমস্যা ও সমাধান - বইটি পড়েছ, তারা সবাই ফ্যাক্টরিয়ালের সঙ্গে পরিচিত। তোমরা জানো যে ০ ও ১-এর ফ্যাক্টরিয়াল হচ্ছে ১। আর যেকোনো ধনাত্মক পূর্ণসংখ্যা n-এর ফ্যাক্টরিয়াল হচ্ছে ১ থেকে n পর্যন্ত ধনাত্মক পূর্ণসংখ্যার গুণফল। তাহলে আমরা লিখতে পারি:

$$n! = n * (n - 1) * (n - 2) * (n - 3) * \dots * 3 * 2 * 1$$

আবার, $(n - 1)! = (n - 1) * (n - 2) * (n - 3) * \dots * 3 * 2 * 1$

তাহলে আমরা লিখতে পারি, $n! = n * (n - 1)!$

এখন ফ্যাক্টরিয়াল যদি একটি ফাংশন হয়, তাহলে আমরা লিখতে পারি, $\text{factorial}(n) = n * \text{factorial}(n - 1)$ । তোমরা কি বুঝতে পারছ যে ফ্যাক্টরিয়াল বের করার জন্য রিকার্শন একটি চমৎকার পদ্ধতি হতে পারে? বুঝতে পারো আর না পারো, নিচের প্রোগ্রামটি দেখে দেখে টাইপ কর, উপরের কম্পাইল ও রান কর।

```
1 #include <stdio.h>
2
3 int factorial(int n)
4 {
5     if (n == 0) {
6         return 1;
7     }
8 }
```

৭৬

```
8     return n * factorial(n - 1);
9 }
10
11 int main()
12 {
13     int n;
14     scanf("%d", &n);
15
16     if (n < 0) {
17         printf("Undefined\n");
18         return 0;
19     }
20
21     printf("Factorial of %d is %d\n", n, factorial(n));
22
23     return 0;
24
25 }
```

প্রোগ্রাম ৮-১১

ওপরের প্রোগ্রামটি কীভাবে কাজ করছে যদি বুঝতে না পারো, তবে কাগজ-কলম নিয়ে বসে যাও। ধর তুমি ইনপুট হিসেবে n-এর মান দিলে ৫। তাহলে main ফাংশন থেকে factorial(5) কল হবে। factorial ফাংশনের ভেতরে n-এর মান প্রথমে ৫। তাই ফাংশনের ভরতে যে if-এর ভেতরে শর্ত দেওয়া আছে, সেটি মিথ্যা হবে। তখন ফাংশনের `return n * factorial(n - 1);` লাইনে প্রোগ্রামটি চলে আসবে। এখানে ৫ (n-এর মান) এর সঙ্গে factorial(n - 1) ভর করে রিটার্ন করবে। কিন্তু যেহেতু factorial একটি ফাংশন, তাই সেটি আবার কল হবে। তাই প্রোগ্রামটি আবার factorial ফাংশনের ভেতরে ঢুকবে। এবারে n-এর মান হবে ৪। কারণ কল করা হয়েছে factorial(n - 1)। এভাবে প্রোগ্রামটি বারবার factorial ফাংশনের ভেতরে ঢুকবে আর n-এর মানও এক করে কর্মতে থাকবে। n-এর মান যখন ০ হবে, তখন ফাংশনটি ১ রিটার্ন করবে। কারণ এবার if-এর ভেতরে শর্ত সত্য। রিটার্ন করবে কার কাছে? যে তাকে কল করেছিল। তাহলে `return 1 * factorial(1 - 1);` এই জায়গায় রিটার্ন করবে এবং তারপরে কম্পিউটার দেখবে যে সে factorial(1 - 1)-এর মান পেয়ে গেছে, যা কী না । তখন `return 1 * 1` লাইনটি এক্সিকিউট হবে, মানে ফাংশনটি 1 রিটার্ন করবে। কোথায় করবে? `return 2 * factorial(2 - 1);` এখানে। তাহলে এটি আবার 2 * 1 মানে 2 রিটার্ন করবে, কোথায়? এখানে - `return 3 * factorial(3 - 1);`। এই লাইনটি আবার $(3 * 2) = 6$ রিটার্ন করবে আর প্রোগ্রাম ফেরত যাবে `return 4 * factorial(4 - 1);` লাইনে। তাহলে এটি `return 4 * 6;` মানে 24 রিটার্ন করে ফেরত যাবে `return`

৭৭

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

`5 * factorial(5 - 1);` লাইনে `factorial(5 - 1)` রিটার্ন করেছে 24। তাই সাইটটি $5 * 24$ মানে 120 রিটার্ন করে `main` ফাংশনের কাছে ফেরত যাবে। কারণ মেইন ফাংশনটি `factorial(5)` কল করেছিল!

প্রত্যেক মনে রাখবে:

একনাগাড়ে যখন তুম তিনি চার ঘন্টা প্রোগ্রামিং করবে, সঙ্গে পানি রাখতে হবে। নিয়মিত পানি পান তোমাকে কেবল সুস্থই রাখে না, তোমার মস্তিষ্ককেও সচল রাখবে। তাই একটু পরপর পানি পান করবে। লেবুর শরবত কিংবা স্যালাইনও পান করতে পারো মাঝে মধ্যে, তাবে তিনি যত কম খাওয়া যায় ততই ভালো। আর প্রোগ্রামিং করার সময় বেশি বেশি পানি পান তোমাকে আরেকভাবে সহায়তা করবে। এটি নিশ্চিত করবে যে তুমি একনাগাড়ে খুব বেশি সময় কম্পিউটারের সামনে বসে থাকছ না!

এখন আমরা আবার প্রোগ্রামিংয়ে ফেরত যাই। আগের ফ্যাক্টরিয়াল বের করার প্রোগ্রামে `factorial` ফাংশনটি কত বার কল করা হয়েছে, সেটি কীভাবে বের করব? এর জন্য মোবাইল ড্যারিয়েবলের প্রয়োজন। তোমরা প্রোগ্রামটি নিজে সেখার চেষ্টা করে দেখো।

```

1 #include <stdio.h>
2
3 int f_calls = 0;
4
5 int factorial(int n)
6 {
7     f_calls = f_calls + 1;
8
9     if (n == 0) {
10         return 1;
11     }
12
13     return n * factorial(n - 1);
14 }
15
16 int main()
17 {
18     int n;
19
20     scanf("%d", &n);
21
22     if (n < 0) {

```

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

```

23     printf("Undefined\n");
24     return 0;
25 }
26
27
28 printf("Factorial of %d is %d\n", n, factorial(n));
29 printf("Number of function calls : %d\n", f_calls);
30
31 }
32 }
```

প্রোগ্রাম 8-12

এখন আমরা লিখব ফিবোনাচি সংখ্যা বের করার প্রোগ্রাম। যদিও লুপ ব্যবহার করে প্রোগ্রামটি খুব সহজেই করা যায়, আমরা রিকার্শন ব্যবহার করে করব, কারণ আমরা খুব ভালোভাবে রিকার্শন শেখার চেষ্টা করছি।

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int f_calls = 0;
5
6 int fib(int n)
7 {
8     f_calls = f_calls + 1;
9
10    if (n == 1 || n == 2) {
11        return 1;
12    }
13
14    return fib(n - 1) + fib(n - 2);
15 }
16
17 int main()
18 {
19     int n;
20     char s[3];
21
22     scanf("%d", &n);
23
24     if (n == 1) {
25         strcpy(s, "st");
26     }

```

```

27     else if (n == 2) {
28         strcpy(s, "nd");
29     }
30     else if (n == 3) {
31         strcpy(s, "rd");
32     }
33     else {
34         strcpy(s, "th");
35     }
36
37     printf("%d%s fibonacci number is %d\n", n, s, fib(n));
38
39     printf("Number of function calls : %d\n", f_calls);
40
41     return 0;
42 }

```

প্রোগ্রাম 8-১৩

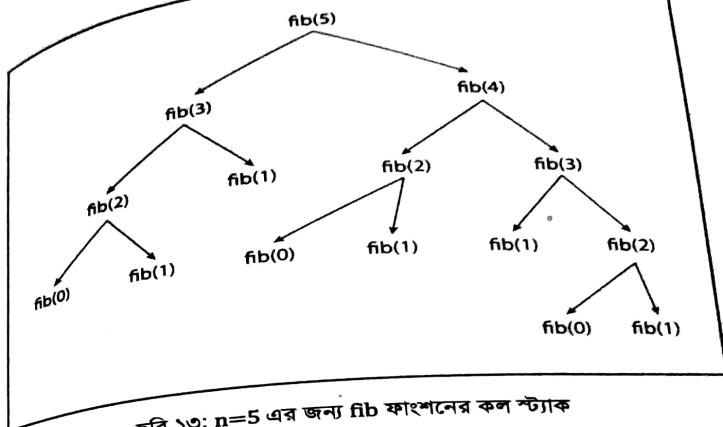
তোমরা ওপরের প্রোগ্রামটি প্রথমে লিখে কম্পাইল ও রান কর। তারপরে কাগজ-কলম নিয়ে বসো। এটি বোৰা কিন্তু স্ট্যাটুরিয়াল বের করার প্রোগ্রামের মতো অত সহজ হবে না। ছবি ১৩ তোমাকে সাহায্য করতে পারে।

তুম যদি বিভিন্ন ইনপুট দিয়ে দেখো, দেখবে যে প্রোগ্রামটি একটু বড় সংখ্যার জন্য চলতেই বেশ ব্যালিকটা সময় নিছে। কারণ কী? একে বারবার fib ফাংশন কল করতে হচ্ছে। কতবার কল হচ্ছে সেটিও তোমরা প্রোগ্রামের আউটপুটে দেখতে পাবে। ওপরের ছবিতে দেখো, n-এর মান মাত্র 5-এর জন্যই তাকে 2 বার fib(3), 3 বার fib(2) কল করতে হয়েছে। আমরা যদি 7-ত ফিবোনাচি সংখ্যা বের করতে চাই, তাহলে ঘটনা ঘটবে ছবি ১৪ এর মতো।

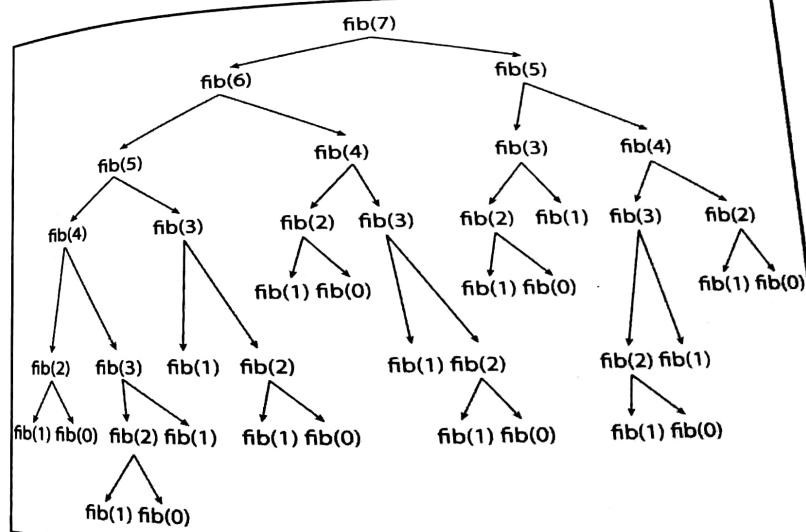
তাহলে দেখো একই প্যারামিটার দিয়ে fib ফাংশন অনেকবার কল হচ্ছে। যেমন বারবার আমাদেরকে fib(3), fib(2), fib(1)-এর মান বের করতে হচ্ছে। কিন্তু যদি কোনো ফিবোনাচি সংখ্যা প্রথমবার বের করেই সেটি কোনো অ্যারেতে রেখে দেওয়া যায়, আর fib(n) কল করার আগে সেই n-তম ফিবোনাচি সংখ্যা অ্যারেতে আছে কি না, আর থাকলে সেটি রিটার্ন করে দেওয়া যায়, তাহলে কিন্তু আমাদের প্রোগ্রামের কাজ অনেক কমে যায়। তোমরা কি নিজেরা প্রোগ্রামটি লিখতে পারবে? একটি প্রোবাল আরে ব্যবহার করতে হবে।

আমি আমার কোড দিয়ে দিচ্ছি, তোমরা এটি টাইপ করে কম্পাইল ও রান করে দেখো। তবে সবচেয়ে ভালো হয় নিজেরা আগে কয়েক ঘটনা চেষ্টা করলে।

কম্পিউটার প্রোগ্রাম - বিজ্ঞান থত



ছবি ১৩: n=5 এর জন্য fib ফাংশনের কল স্ট্যাক



ছবি ১৪: n=7 এর জন্য fib ফাংশনের কল স্ট্যাক

১০. কলন স্ট্রিং

প্রোটোটাইপ:
char * strcpy (char * dest, const char * src)

ইনপুট:
char * dest : যেখানে স্ট্রিং কপি করা হবে তার পয়েন্টার
const char * src : যেই স্ট্রিং কপি করা হবে তার পয়েন্টার

রিটার্ন ভালুক:
char * : যেখানে স্ট্রিং কপি করা হয়েছে তার পয়েন্টার

কাজ:
src স্ট্রিংকে dest স্ট্রিংে কপি করে তার পয়েন্টার রিটার্ন করে।

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int f_calls = 0;
5 int fibo[50];
6
7 int fib(int n)
8 {
9     f_calls = f_calls + 1;
10    if (fibo[n] != 0) {
11        return fibo[n];
12    }
13
14    if (n == 1 || n == 2) {
15        return fibo[n];
16    }
17
18    fibo[n] = fib(n - 1) + fib(n - 2);
19
20    return fibo[n];
21 }
22
23 int main()
24 {
```

```

15     int n;
16     char s[3];
17
18     fibo[1] = 1;
19     fibo[2] = 1;
20
21     scanf("%d", &n);
22
23     if (n == 1) {
24         strcpy(s, "st");
25     } else if (n == 2) {
26         strcpy(s, "nd");
27     } else if (n == 3) {
28         strcpy(s, "rd");
29     } else {
30         strcpy(s, "th");
31     }
32
33     printf("%d%s fibonacci number is %d\n", n, s, fib(n));
34     printf("Number of function calls : %d\n", f_calls);
35
36     for (n = 1; n < 12; n++) {
37         printf("%d: %d\n", n, fibo[n]);
38     }
39
40     return 0;
41 }
```

প্রোগ্রাম 8-18

প্রোগ্রামটি আমি আর ব্যাখ্যা করব না। কারণ এটি বুঝতে হলে যা যা জানা প্রয়োজন, সবই তোমাদের জানা হয়ে গিয়েছে। তাই তোমরা নিজেরা একটু মাথা খাটিয়ে বুঝে নাও। প্রয়োজনে কোনো বদ্ধ কিংবা বড় ভাই-আপুর মাথা ধার নাও।

এখন তোমাদের জন্য আরও কাজ।

- 1) তোমরা তো সবাই লুপ ব্যবহার করে 1 থেকে n পর্যন্ত পূর্ণসংখ্যার যোগফল নির্ণয় করতে পারো। এই কাজটি এখন তোমাদের করতে হবে রিকার্শন ব্যবহার করে। প্রোগ্রামে কোনো লুপ ব্যবহার করা যাবে না।

অধ্যায় ৫ - বিটওয়াইজ অপারেশন (Bitwise Operation)

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

- 2) একটি স্ট্রিং উল্লে প্রিন্ট করতে হবে (রিভার্স), আর সেটিও লুপ ব্যবহার না করে রিকার্চনের সাহায্যে।
- 3) একটি আরেতে দশটি পূর্ণসংখ্যা আছে। সংখ্যাগুলোর সব রকম বিন্যাস বা পারমুটেশন প্রিন্ট করতে হবে। বিন্যাস জানা না থাকলে উচ্চ মাধ্যমিকের গণিত বই পড়ে নিশ্চে পারো, তেমন কঠিন কিছু নয়।

নি প্রোগ্রামিং ভাষায় আমরা যখন বিভিন্ন ভারিয়েবল ব্যবহার করি, সেগুলোর স্কুলতম একক হচ্ছে বাইট। যেমন, ক্যারেক্টার টাইপের ভারিয়েবল হচ্ছে এক বাইট। আমরা যদি চাই যে, কেবল বাইটের মধ্যেই সীমাবদ্ধ না থেকে বিট নিয়েও কাজ করব, সি-তে সেই সুবিধা আছে। বিভিন্ন ধরনের বিটওয়াইজ অপারেটর (bitwise operator) ব্যবহার করে কাজগুলো আমরা করতে পারি। এই অধ্যায়ে কিছু বিটওয়াইজ অপারেটরের ব্যবহার দেখব।

প্রথমেই বিটওয়াইজ নট (not), যাকে প্রকাশ করা হয়ে ~ চিহ্ন দিয়ে। এটি একটি ইউনারি অপারেটর, অর্থাৎ কেবল একটি জিনিসের ওপরই অপারেশন করতে পারে। তো নট (unary) অপারেটর, অর্থাৎ কেবল একটি জিনিসের ওপরই অপারেশন করতে পারে। আর যেহেতু বিট নিয়ে মানে হচ্ছে না, তামি যেটি ইনপুট দিবে, সে সেটির উল্লে আউটপুট দিবে। আর যেহেতু বিট নিয়ে কাজ করবার, তাই 0 ইনপুট দিলে আউটপুট দিবে 1, আর 1 ইনপুট দিলে আউটপুট দিবে 0। কিন্তু 0 বা 1 বিট ইনপুট দেওয়ার তো উপায় নেই, তাই তামি যদি কোনো ইন্টিজার যেমন 1 ইনপুট দাও, তাহলে তার সবগুলো বিট উল্লে যাবে। তো ইন্টিজার 1 এর জন্য যদি 32 বিট থাকে, তাহলে সেটি হবে এমন : 00000000 00000000 00000000 00000001। আবার আমরা কিন্তু চাইলে ইন্টিজারের পরিবর্তে ক্যারেক্টার ব্যবহার করতে পারি, কারণ ক্যারেক্টারের সাইজ হচ্ছে এক বাইট আর -128 থেকে 127 সংখ্যাগুলো নিয়ে কাজ করতে চাইলে এক বাইটই যথেষ্ট। এখন, এক বাইট ব্যবহার করলে 1-এর বাইনারি মান হবে 00000001। এখন সবগুলো বিট যদি উল্লে দিই, তাহলে সেটি হবে 11111111। তোমরা দুজ কমপ্লিমেন্ট (Two's complement)-এর সঙ্গে পরিচিত, তারা একটি হিসাব করলেই দুবারে যে এটি হচ্ছে -2-এর বাইনারি (তুর্জ কমপ্লিমেন্ট পদ্ধতিতে)। যারা বিষয়টির সঙ্গে পরিচিত নও, তারা একাদশ-দ্বাদশ প্রেগির আইসিটি বই দেখতে পারো, সেখানে বিজ্ঞাপিত দেওয়া আছে। এখন আমরা কোড লিখে দেখব যে কী হয়।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     char a, b;
6
7     a = 0;
8     b = ~a;
9     printf("a = %d, b = %d\n", a, b);
10

```

কম্পিউটার প্রোগ্রামিং – দ্বিতীয় খণ্ড

```

131     a = 1;
132     b = -a;
133     printf("a = %d, b = %d\n", a, b);
134
135     return 0;
136 }
```

গ্রোহাম ৫-১

তোমরা ওপরের কোডটি কম্পাইল ও রান করলে দেখবে যে আউটপুট আসবে এরকম –

```
a = 0, b = -1
a = 1, b = -2
```

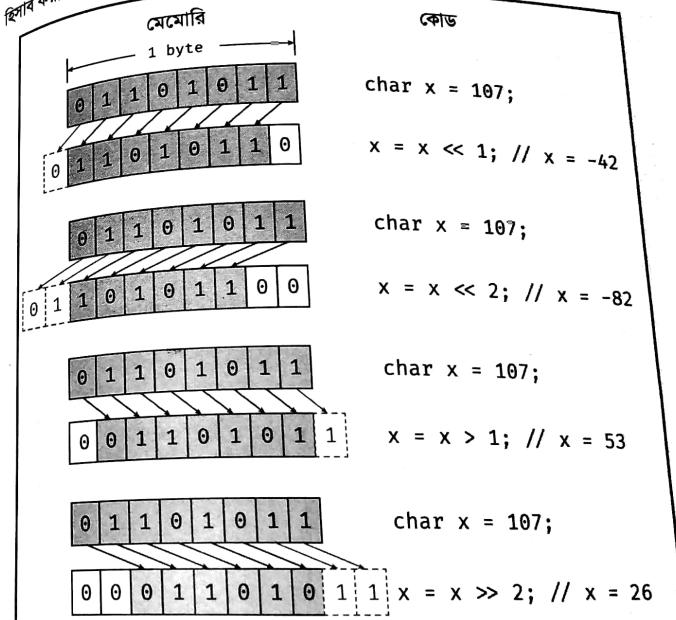
প্রথম ক্ষেত্রে a-এর মান 0, তাহলে এর বাইনারি রূপ হবে এমন: [00000000]। তার সবগুলো বিট টুল্ট দিলে সেটি হবে [11111111]। এটি হচ্ছে -1-এর বাইনারি (টুজ কমপ্লিমেন্ট পদ্ধতিতে)। আর দ্বিতীয় ক্ষেত্রে কী হয়, সেটি একটি আগেই বলেছি। আর টুজ কমপ্লিমেন্ট এখন না বুবলেও চলবে। কেবল জনে রাখ যে এই বিটওয়াইজ নট অপারেটরের কাজ হচ্ছে সবগুলো বিট উল্টো দেওয়া (1 হল 0 আর 0 হল 1)। আর এটি কিন্তু লজিক্যাল নট (!) অপারেটর থেকে আলাদা, শুধুয়ে ফেল না যান।

এখন আমরা দেখব শিফট (shift) অপারেটর। এখানে শিফট করা মানে সরানো। শিফট আবার দুই দিকে করা যায়, বামদিকে কিংবা ডানদিকে। তাই সি-তে দেখবে লেফট শিফট ও রাইট শিফট এই দুই রকম অপারেটর আছে। লেফট শিফট কোনো সংখ্যার বিটগুলো সব বামদিকে সরাবে আর রাইট শিফট সরাবে ডান দিকে। আর যত ঘর বলবে, তত ঘর সরাবে। যেমন আমি যদি লিখি $x \ll 2$ তাহলে x-এর সবগুলো বিট বামদিকে দুই ঘর সরে যাবে, আর যদি চাইতাম যে ডানদিকে তিন ঘর সরাব, তাহলে লিখতাম $x \gg 3$ । আর এই যে বিট সরিয়ে দেওয়া হচ্ছে, তাহলে শূন্যস্থান পূরণ হবে কী দিয়ে? সেগুলো সব 0 বিট দিয়ে পূর্ণ হবে।

এখন ধর 1-এর বাইনারি রূপ হচ্ছে [00000001]। এখন এই সংখ্যার বিটগুলোকে যদি এক ঘর বামে সরাই, তাহলে সেটি হবে [00000010], যা 2-এর বাইনারি রূপ। এখানে 1 বামদিকে সরে গেল আর সেই ফাঁকা ঘরটি 0 দিয়ে পূরণ হলো। আর একেবারে বামদিকে যে 0 ছিল, তার কী হবে? সে হারিয়ে যাবে। আবার ধর 5-এর বাইনারি রূপ হচ্ছে [00000101]। এখন এর বিটগুলো সব এক ঘর বামে সরালে সেটি হবে [00001010], যা কী না 10 এর বাইনারি রূপ। আবার 10-এর বাইনারি রূপ হচ্ছে [00001010], তাই একে এক ঘর ডানদিকে সরালে সেটি হবে [00000101], অর্থাৎ 5। আরও এক ঘর ডানদিকে সরালে হবে [00000010], যা 2 এর বাইনারি রূপ। তোমরা কেউ কেউ হ্যাতো ভাবছ, তার মানে কি কোনো সংখ্যাকে এক ঘর লেফট শিফট

কম্পিউটার প্রোগ্রামিং – দ্বিতীয় খণ্ড

কোরা মানে তাকে 2 দিয়ে গুণ করা আর এক ঘর রাইট শিফট করা মানে 2 দিয়ে ভাগ করা (কেবল ভাগফল বিবেচনা করে, ইন্টিজার টাইপের সংখ্যাকে ইন্টিজার দিয়ে ভাগ করলে কেবল ভাগফল হিসাব করা হয়, ভাগশেষ বাদ দিয়ে দেওয়া হয়) ? হ্যাঁ, ঠিক তাই।



ছবি ১৫: বিভিন্ন রকম বিটওয়াইজ অপারেশন

আমি নিচে একটি কোড লিখে দিচ্ছি। এটি তোমরা কম্পাইল করে চালাবে আর নিজেরাও নামান রকম পরীক্ষা করে দেখবে যে কী রকম ইনপুটের জন্য কী রকম আউটপুট আসে।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int n, x, m;
```

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

```

6   while(1) {
7     printf("Please enter your number (0 to exit): ");
8     scanf("%d", &n);
9     if (n == 0) {
10       break;
11     }
12     printf("How many bits you want to shift left? ");
13     scanf("%d", &x);
14
15     m = n << x;
16
17     printf("Result is %d\n\n", m);
18   }
19
20   return 0;
21 }
22 }
```

প্রোগ্রাম ৫-২

প্রোগ্রামটি আমি চালিয়ে নিচের মতো ইনপুট দিয়ে কী আউটপুট পেলাম দেখো। তোমরা তোমাদের মতো ইনপুট-আউটপুট দিয়ে দেখতে পারো।

Please enter your number (0 to exit): 9
How many bits you want to shift left? 1
Result is 18

Please enter your number (0 to exit): 9
How many bits you want to shift left? 2
Result is 36

Please enter your number (0 to exit): 2
How many bits you want to shift left? 2
Result is 4

Please enter your number (0 to exit): 2
How many bits you want to shift left? 8
Result is 16

Please enter your number (0 to exit): 5
How many bits you want to shift left? 3
Result is 16

৮৮

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

How many bits you want to shift left? 1
Result is 18

Please enter your number (0 to exit): 5
How many bits you want to shift left? 2
Result is 20

Please enter your number (0 to exit): 5
How many bits you want to shift left? 3
Result is 40

Please enter your number (0 to exit): 0

এখন তোমাদের কাজ হবে ওপরের প্রোগ্রামটি রাইট শিফট ব্যবহার করে লিখা এবং বিভিন্ন ইনপুট দিয়ে পরীক্ষা-নিরীক্ষা করা।

এখন আমরা দেখব বিটওয়াইজ অ্যান্ড (&) আর বিটওয়াইজ অর (|) অপারেটরের ব্যবহার। এগুলো কিন্তু লজিক্যাল অ্যান্ড (&&) ও লজিক্যাল অর (||) অপারেটর - যেগুলো আমরা কন্ডিশনাল লজিকে ব্যবহার কর, সেগুলো থেকে আলাদা। এগুলো কেবল বিটের ওপর কাজ করে।

5 & 6 এর ফলাফল কী হবে? 5-এর বাইনারি রূপ হচ্ছে **00000101** আর 6-এর বাইনারি রূপ হচ্ছে **00000110**। বিটওয়াইজ অ্যান্ড অপারেটর করবে কী, 5-এর একদম ডানদিকের বিটের সঙ্গে 6 এর একদম ডানদিকের বিটের মধ্যে অ্যান্ড অপারেশন চালাবে। সেক্ষেত্রে কেবল দুটি যদি 1 হয়, তাহলে আউটপুট হবে 1 আর অন্য সব ক্ষেত্রে আউটপুট 0 (মানে কেবল ওই দুটি বিটের মধ্যে লজিক্যাল অ্যান্ডের মতো অপারেশন করবে)। এরপর ডানদিক থেকে দ্বিতীয় বিটগুলোর মধ্যে অ্যান্ড অপারেশন, তৃতীয় বিটগুলোর মধ্যে অ্যান্ড অপারেশন, এভাবে চলবে, একবারে বামদিকের প্রথম বিট পর্যন্ত।

5 = 0000 0101
6 = 0000 0110

5&6 = 0000 0100

00000100 হচ্ছে 4 এর বাইনারি রূপ। তাহলে একটি প্রোগ্রাম চালিয়ে দেখি বিষয়টি সত্য কি না।

1 #include <stdio.h>
2

৮৯

কম্পিউটার প্রোগ্রামিং – বিতীয় খণ্ড

```

3 int main()
4 {
5     int n1 = 5, n2 = 6, n3;
6
7     n3 = n1 & n2;
8
9     printf("%d & %d = %d\n", n1, n2, n3);
10
11    return 0;
12 }

```

প্রোগ্রাম ৫-৩

ওপরের প্রোগ্রামটি চালিয়ে দেখো যে আউটপুট কী আসল।

এখন আসা যাক বিটওয়াইজ অর-এর কথায়। $5 | 6$ এর ফলাফল কী হবে? 5-এর বাইনারি রূপের একেবারে ডানদিকের বিটের সঙ্গে 6-এর বাইনারি রূপের একেবারে ডানদিকের বিটের মধ্যে অর অপারেশন চলবে আর কেবল দুটিই 0 হলে ফলাফল 0 হবে, নতুনে 1 হবে (যেকোনো একটি 1 কিংবা দুটিই 1 হলে)। তারপর ডানদিক থেকে বিতীয় বিটের মধ্যে একই কাজ হবে, এভাবে চলবে, একেবারে বামদিকের প্রথম বিট পর্যন্ত।

5 =	0000 0101
6 =	0000 0110
<hr/>	
$5 6 =$	0000 0111

এখন 00000111 হচ্ছে 7-এর বাইনারি রূপ। প্রোগ্রাম লিখে দেখি ঘটনা সত্য কি না।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int n1 = 5, n2 = 6, n3;
6
7     n3 = n1 | n2;
8
9     printf("%d | %d = %d\n", n1, n2, n3);
10
11    return 0;
12 }

```

প্রোগ্রাম ৫-৪

১০

কম্পিউটার প্রোগ্রামিং – বিতীয় খণ্ড

প্রোগ্রামটি রান করলে দেখবে আউটপুট হবে 7।
এখন দেখব এক্সর (xor) অপারেটরের কাজ। এক্সর হচ্ছে এক্সক্লিসিভ অর (exclusive or)-এর সংক্ষিপ্ত রূপ। এর জন্য \wedge চিহ্ন ব্যাবহার করা হয়েছে। দুটি বিটের তুলনা করার সময় যদি যেকোনো একটি বিট 1 এবং অপর বিটটি 0 হয়, তখন আউটপুট হবে 1, আর দুটিই যদি 1 কিংবা দুটিই যদি 0 হয়, তাহলে আউটপুট হবে $0 \wedge 1 \wedge 6$ এর মান কী হবে?

5 =	0000 0101
6 =	0000 0110
<hr/>	
$5 \wedge 6 =$	0000 0011

00000011 হচ্ছে 3-এর বাইনারি রূপ। তাই ফলাফল হবে 3। আমরা প্রোগ্রাম লিখে যাচাই করি।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int n1 = 5, n2 = 6, n3;
6
7     n3 = n1 ^ n2;
8
9     printf("%d ^ %d = %d\n", n1, n2, n3);
10
11    return 0;
12 }

```

প্রোগ্রাম ৫-৫

তোমরা প্রোগ্রাম রান করে দেখো আউটপুট কী আসে?

এখন আমরা এই অধ্যায়ে যা শিখলাম, সেগুলো ব্যবহার করে মজার কিছু কাজ করব।
তোমরা সবাই নিশ্চয়ই জোড়-বিজোড় সংখ্যা নির্ণয় করার প্রোগ্রাম লিখতে পারো। এর জন্য সংখ্যাকে 2 দিয়ে ভাগ করে ভাগশেষ দেখে সিদ্ধান্ত নাও, ভাগশেষ শূন্য হলে জোড়, নইলে বিজোড়। তোমরা যদি বিভিন্ন সংখ্যার বাইনারি রূপ লক্ষ কর, দেখবে জোড় সংখ্যাগুলোর একেবার ডানদিকের বিটটি হচ্ছে 0, আর বিজোড় সংখ্যার বেলায় 1। আমি নিচে উদাহরণস্বরূপ
0 থেকে 16 পর্যন্ত সবগুলো সংখ্যার বাইনারি রূপ লিখে দিচ্ছি:

১

কম্পিউটার প্রোগ্রামিং - বিজীয় খণ্ড

```

9 = 0000 0000
1 = 0000 0001
2 = 0000 0010
3 = 0000 0011
4 = 0000 0100
5 = 0000 0101
6 = 0000 0110
7 = 0000 0111
8 = 0000 1000
9 = 0000 1001
10 = 0000 1010
11 = 0000 1011
12 = 0000 1100
13 = 0000 1101
14 = 0000 1110
15 = 0000 1111
16 = 0001 0000
  
```

তাহলে আমরা যদি কেবল শেষ বিটটি (মানে একবারে ডানদিকের বিটটি) 0 না 1 সেটি নির্ণয় করতে পারি, তাহলেই কেল্পা ফতে! এখন আমরা জানি যে 1 এর বাইনারি রূপের সবচেয়ে ডানদিকের বিট হচ্ছে 1, বাকি সব বিট 0। তাহলে 1 এর সঙ্গে যদি কোনো সংখ্যার বাইনারি অ্যান্ড (&) করা হয়, তাহলে ফলাফল কী দাঢ়াবে? ওই সংখ্যার একবারে ডানদিকের বিট বাদে, বাকি সবগুলো বিট হবে 0। আর একবারে ডানদিকের বিট যদি 1 হয়, তাহলে ফলাফলের ডানদিকের বিটও 1, নইলে 0। যেমন:

```

4 & 1 = 00000100 & 00000001 = 00000000, অর্থাৎ 0।
5 & 1 = 00000101 & 00000001 = 00000001, অর্থাৎ 1।
  
```

তাহলে তোমরা কি এখন প্রোগ্রাম লিখতে পারবে? তাড়াতাড়ি বইটি বন্ধ করে নিজে প্রোগ্রাম লেখার চেষ্টা কর। না পারলে (কিংবা তোমার প্রোগ্রাম লেখা শেষ হলে) আবার বইটি খুলে আমার নিচের কোড দেখো:

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int n;
6
7     scanf("%d", &n);
8
  
```

৯২

কম্পিউটার প্রোগ্রামিং - বিজীয় খণ্ড

```

9
10    if (n & 1) {
11        printf("%d is odd\n", n);
12    }
13    else {
14        printf("%d is even\n", n);
15    }
16
17 }
  
```

প্রোগ্রাম ৫-৬

কোডটি কম্পাইল করে রান কর এবং বিভিন্ন ইনপুট দিয়ে পরীক্ষা কর।
তোমরা তো আসকি (ASCII) সম্পর্কে ইতিমধ্যেই জানো। তোমরা হ্যাতো লক্ষ করেছ যে 'A' তোমরা তো আসকি (ASCII) সম্পর্কে ইতিমধ্যেই জানো। তোমরা হ্যাতো লক্ষ করেছ যে 'A' থেকে 'Z' পর্যন্ত অক্ষরগুলোর আসকি মান হচ্ছে, যথাক্রমে 65 থেকে 90। আবার 'a' থেকে 'z' পর্যন্ত অক্ষরগুলোর আসকি মান যথাক্রমে 97 থেকে 122। এখন তোমাদের মনে কি প্রশ্ন জেগেছে যে 91 থেকে 96 - এই ছয়টি মান কেন অন্য অক্ষরের জন্য ব্যবহার করা হলো? কেন জেগেছে যে 91 থেকে 96 - এই ছয়টি মান কেন অন্য অক্ষরের জন্য ব্যবহার করা হলো? কেন 'a' এর মান 91 হলো না? এর কিন্তু একটি কারণ রয়েছে 97 থেকে 65 বিয়োগ কর। বিয়োগফল 'a' এর মান 91 হলো না? এর কিন্তু একটি প্রতিটি অক্ষরের জন্য ছেট হাতের অক্ষর আর বড় 32। এভাবে তুম খেয়াল করে দেখো যে প্রতিটি অক্ষরের জন্য ছেট হাতের অক্ষর আর বড় 32। তাহলে আমরা কেবলমাত্র একটি বিট পরিবর্তন করে ছেট হাতের অক্ষর থেকে বড় হাতের অক্ষর কিংবা বড় হাতের অক্ষর থেকে ছেট হাতের অক্ষর - এই রূপান্তরের কাজটি করতে পারি। তোমরা কি নিজেরা সেটি পরীক্ষা করে দেখবে?

আমি তোমাদেরকে এখন একটি প্রোগ্রাম লিখে দেখাব। সেটি প্রথমে নিজেরা টাইপ করে কম্পাইল ও রান করবে।

```

1 #include <stdio.h>
2
3 char to_upper(char ch)
4 {
5     return ch & 95;
6 }
7
8 char to_lower(char ch)
9 {
10    return ch | 32;
11 }
12
13 int main()
14 {
  
```

৯৩

কম্পিউটার প্রোগ্রামিং - বিতীয় খণ্ড

```

15     str = "ABCdEFGhIJkLMnOpQRStUVWxYZ";
16     for (int i = 0; i < 26; i++) {
17         printf("Uppercase : %c, ", to_upper(str[i]));
18         printf("Lowercase : %c\n", to_lower(str[i]));
19     }
20     return 0;
21 }
```

প্রোগ্রাম ৫-৭

তোমাদের কাজ হচ্ছে ওপরের প্রোগ্রামটি আসলে কীভাবে কী কাজ করছে, সেটি বোঝার চেষ্টা কর। `to_upper()` ফাংশনে কেন 95 এর সঙ্গে বিটওয়াইজ অ্যান্ড (&) ব্যবহার করলাম আর `to_lower()` ফাংশনে কেন 32 এর সঙ্গে বিটওয়াইজ অর ব্যবহার করলাম? 95 আর 32 -এই সংখ্যাদ্বয়টি বা কীভাবে পেলাম? এসব প্রশ্নের উত্তর খুঁজে দেব কর।

এখন আমাদের পরবর্তী প্রোগ্রাম। কোনো সংখ্যা 2-এর পাওয়ার (ঘাত) কি না, সেটি বের করতে হবে। কোন সংখ্যাগুলো 2-এর পাওয়ার? 1, 2, 4, 8, 16, 32 ইত্যাদি। যাদের মনে প্রশ্ন জাগছে 1 কীভাবে তাদের মনে করিয়ে দিই, 2-এর পাওয়ার 0 হলে তার মান 1। তোমরা যদি এই সংখ্যাগুলোর বাইনারি রূপ লক্ষ কর, তাহলে দেখবে, এগুলোতে কেবল একটি বিট 1, বাকি বিটগুলো 0। আর যে বিটটি 1, সেটিকে 0 এবং তার ডানদিকের সবগুলো বিট 1 করে দিলে কী হয়? করে দেখি। যেমন 16 এর বাইনারি রূপ হচ্ছে **00010000**। এখানে যে 1 আছে, সেটিকে 0 আর 1-এর ডানদিকের সব 0 বিটগুলো 1 করে দিলে পাব **00001111**, যা কী না 15-এর বাইনারি রূপ। একই কাজ 8-এর বেলায় করলে আমরা 7 পাব, 4-এর বেলায় করলে পাব 3। তার মানে সংখ্যাটির চেয়ে এক ছোট, অর্ধাং সংখ্যাটি x হলে, $x-1$ । এখন এই x ও $x-1$ -এর মধ্যে বিটওয়াইজ অ্যান্ড অপারেশন করলে ফলাফল অবশ্যই শূন্য হবে। তাহলে আমরা এখন কমিশনাল লজিক প্রয়োগ করতে পারি। যদি x-এর মান শূন্য না হয় এবং $x \& (x-1)$ এর মান 0 হয়, তাহলে x হচ্ছে 2-এর পাওয়ার। এবারে প্রোগ্রাম লিখে ফেলা যাক। তোমাদের উচিত হবে আপে নিজেরা চেষ্টা করা, তারপরে আমার কোড দেখা। যারা নিজেরা চেষ্টা করবে না, তারা আসলে ঠুকে যাবে।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int n;
6
7     scanf("%d", &n);
8 }
```

৯৪

কম্পিউটার প্রোগ্রামিং - বিতীয় খণ্ড

```

9     if (n > 0 && (n & (n - 1)) == 0) {
10        printf("%d is a power of 2\n", n);
11    }
12    else {
13        printf("%d is NOT a power of 2\n", n);
14    }
15 }
16
17 }
```

প্রোগ্রাম ৫-৮

প্রোগ্রামটি রান করে বিভিন্ন সংখ্যা ইনপুট দিয়ে দেখো। আর হাঁ, আমি যেই কমিশনাল লিখেছি, সেটি চাইলে এভাবে লেখা যেত: `if (n && !(n & (n - 1)))`।

এবারে তোমাদের আরেকটি প্রোগ্রাম লিখতে হবে। কোনো ধনাত্মক পূর্ণসংখ্যার বাইনারি রূপে এবারে তোমাদের ক্যারেক্ট বিট 1? তোমরা যারা কম্পিউটার প্রোগ্রামিং ১ম খণ্ড বইটি পড়েছ, তোমরা সহজেই মোট কয়টি বিট 1? তোমরা যারা কম্পিউটার টাইপের অ্যারেতে নিয়ে যেতে পারবে, তারপরে সেই স্ট্রিংয়ে কতটি সংখ্যাকে একটি ক্যারেক্টোর টাইপের অ্যারেতে পারবে। তোমরা অথবে এই পদ্ধতিতে প্রোগ্রামটি লেখার চেষ্টা কর। '1' আছে, সেটি গান্ধা করতে পারবে। তারপরে আমরা অপারেশন ব্যবহার করে কাজটি করব।

তারপরে আমরা বিট অপারেশন ব্যবহার করে কাজটি করব।

কোনো সংখ্যাকে যদি 1-এর সঙ্গে বিটওয়াইজ অ্যান্ড করা হয়, তাহলে সংখ্যাটির সর্বডানের বিটটি 1 হলে ফলাফল হবে 1, নইলে ফলাফল 0। সেটি জোড়-বিজোড় বের করতে গিয়ে একটু আমেই দেখেছি। তাহলে আমরা সর্বডানের বিটটি পরীক্ষা করার পরে সংখ্যাটিকে যদি এক ঘর যাই শিফট করি, তাহলে কিন্তু ডানদিক থেকে বিতীয় সবগুলো বিট 0 হয়ে যাবে (কারণ একঘর ডানে শিফট করলে বামদিকের ফাঁকা ঘর 0 দিয়ে পূরণ হয়), তখন আমরা বলে দিতে পারব যে মোট কয়টি বিট 1। তাহলে প্রোগ্রাম লিখে ফেলা যাক:

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int n, count, i;
6
7     scanf("%d", &n);
8     count = 0;
9
10    while(n) {
11        if (n & 1) count++;
12    }
13 }
```

৯৫

কম্পিউটার প্রোগ্রামিং - বিটীয় খণ্ড

```

12   n = n >> 1;
13
14   printf("Number of 1: %d\n", count);
15
16   return 0;
17 }
18 }
```

প্রোগ্রাম ৫-৯

আচ্ছা, 1-এর সঙ্গে & করে আমরা সবচেয়ে ডানদিকের বিটটি 0 নাকি 1 সেটি ব্যবহাতে পারি। আমরা যদি ডানদিক থেকে দ্বিতীয় বিটটি কী, সেটি নির্ণয় করতে চাই, তাহলে আমরা কী করতে পারি? 1-কে একবার লেফট শিফট করে, তার সঙ্গে সংখ্যাটির & করলে আমরা ব্যবহাতে পারব ডানদিক থেকে দ্বিতীয় বিটটি 0 নাকি 1। এভাবে যদি n-তম বিট 0 নাকি 1, সেটি বের করতে চাই, তাহলে আমরা সংখ্যাটির সঙ্গে $(1 \ll n)$ -এর বিটওয়াইজ অ্যান্ড করতে পারি। তাহলে আমি নিচের প্রোগ্রামটি লিখতে পারি:

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int n = 32;
6     int i, count = 0, num;
7
8     scanf("%d", &num);
9
10    for (i = 0; i < n; i++) {
11        if (num & (1 << i)) count++;
12    }
13
14    printf("Number of 1: %d\n", count);
15
16    return 0;
17 }
```

প্রোগ্রাম ৫-১০

এখন আমি n-এর মান 32 লিখেছি। কারণ আমি জানি আমার কম্পিউটার ইন্টিজার হচ্ছে 32 বিট। কিন্তু বিষয়টি পৃথিবীর সব কম্পিউটারের জন্য সত্য নয়। তাই আমরা একটি কাজ করতে পারি। কম্পিউটারে ইন্টিজার কত বাইট, সেটি বের করে, এক বাইটের জন্য ওই কম্পিউটারে কত বিট বরাদ্দ, তার সঙ্গে গুণ করতে পারি। এভাবে : int n = sizeof(int) *

৯৬

কম্পিউটার প্রোগ্রামিং - বিটীয় খণ্ড

CHAR_BIT-এর মান বলা আছে Limits.h নামক হেডার প্যাকেজের মধ্যে। সক্ষ কর, CHAR_BIT-এর কিছু মজার ব্যবহার। দুটি সংখ্যা সমান কি না, সেটি আমরা বের করতে পারি। এখাবে xor এর কিছু মজার ব্যবহার। দুটি সংখ্যা সমান কি না, সেটি আমরা বের করতে পারি। নিচের প্রোগ্রামটি দেখো :

নিচের প্রোগ্রামটি দেখো :

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int n, m;
6     scanf("%d %d", &n, &m);
7
8     if (n ^ m) {
9         puts("Numbers are not equal\n");
10    } else {
11        puts("Numbers are equal\n");
12    }
13
14    return 0;
15 }
```

প্রোগ্রাম ৫-১১

দুটি সংখ্যা সমান হলে তাদের এক্সর করলে মান হবে 0। এখন তোমাকে যদি একটি আরে দেওয়া হয়, যেখানে একটি সংখ্যা কেবল একবার আছে, বাকি প্রতিটি সংখ্যা দুইবার করে আছ, দেওয়া হয়, যেখানে একটি সংখ্যা কেবল একবার আছে, সেটি কীভাবে বের করবে? অনেকভাবেই কাজটি করা তাহলে যে সংখ্যাটি কেবল একবার আছে, সেটি কীভাবে বের করবে? অনেকভাবেই কাজটি করা যাব। আমরা যদি সবগুলো সংখ্যা এক্সর করে দিই, তাহলে কিন্তু ফলাফল হবে যে সংখ্যাটি কেবল একবার আছে সেটি। কারণ যেগুলো দুবার করে আছে, সেগুলো এক্সর করে তো শূন্য হয়ে যাবে।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int ara[] = {1, 2, 3, 4, 1, 2, 3};
6     int result, i, n = 7;
7
8     result = ara[0];
9
10    for (i = 1; i < n; i++) {
11        result = result ^ ara[i];
12    }
13
14    printf("Result: %d\n", result);
15 }
```

৯৭

```
printf("result: %d\n", result);
return 0;
```

প্রোগ্রাম ৫-১২

এখন তোমাদের জন্য দুটি সমস্যা। সমস্যাগুলো একটি বাট্টন, বিশেষ করে দিতীয়টি। তোমরা এগুলো নিজে করার চেষ্টা করবে। সমাধান করতে না পারলে বেশি দুঃখিত হওয়ার কিছু নেই। সমাধানগুলো xor ব্যবহার না করেও করা যায়, তবে তোমাদের কাজ হচ্ছে এক্সর ব্যবহার করে সমাধান করা।

- 1) একটি অ্যারেতে 1 থেকে n পর্যন্ত সংখ্যাগুলোর মধ্যে একটি বাদে বাকি n-1টি সংখ্যা প্রতিটি একবার করে আছে। কোন সংখ্যাটি অ্যারেতে নেই সেটি বের করতে হবে।
- 2) একটি অ্যারেতে 1 থেকে n পর্যন্ত সংখ্যাগুলোর মধ্যে দুটি বাদে বাকি n-2টি সংখ্যা প্রতিটি একবার করে আছে। কোন দুটি সংখ্যা অ্যারেতে নেই সেটি বের করতে হবে।

অধ্যায় ৬ - স্ট্রাকচার (Structure) ও ইউনিয়ন (Union)

৬.১ - স্ট্রাকচার (Structure)

আমরা যারা ইতিমধ্যে মোটামুটি সি শিখে ফেলেছ, তার নিচয়ই বড় বড় প্রোগ্রাম লেখার জন্য আমরা যারা নিচের প্রয়োজন হচ্ছে। তোমরা যারা তোমাদের স্কুল-কলেজের জন্য বিভিন্ন রকম ব্যবহার করতে চাচ্ছ, তাদের নিচয়ই মনে হয়েছে যে, ইশ্য, যদি একটি ভ্যারিয়েবলের প্রোগ্রাম তৈরি করতে চাচ্ছ, তাদের নিচয়ে রাখা যেত, কত সুবিধাই না হত। আমরা যদি student মধ্যেই একজন ভ্যারিয়েবল তৈরি করতে পারতাম। এখন তোমাকে একটি কাজ করতে হবে। টাইপ একটি ভ্যারিয়েবল কিংবা চেয়ার মেখানেই থাক না কেন, সেখান থেকে উঠে একটি লাস্ট নিতে হবে। তুমি বিছানা কিংবা চেয়ার মেখানেই থাক না কেন, সেখান থেকে উঠে একটি টাইপ তৈরি করতে পারবে। কী আবস্দ!

আমরা খুব সহজ একটি উদাহরণ দিয়ে শুরু করব। ধৰা যাক, একজন শিক্ষার্থীর দুটি তথ্য আমাদের কাছে গুরুত্বপূর্ণ, তার আইডি ও নাম। এখন সি প্রোগ্রামিং ভাষায় স্ট্রাকচার নামে একবারের জিমিস আছে, যা দিয়ে এমন ডেটা টাইপ তৈরি করা সম্ভব যার মধ্যে একধরিক অংশ থাকবে। আমরা যদি চাই যে student নামে একটি স্ট্রাকচার তৈরি করব, যার মধ্যে আবার id ও name নামে দুটি ডেটা টাইপ থাকবে (যাদেরকে আমরা সদস্য বা মেম্বার বলতে পারি), তাহলে আমরা এভাবে লিখব:

```
1 struct student {
2     int id;
3     char name[40];
4 };
```

এখনে আমি ধরে নিয়েছি যে আইডি হচ্ছে পূর্ণসংখ্যা, আর নাম হচ্ছে 40 ক্যারেক্টারের আয়োজন। তোমাদের অনেকের স্কুল-কলেজের আইডি কিন্তু পূর্ণসংখ্যা না হয়ে স্ট্রিংও হতে পারে।

এখন স্ট্রাকচারটির ভ্যারিয়েবল ডিক্রিয়ার করব কীভাবে? যদি std_one নামে একটি ভ্যারিয়েবল ডিক্রিয়া করতে চাই, তাহলে আমরা লিখব **struct student std_one;**। তাহলে std_one নামে একটি স্ট্রাকচার ভ্যারিয়েবল তৈরি হয়ে গেল, যার মধ্যে id ও name নামে দুটি সদস্য রয়েছে। সেগুলো আবার একসেস করব কীভাবে? স্ট্রাকচার ভ্যারিয়েবলের পরে একটি ডট(.) দিয়ে সদস্য ভ্যারিয়েবলের নাম লিখতে হবে। যেমন std_one.id।

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড
তাহলে চলো একটি পুরো প্রোগ্রাম লিখে কম্পাইল ও রান করে দেখি কী হয়।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     struct student {
6         int id;
7         char name[40];
8     };
9
10    struct student one;
11
12    one.id = 1;
13    one.name = "Tahmid Rafi";
14
15    printf("ID: %d\n", one.id);
16    printf("Name: %s\n", one.name);
17
18    return 0;
19 }
```

প্রোগ্রাম ৬-১

প্রোগ্রামটি রান করলে কী আউটপুট দেখতে পাচ্ছ?

error: array type 'char [40]' is not assignable

ক্যারেক্টোরের অ্যারের মধ্যে অ্যাসাইনমেন্ট অপারেশন চালানো যায় না, আর আমরা সেই ভুল করেছি `one.name = "Tahmid Rafi";`, লিখে। এর দুটি সমাধান আছে।

প্রথম সমাধান হচ্ছে ক্যারেক্টোরের অ্যারের বদলে ক্যারেক্টোর পয়েন্টার ব্যবহার করা। আর দ্বিতীয় সমাধান হচ্ছে অ্যাসাইনমেন্টের বদলে `strcpy` ফাংশন ব্যবহার করে কাজ করা।

প্রথম সমাধান:

```

1 #include <stdio.h>
2
3 int main()
4 {
5     struct student {
6         int id;
7     };
8 }
```

১০০

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

```

1     char* name;
2
3 struct student one;
4
5     one.id = 1;
6     one.name = "Tahmid Rafi";
7
8     printf("ID: %d\n", one.id);
9     printf("Name: %s\n", one.name);
10
11    return 0;
12 }
```

প্রোগ্রাম ৬-২

দ্বিতীয় সমাধান:

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     struct student {
7         int id;
8         char name[40];
9     };
10
11    struct student one;
12
13    one.id = 1;
14    strcpy(one.name, "Tahmid Rafi");
15
16    printf("ID: %d\n", one.id);
17    printf("Name: %s\n", one.name);
18
19    return 0;
20 }
```

প্রোগ্রাম ৬-৩

তোমরা দুটি প্রোগ্রামই পুরোপুরি টাইপ করবে (শুরু থেকে), একটি প্রোগ্রামের মধ্যে এডিট করে আরেকটি লিখবে না। আমি এই ব্যাপারে জোর দিই, কারণ এতে প্রোগ্রামিং শেখা একটু পেক্ষে হয়। উভয় প্রোগ্রামের ক্ষেত্রেই রান করলে দেখবে যে নিচের আউটপুট আসছে:

১০১

ID: 1
Name: Tahmid Rafi

এখন আমরা চাইলে স্ট্রাকচার ভ্যারিয়েবলের মধ্যে ইনপুটও নিতে পারি :

```

1 #include <stdio.h>
2 #include <string.h>
3
4 struct student {
5     int id;
6     char name[40];
7 };
8
9 int main()
10 {
11     struct student one;
12
13     scanf("%d", &one.id);
14     scanf("%s", one.name);
15
16     printf("ID: %d\n", one.id);
17     printf("Name: %s\n", one.name);
18
19     return 0;
20 }
```

প্রোগ্রাম ৬-৪

প্রোগ্রামের আউটপুট দেখবে যে নামের কেবল প্রথম অংশ প্রিন্ট হচ্ছে। তার কারণ হচ্ছে `scanf` ফাংশনের ভেতরের `%s`। তোমরা যদি এভাবে লিখ `scanf(" %[^\n]", one.name);`, তাহলে ঠিকঠাক ইনপুট নেবে।

এখন আমরা আরেকটি কাজ করতে চাই। যেহেতু আমাদের বেশিরভাগেরই নামের দুটি অংশ, তাই আমরা সেই দুটি অংশ আলাদা করে ফেলতে চাই। তাহলে আমরা নতুন একটি স্ট্রাকচার তৈরি করতে পারি, যার দুটি সদস্য থাকবে:

```

1 struct nametype {
2     char first[20];
3     char last[20];
4 };
```

তাহলে `student` স্ট্রাকচারটি ও একটু পরিবর্তন করতে হবে:

১০২

```

1 struct student {
2     int id;
3     struct nametype name;
4 };
```

নম্বা, আমরা কীভাবে একটি স্ট্রাকচারের ভেতরে আরেকটি স্ট্রাকচার ভ্যারিয়েবল ব্যবহার করছি। তোমরা চাইলে এখন আরেকবার লাফ দিতে পারো।
এখন আমরা যদি `one` নামে কোনো `student` স্ট্রাকচারের ভ্যারিয়েবল তৈরি করি। তার নামের প্রথম অংশ এভাবে একসেস করতে পারব : `one.name.first`।
নিচে প্রোগ্রামটি আগে মনোযোগ দিয়ে দেখো, তারপরে টাইপ করে কম্পাইল ও রান কর:

```

1 #include <stdio.h>
2 #include <string.h>
3
4 struct nametype {
5     char first[20];
6     char last[20];
7 };
8
9 struct student {
10     int id;
11     struct nametype name;
12 };
13
14 main()
15 {
16     struct student one;
17
18     scanf("%d", &one.id);
19     scanf("%s", one.name.first);
20     scanf("%s", one.name.last);
21
22     printf("ID: %d\n", one.id);
23     printf("Name: %s %s\n", one.name.first, one.name.last);
24
25     return 0;
26 }
```

প্রোগ্রাম ৬-৫

১০৩

কম্পিউটার প্রোগ্রামিং - বিতীয় খণ্ড

আমি রান করে যে ইনপুট দিলাম আর আউটপুট পেলাম, সেটি এখানে দিয়ে দিচ্ছি, তুমি শেষের ইচ্ছামতো ইনপুট দিতে পারো।

10
Tahmid
Rafi
ID: 10
Name: Tahmid Rafi

এখন কথা হচ্ছে এই একজন শিক্ষার্থীর তথ্য দিয়ে তো আমাদের চলবে না। আমাদের অনেক শিক্ষার্থীর ডেটা নিয়ে কাজ করতে হবে। সেটি আমরা কীভাবে করব? খুবই সহজ, স্ট্রাকচারের অ্যারে তৈরি করে। উদাহরণ হিসেবে তোমরা নিচের প্রোগ্রামটি চটপট টাইপ করে কম্পাইল ও রান করে ফেল, তারপরে মনোযোগ দিয়ে কোড লক্ষ কর :

```
1 #include <stdio.h>
2 #include <string.h>
3
4 struct nametype {
5     char first[20];
6     char last[20];
7 };
8
9 struct studenttype {
10    int id;
11    struct nametype name;
12 };
13
14 int main()
15 {
16    struct studenttype student[5];
17    int i, n = 5;
18
19    for (i = 0; i < n; i++) {
20        printf("Enter the ID for student %d: ", i+1);
21        scanf("%d", &student[i].id);
22        printf("Enter the first name for student %d: ", i+1);
23        scanf("%s", student[i].name.first);
24        printf("Enter the last name for student %d: ", i+1);
25
26        scanf("%s", student[i].name.last);
27        printf("\n");
28    }
}
```

কম্পিউটার প্রোগ্রামিং - বিতীয় খণ্ড

```
39     printf("Output: \n\n");
40
41     for (i = 0; i < n; i++) {
42         printf("ID: %d\n", student[i].id);
43         printf("Name: %s %s\n", student[i].name.first,
44                student[i].name.last);
45     }
46
47     return 0;
48 }
```

প্রোগ্রাম ৬-৬

সিঙ্কে **typedef** নামে একটি কিওয়ার্ড (keyword) আছে, যেটি ব্যবহার করে আমরা আরেকটু সহজে বিভিন্ন ডেটা টাইপ তৈরি করতে পারি। যেমন আমরা লিখতে পারি :

```
1 typedef struct {
2     char first[20];
3     char last[20];
4 } nametype;
```

অহল আমাদের সুবিধা হবে যে, এরপর আমাদেরকে **struct nametype name;** এভাবে ভারিয়েল ডিক্লেয়ার করতে হবে না, আমরা সরাসরি **nametype name;** লিখে ভারিয়েল ডিক্লেয়ার করতে পারব।

এখন কেবল আইডি আর নাম দিয়ে আমাদের চলবে না। আমরা শিক্ষার্থীদের প্রেত নিয়েও হিসাব-নিকাশ করতে চাই, বা তার পরীক্ষার গ্রেডও জানতে চাই। তাহলে আমরা কী করব? খুবই সহজ, আমরা **studenttype** স্ট্রাকচারটিতে **grade** নামে একটি ক্যারেক্টর অ্যারে যোগ করে দিব।

```
1 typedef struct {
2     int id;
3     nametype name;
4     char grade[3];
5 } studenttype;
```

এখন চলো একটি পুরো প্রোগ্রাম লিখে দেখি। এই প্রোগ্রামে আমরা শিক্ষার্থীদের পরীক্ষার নম্বর থেকে গ্রেড নির্ণয়ের চেষ্টা করব। গ্রেড নির্ণয়ের কাজটি আলাদা একটি ফাংশন লিখে করব।

```

1 #include <stdio.h>
2 #include <string.h>
3
4 typedef struct {
5     char first[20];
6     char last[20];
7 } nametype;
8
9 typedef struct {
10    int id;
11    nametype name;
12    char grade[3];
13 } studenttype;
14
15 void calculate_grade(studenttype s, int mark)
16 {
17     if (mark >= 80) {
18         strcpy(s.grade, "A+");
19     }
20     else if (mark >= 70) {
21         strcpy(s.grade, "A");
22     }
23     else if (mark >= 60) {
24         strcpy(s.grade, "A-");
25     }
26     else if (mark >= 50) {
27         strcpy(s.grade, "B");
28     }
29     else if (mark >= 40) {
30         strcpy(s.grade, "C");
31     }
32     else {
33         strcpy(s.grade, "F");
34     }
35 }
36
37 int main()
38 {
39     studenttype student[3];
40     int i, n = 3;
41     int marks[] = {72, 82, 60};
42
43     for (i = 0; i < n; i++) {
44         printf("Enter the ID for student %d: ", i+1);

```

```

45         scanf("%d", &student[i].id);
46         printf("Enter the first name for student %d: ", i+1);
47         scanf("%s", student[i].name.first);
48         printf("Enter the last name for student %d: ", i+1);
49         scanf("%s", student[i].name.last);
50         strcpy(student[i].grade, "");
51     }
52     for (i = 0; i < n; i++) {
53         calculate_grade(student[i], marks[i]);
54     }
55
56     printf("Output: \n\n");
57
58     for (i = 0; i < n; i++) {
59         printf("ID: %d\n", student[i].id);
60         printf("Name: %s %s\n", student[i].name.first,
61               student[i].name.last);
62         printf("Grade: %s\n", student[i].grade);
63     }
64
65     return 0;
66 }
67 }
```

তোমরা যদি প্রোগ্রামটি ঠিকঠাক টাইপ করে কম্পাইল ও রান কর, তাহলে দেখবে আউপুটে
শিক্ষার্থীদের গ্রেড দেখাচ্ছে না। তোমরা কেউ কি বলতে পারবে এর কারণ কী?
তোমরা যদি প্রোগ্রামটি ফাংশনের ভেতরে যেই প্যারামিটার
শিক্ষার্থীদের গ্রেড দেখাচ্ছে না। তোমরা কেউ কি বলতে পারবে এর কারণ কী?
তোমরা যদি প্রোগ্রামটি ফাংশনের ভেতরে যেই প্যারামিটারে
গ্রেড না দেখানোর কারণ হচ্ছে, আমরা calculate_grade ফাংশনের ভেতরে যেই প্যারামিটার
পাঠিয়েছি, সেটি ওই ফাংশনের আর্গুমেন্টে কপি হচ্ছে আর ফাংশন থেকে রিটার্ন করার পরে সেটি
হারিয়ে যাচ্ছে। অর্থাৎ, আমরা যখন calculate_grade(student[i], marks[i]) কল করছি,
তখন প্রোগ্রাম calculate_grade(studenttype s, int mark)-এ তুকছে আর সেসময়
সেটি পাঠিয়েছে সেটি এবং সেটি এর মান কপি হচ্ছে s-এ আর marks[i] কপি হচ্ছে mark ভারিয়েবলে। এই s ও
student[i]-এর মান কপি হচ্ছে s-এ আর marks[i] কপি হচ্ছে mark ভারিয়েবলে। আর
mark কিন্তু দুটি নতুন ভারিয়েবল, প্রথমটি studenttype ও দ্বিতীয়টি int টাইপের। আর
এগুলো calculate_grade ফাংশনের লোকাল ভারিয়েবল হিসেবে বিবেচনা করা যায়।
ফাংশনের ভেতরে এগুলোর কোনো পরিবর্তন করলে ফাংশনের বাইরে গেলে (মানে ফাংশন
থেকে রিটার্ন করলে কিংবা অন্য কোনো ফাংশন কল করে সেই ফাংশনের ভেতরে তুকে গেলে)
আর সেগুলো খুঁজে পাওয়া যাবে না। তো এভাবে ফাংশন কল করার পদ্ধতিকে বলে, কল বাই
ভ্যালু (call by value)। এখন আমরা পয়েন্টারের শরণাপন্ন হতে পারি। ফাংশনের ভেতরে যদি

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

সরাসরি ভ্যারিয়েবল না পাঠিয়ে, এই ভ্যারিয়েবলের ঠিকানা পাঠানো হয়, তাহলে সেই ঠিকানার ভ্যারিয়েবলে কোনো পরিবর্তন করলে আমরা ফাংশনের বাইরে গেলেও সেই পরিবর্তন অন্তর্ভুক্ত থাকবে। এটিকে বলে, কল বাই রেফারেন্স (call by reference)। কল বাই ভ্যালু, কল বাই রেফারেন্স এই কথাগুলো মনে রাখা দরকার কারণ পরীক্ষায় কিংবা চাকরির ইন্টারভিউতে আমরা এমন প্রশ্ন জিজ্ঞাসা করে। এখন চলো, একটি ছেট প্রোগ্রাম লিখে পরীক্ষা করে দেখি।

```

1 #include <stdio.h>
2
3 void plus_ten1(int a)
4 {
5     a = a + 10;
6 }
7
8 void plus_ten2(int* a)
9 {
10    *a = (*a) + 10;
11 }
12
13 int main()
14 {
15     int a;
16
17     a = 5;
18
19     plus_ten1(a);
20
21     printf("a = %d\n", a);
22
23     plus_ten2(&a);
24
25     printf("a = %d\n", a);
26
27     return 0;
28 }
```

প্রোগ্রাম ৬-৮

ওপরের প্রোগ্রামটি চালালে তোমরা আউটপুট দেখবে এমন:

```
a = 5
a = 15
```

১০৮

অর্থাৎ প্রথম ফাংশনের ভেতরে ভ্যারিয়েবলের সঙ্গে 10 যোগ করলেও ফাংশনটি রিটুর্ন করার সময় সেই ভ্যারিয়েবল হারিয়ে পেছে, আর মেইন ফাংশনের a-এর মানেরও কোনো পরিবর্তন হ্যায়ি নেই। কিন্তু দ্বিতীয় ক্ষেত্রে আমরা a-এর অ্যাড্রেস পাঠিয়ে দিয়েছি। আর সেটা প্রথম স্লেল সেটি হারিয়ে দিয়ে। এখন সেই পয়েন্টার যাকে পয়েন্ট করে, তার হেকোনো পরিবর্তন নেয়েছে তাই মেইন ফাংশনে ফেরত আসার পরেও সেটি অঙ্কুণ্ডি থাকবে।

এখন আমরা স্ট্রাকচার ভ্যারিয়েবলকে পয়েন্ট করবে। আমরা যদি studenttype-এর পয়েন্টার তৈরি করতে স্ট্রাকচার ভ্যারিয়েবলকে পয়েন্ট করবে : `studenttype * p;`। আর তার সদস্য যেমন id-কে জাই, সেটি লিখতে হবে এভাবে : `p->id`। নিচে আমি পুরো প্রোগ্রাম লিখে দিলাম। তোমরা একসেস করতে হলে, লিখতে হবে `p->id`। নিচে আমি পুরো প্রোগ্রাম লিখে দিলাম। তোমরা প্রথম প্রোগ্রামটি টাইপ করবে, তারপরে শেয়াল করবে যে, ফাংশনটি কীভাবে কল করা হচ্ছে, ফাংশনের আর্টিমেন্ট কীভাবে লেখা হচ্ছে, আর স্ট্র্যাকচারের সদস্যগুলো কীভাবে একসেস করা হচ্ছে।

```

1 #include <stdio.h>
2 #include <string.h>
3
4 typedef struct {
5     char first[20];
6     char last[20];
7 } nametype;
8
9 typedef struct {
10    int id;
11    nametype name;
12    char grade[3];
13 } studenttype;
14
15 void calculate_grade(studenttype* s, int mark)
16 {
17    if (mark >= 80) {
18        strcpy(s->grade, "A+");
19    }
20    else if (mark >= 70) {
21        strcpy(s->grade, "A");
22    }
23    else if (mark >= 60) {
24        strcpy(s->grade, "A-");
25    }
}
```

১০৯

```

26     else if (mark >= 50) {
27         strcpy(s->grade, "B");
28     }
29     else if (mark >= 40) {
30         strcpy(s->grade, "C");
31     }
32     else {
33         strcpy(s->grade, "F");
34     }
35 }
36
37 int main()
38 {
39     studenttype student[5];
40     int i, n = 5;
41     int marks[] = {72, 82, 60, 20, 50};
42
43     for (i = 0; i < n; i++) {
44         printf("Enter the ID for student %d: ", i+1);
45         scanf("%d", &student[i].id);
46         printf("Enter the first name for student %d: ", i+1);
47         scanf("%s", student[i].name.first);
48         printf("Enter the last name for student %d: ", i+1);
49         scanf("%s", student[i].name.last);
50         strcpy(student[i].grade, "");
51     }
52
53     for (i = 0; i < n; i++) {
54         calculate_grade(&student[i], marks[i]);
55     }
56
57     printf("Output: \n\n");
58
59     for (i = 0; i < n; i++) {
60         printf("ID: %d\n", student[i].id);
61         printf("Name: %s %s\n", student[i].name.first,
62               student[i].name.last);
63         printf("Grade: %s\n", student[i].grade);
64     }
65
66 }
67

```

কম্পিউটার প্রোগ্রামিং - বিভিন্ন খণ্ড
ওপরের প্রোগ্রামটি তুমি পুরোপুরি সুবেঁধে থাকলে এবং চিক্কাক টাইপ করে কম্পাইল ও রান করতে
পারলে তুমি এক ধাপ এগিয়ে গেলে। স্ট্রাকচার ও ফাইল ব্যবহার করে তুমি এখন চয়কার সব
প্রোগ্রাম তৈরি করতে পারবে। এখন তোমাদের জন্য কাজ।

- 1) যেসব সংখ্যাকে p/q আকারে প্রকাশ করা যায়, তাদেরকে বলে মূলদ সংখ্যা (irrational number)। যেমন, $\frac{1}{2}$, $\frac{3}{4}$ ইত্যাদি। এখন তোমাদেরকে আমি একটি স্ট্রাকচার
তৈরি করে দিচ্ছি:

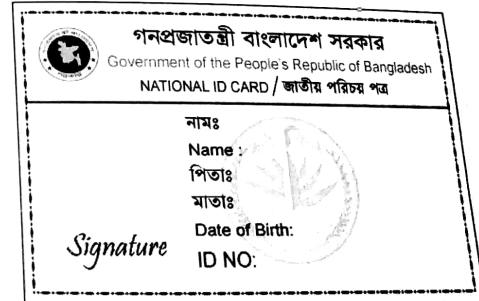
```

1 typedef struct {
2     int p;
3     int q;
4 } Rational;

```

তোমাদের কাজ হবে, দুটি মূলদ সংখ্যার যোগ, বিয়োগ, গুণ, ভাগ এসব অপারেশনের জন্য
আলাদা ফাংশন লিখা। দুটি মূলদ সংখ্যা সমান কি না, সেটি পরীক্ষা করার জন্যও একটি
ফাংশন লিখা। ভগ্নাংশের যোগ-বিয়োগ-গুণ-ভাগ করার পদ্ধতি জানা থাকলে এবং ইউক্লিডের
গসাণ নির্ণয়ের প্রক্রিয়া জানা থাকলে ফাংশনগুলো লিখতে তোমাদের তেমন সমস্যা হবে না।

- 2) তুমি নিশ্চয়ই তোমার বাবা-মা কিংবা বড় কারও কাছে আমাদের ন্যাশনাল আইডি কার্ড বা
জাতীয় পরিচয়পত্র দেখেছ? সেটি দেখতে ছবি ১৬ এর মতো:



ছবি ১৬: বাংলাদেশের জাতীয় পরিচয়পত্র শৃঙ্খলা

এখন এই পরিচয়পত্রটি বাংলাদেশের একজন মানুষের পূর্ণাঙ্গ পরিচয় বহন করে। তোমার কাজ হবে এই পরিচয়পত্রটির জন্য একটি স্ট্রাকচার তৈরি করা। আর সেই স্ট্রাকচার করতে গেলে তোমার বেশ কিছু ছেট ছেট স্ট্রাকচার তৈরি করে। আর সেই স্ট্রাকচার তৈরি করতে হবে। তোমার DATE একটি স্ট্রাকচার হতে পারে যার তিনটি অংশ থাকবে - দিন, মাস ও বছর। স্ট্রাকচার তৈরি শেষ হলে কোনো বস্তুকে দিয়ে পরীক্ষা করিয়ে নিও।

৬.২ – ইউনিয়ন (Union)

অনেক বইতেই স্ট্রাকচারের সঙ্গে আরেকটি বিষয় আলোচনা করা হয়। সেটি হচ্ছে ইউনিয়ন (Union)। এটি অনেকটি স্ট্রাকচারের মতোই তবে স্ট্রাকচারের সঙ্গে পার্থক্য হচ্ছে, ইউনিয়ন সদস্যগুলো একই মেমোরি শেয়ার করে। উদাহরণ দিয়ে বিষয়টি পরিষ্কার করছি। নিচের

```

1 #include <stdio.h>
2
3 struct s {
4     char ch;
5     int n;
6     char str[16];
7 } sv;
8
9 union u {
10     char ch;
11     int n;
12     char str[16];
13 } uv;
14
15 int main()
16 {
17     int struct_size, union_size;
18     struct_size = sizeof(sv);
19     union_size = sizeof(uv);
20
21     printf("Structure variable took %d bytes\n",
22         struct_size);
23     printf("Union variable took %d bytes\n", union_size);
24 }
```

```

25 }
26 }
```

return 0;

প্রোগ্রাম ৬-১০

আমার কম্পিউটারে আউটপুট আসে এমন:

structure variable took 24 bytes
Union variable took 16 bytes

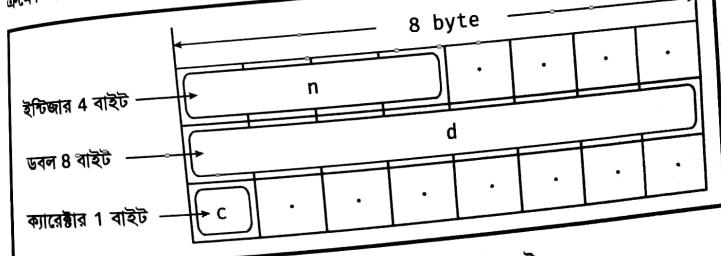
স্ট্রাকচার ভ্যারিয়েবল কীভাবে 24 বাইট জায়গা নিল, সে আলোচনায় পরে আসছি। ইউনিয়ন ভ্যারিয়েবল 16 বাইট জায়গা নিয়েছে, তার কারণ হচ্ছে এই ইউনিয়নের তিনটি সদস্য একই মেমোরি শেয়ার করে, তাই তাদের মধ্যে সবচেয়ে বড় যে, 16 বাইটের স্ট্রিং, তার জন্য 16 বাইট মেমোরি নিয়েছে। এখন কোনো ইউনিয়ন ভ্যারিয়েবল, তার যেকোনো একটি সদস্য নিয়ে কাজ জায়গা নিয়েছে। এখন কোনো ইউনিয়ন ভ্যারিয়েবল, তার যেকোনো একটি সদস্য নিয়ে কাজ করবে, একাধিক সদস্য নিয়ে কাজ করতে গেলে মেমোরিতে গওগোল লেগে যাবে। উদাহরণ দিচ্ছি:

```

1 #include <stdio.h>
2
3 union u {
4     char ch;
5     int x;
6     int y;
7 } uv;
8
9 int main()
10 {
11     uv.ch = 'A';
12     printf("uv.ch = %c\n", uv.ch);
13
14     uv.x = 17;
15     printf("uv.x = %d\n", uv.x);
16     printf("uv.ch = %c\n", uv.ch);
17
18     uv.y = 18;
19     printf("uv.y = %d\n", uv.y);
20
21     printf("uv.ch = %c\n", uv.ch);
22     printf("uv.x = %d\n", uv.x);
23 }
```

আউটপুট হবে এমন:

char size : 1 bytes
int size : 4 bytes
double size : 8 bytes
s1 size : 24 bytes
s2 size : 16 bytes



ছবি ১৭: স্ট্রাকচার s1 এর মেমোরি অ্যালাইনমেন্ট

আমার কম্পিউটারের কম্পাইলার (আমার কম্পিউটারের কথা বিশেষভাবে বলছি কারণ তোমার আমার কম্পিউটারের কম্পাইলার ক্ষেত্রে অন্যভাবেও কাজ করতে পারে) কীভাবে কম্পাইলার আলাদা হতে পারে এবং সেটি এক্ষেত্রে অন্যভাবেও কাজ করতে পারে (s1-এর ক্ষেত্রে) n-ডিক্রেয়ার করা হয়, সে ক্রমেই মেমোরিতে জায়গা দখল করে। প্রথম ক্ষেত্রে (s1-এর ক্ষেত্রে) n-এর জন্য চার বাইট জায়গা নিবে। পরের ভারিয়েবলটি হচ্ছে ডবল, অর্থাৎ আট বাইট। তাই আগের চার বাইটের পরে সে আরো চার বাইট জায়গা দখল করবে, মোট আট বাইট বানানোর

এখনে যখনই আমি x-এর মান অ্যাসাইন করেছি, ch ও y-তে কিছু থাকতে সেগুলো আর থাকবে uv.ch এঙ্গেলোর মান আমরা আর পাব না। তোমরা যদি বিভিন্ন ডিভাইসের জন্য এমনক্ষেত্রে প্রোগ্রামিং (embedded programming) কর, তাহলে হয়তো তোমরা আবারও ইউনিয়নের দেখা পাবে। নিলে তোমাদের ভবিষ্যতে ইউনিয়ন ব্যবহার করার সম্ভাবনা খুবই কম।

৬.৩ - স্ট্রাকচারের মেমোরি অ্যালাইনমেন্ট

এখন আমরা স্ট্রাকচারের মেমোরি নিয়ে আলোচনা করব। সবচেয়ে সহজ উপায় হচ্ছে প্রোগ্রাম লিখে আউটপুট দেখা ও তা বোঝার চেষ্টা করা :

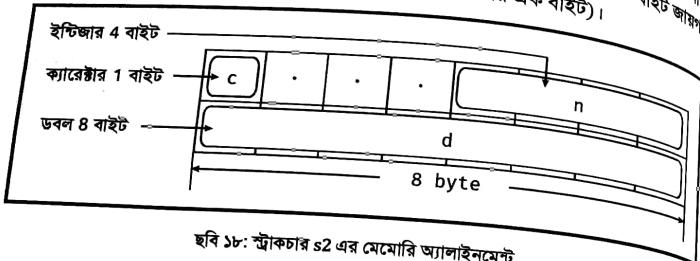
```

1 #include <stdio.h>
2
3 struct s1 {
4     int n;
5     double d;
6     char c;
7 };
8
9 struct s2 {
10    char c;
11    int n;
12    double d;
13 };
14
15 int main()
16 {
17
18     printf("char size : %lu bytes\n", sizeof(char));
19     printf("int size : %lu bytes\n", sizeof(int));
20     printf("double size : %lu bytes\n", sizeof(double));
21
22     printf("s1 size : %lu bytes\n", sizeof(struct s1));
23     printf("s2 size : %lu bytes\n", sizeof(struct s2));

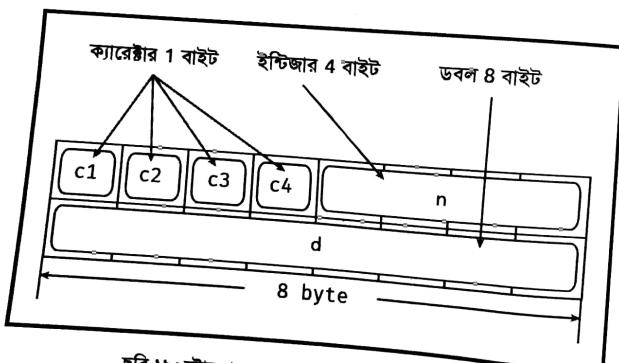
```

কম্পিউটার প্রোগ্রামিং - বিতাম খণ্ড

জন্য, আর তারপরে ডবল ভ্যারিয়েবলটি আট বাইটে রাখবে। তারপরে ক্যারেক্টার ভ্যারিয়েবলটি এক বাইটে রাখবে। আর মেমোরির অ্যালাইনমেন্ট ঠিক রাখার জন্য সে আরও সাত বাইট জায়গা দখল করবে। এভাবে মোট 24 বাইট। ছবি ১৭ দেখো (প্রতিটি ঘর এক বাইট)।



বিতায় ক্ষেত্রে (s2-এর ক্ষেত্রে), প্রথমে সে এক বাইট জায়গা নেবে, ক্যারেক্টার টাইপের ভ্যারিয়েবলের জন্য। তারপরে যখন চার বাইট ইন্টিজার পাবে, তখন ক্যারেক্টারের জন্য দখল করা এক বাইটের পরে আরও তিনটি বাইট দখল করবে। একে প্যাডিং (padding) বলা হয়। তারপরে ইন্টিজারের জন্য চার বাইট দখল করবে, মোট আট বাইট হলো। তারপরে ডবলের জ্যা বাইট। আবার আট বাইট দখল করবে। এভাবে মোট 16 বাইট। ছবি ১৮ দেখো (প্রতিটি ঘর এক বাইট)।



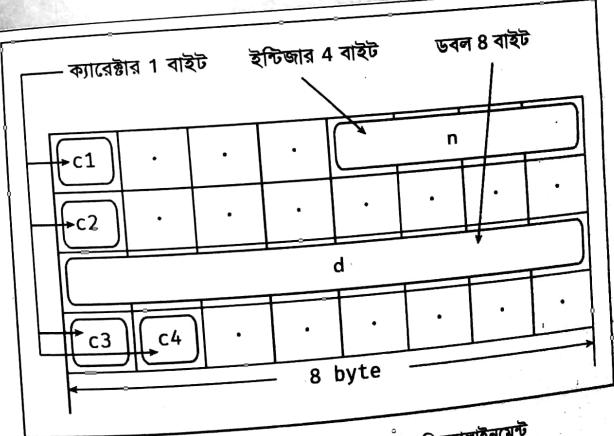
আমরা যদি s2-তে আরও তিনটি ক্যারেক্টার শুরুতে যোগ করতাম, তাহলেও মোট 16 বাইট জায়গা খরচ হতো।

কম্পিউটার প্রোগ্রামিং - বিতাম খণ্ড
প্রশ্ন: নিচের স্ট্রাকচারের একটি ভ্যারিয়েবলও 16 বাইট জায়গা নেবে (ছবি ১৯)?

```
1 struct s2 {
2     char c1;
3     char c2;
4     char c3;
5     char c4;
6     int n;
7     double d;
8 }
```

আবার এটিকে যদি নিচের মতো করে ডিক্লেয়ার করি, তাহলে কত বাইট জায়গা নেবে (ছবি ২০)?

```
1 struct s2 {
2     char c1;
3     int n;
4     char c2;
5     double d;
6     char c3;
7     char c4;
8 }
```



অধ্যায় ৭ - আরও পয়েন্টার

তোমাদের যদি এই স্ট্রাকচারের মেমোরির হিসাব-নিকাশ ব্রাতে সমস্যা হয়, তাহলে আরেকবার পড়, এবং তারপরেও সমস্যা হলে চিন্তিত হয়ে না। কেবল দুটি জিনিস মনে রাখবে :

এক : কেউ স্ট্রাকচারের সাইজ জিজ্ঞাসা করলে ছট করে উত্তর দেবে না, হিসাব-নিকাশ করে উত্তর দেবে।

দুই : স্ট্রাকচারে মেমোরির ভ্যারিয়েবলগুলো সবসময় সাইজ অনুসারে ছোট থেকে বড় করে ডিক্রেশার করবে। তাহলে কম মেমোরি খরচ হবে।

কম্পিউটার প্রোগ্রামিং ১ম খণ্ড বইতে আমরা আরে ব্যবহার করে প্রোগ্রাম তৈরি করেছিলাম, যেটি দিয়ে একটি ফ্লাসের শিক্ষার্থীদের পরীক্ষার ফলাফল বের করা যায়। আমরা প্রোগ্রাম তৈরি করার আগেই জনতাম যে, ফ্লাসে মেট ৪০ জন শিক্ষার্থী আছে। তাই আরের সাইজ বের করতে কোনো সমস্যাই হ্যানি। এখন, তোমরা যদি আমাদের জাতীয় বিশ্ববিদ্যালয়ের পরীক্ষার ফলাফল নির্ণয় করার জন্য একটি প্রোগ্রাম লিখ, সেখানে আরে ব্যবহার করলে তার সাইজ কত হবে? এক লাখ, দুই লাখ, দশ লাখ? সঠিক সংখ্যাটি কিন্তু তোমরা জানো না। আবার একেক বছর একেক সংখ্যাক দুই লাখ, দশ লাখ? সঠিক সংখ্যাটি কিন্তু পরীক্ষা দেয়। তাই বলে তোমরা যদি এখন আরের সাইজ শিক্ষার্থী জাতীয় বিশ্ববিদ্যালয়ের ভর্তি পরীক্ষা দেয়। তাই বলে তোমরা যদি এখন আরের সাইজ দশ মেটি দিয়ে রাখে, তাহলে সেটি তো কোনো কাজের কথা হলো না। আমরা সফটওয়্যার তৈরির সময় অথবা কম্পিউটারের মেমোরি কেন খরচ করব? এই অপয়োজনীয় মেমোরি খরচ করা বঙ্গ করার জন্য সি প্রোগ্রামিং ভাষায় একটি চমৎকার উপায় আছে!

আমরা এখন শিখব ডায়ানামিক মেমোরি আলোকেশন (dynamic memory allocation)। যা ব্যবহার করলে, আমাদের আগে থেকে মেমোরির সাইজ নির্ধারণ করে দিতে হবে না। প্রোগ্রামটি জ্বালার সময়ই আমরা দরকার মতো মেমোরি নিতে পারব। এজন্য পয়েন্টার ব্যবহার করে আমাদের মেমোরি বরাদ্দ বা আলোকেশন (allocation) করতে হবে। চলো, একটি প্রোগ্রাম লিখে কম্পাইল ও রান করে দেখি। তারপরে সেটি নিয়ে আলোচনা করা যাবে।

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int *marks;
7     int i, n;
8     printf("Please enter the number of students: ");
9     scanf("%d", &n);
10
11    // now allocate memory
12    marks = (int *)malloc(sizeof(int) * n);
13
14    printf("Enter the marks for each student: \n");
15    for (i = 0; i < n; i++) {
16        scanf("%d", &marks[i]);
17    }

```

কম্পিউটার প্রোগ্রামিং - বিভিন্ন খণ্ড

```

16
17     printf("Now here you can see the values:\n");
18
19     // now print the marks array
20     for (i = 0; i < n; i++) {
21         printf("%d\n", marks[i]);
22     }
23
24
25     return 0;
26 }
```

প্রোগ্রাম ৭-১

আমি এখনে আরে ব্যবহার না করে একটি ইন্টিজার পয়েন্টার int *marks; ব্যবহার করেছি। কারণ ইন্টিজারের পয়েন্টার কেবল ইন্টিজারকেই পয়েন্ট করতে পারে এমনটি নয়, এটি ইন্টিজার আরেকেও পয়েন্ট করতে পারে। ক্যারেষ্টার পয়েন্টার যেমন ক্যারেষ্টার ভ্যারিয়েবল ছাড়াও স্ট্রিং - অর্থাৎ ক্যারেষ্টার টাইপের আরেকে পয়েন্ট করতে পারে।

তারপরে আমি মেট শিক্ষার্থীর সংখ্যা কত, সেটি ব্যবহারকারীকে জিজ্ঞাসা করে জেনে নিয়েছি। সেটি n নামক ভ্যারিয়েবলে আছে।

এখন আমি n সংখ্যক শিক্ষার্থীর নম্বর রাখার জন্য মেমোরি বরাদ্দ করার চেষ্টা করব। কঠুক আর মেট ইন্টিজার সংখ্যা n, তাই আমাদের sizeof(int) × n বাইট জায়গা প্রয়োজন। সেই stdlib.h হেডার ফাংশনটি ইনক্লুড করতে হবে। ঠিকঠাক মেমোরি বরাদ্দ করতে পারলে ফাংশনটি একটি পয়েন্টার রিটার্ন করবে, যে মেমোরি বরাদ্দ করা হয়েছে, তার শুরুর ঠিকানা। আর কোনো সমস্যা হলে NULL রিটার্ন করবে। এভাবে মেমোরি বরাদ্দ করা হলে, সেটি কম্পিউটার মেমোরি যে, malloc এর আগে আমরা কেন (int *) লিখলাম? কারণ, *marks হচ্ছে ইন্টিজারের পয়েন্টার, আর malloc ফাংশনটি দিয়ে আমরা যেই ধরনের পয়েন্টারের জন্য মেমোরি অ্যালোকেশন করব, সেই ধরনের পয়েন্টারে টাইপ কাস্ট করে নিতে হবে।

* ফাংশন malloc

প্রোটোটাইপ:
void * malloc (size_t size)

ইনপুট:

কম্পিউটার প্রোগ্রামিং - বিভিন্ন খণ্ড

size_t size : ক্রত বাইট জায়গা নিতে হবে, সেই সংখ্যা ইনপুট মের
বিলান ভাবে:
void * : মেমোরি অ্যালোকেশনের পরে তার ঠিকানার ভয়েত পয়েন্টার রিটার্ন করে।
কাজ: size বাইট সমপরিমাণ জায়গা মেমোরি থেকে নেয় এবং সেই ঠিকানার একটি ভয়েত
পয়েন্টার রিটার্ন করে।

মেমোরি বরাদ্দ করার পরে আমরা এখন marks-কে আরের মতো ব্যবহার করতে পারব। প্রোগ্রাম যখন ব্যবহার করে থাকে, তখন তোমার কম্পিউটারের অপারেটিং সিস্টেম ওই প্রোগ্রাম হিপে মেমোরি দখল করেছিল, সেগুলো মুক্ত (Free - Free) করে দেবে। তাহলে অব্য সফটওয়্যার সেব মেমোরি ব্যবহার করতে পারবে। কিন্ত অনেক বড় সফটওয়্যার তৈরি করতে গেলে, তোমাকে আরই মেমোরি বরাদ্দ করার প্রয়োজন হতে পারে। আবার একসময় তোমার প্রোগ্রামে অন্য কোনো জায়গায় আগের বরাদ্দকৃত মেমোরি আর ব্যবহার করার দরকার নাও থাকতে পারে। তাই তোমার নিজে থেকে সেই মেমোরি ফ্রি করে দিতে হবে। এর শুরুত তোমরা এখন ছোটখাটো প্রোগ্রাম তৈরি সময় বুঝবে না। কিন্ত এই অভ্যাসটি করার বিকল্প নেই। তোমরা কেউ কেউ হ্যাতো মেমোরি লিক (memory leak) কথাটি শুনে থাকবে। ডায়নামিক মেমোরি অ্যালোকেশন করার পরে সেটি ফ্রি না করলে এ সমস্যার উভব হয়। আর সেটি খুঁজে বের করা অত্যন্ত কঠিন একটি কাজ। আর এ কারণেই অনেক প্রোগ্রামার পয়েন্টার ব্যবহার করতে ভয় পায়। যাই হোক, তুম যদি ঠিকঠাক প্রোগ্রাম লেখার অভ্যাস কর, তাহলে ভয়ের কিছু নেই।

মেমোরি ফ্রি করার জন্য আমরা free() ফাংশন ব্যবহার করব। তাহলে ওপরের প্রোগ্রামটি এখন

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int *marks;
7     int i, n;
8     printf("Please enter the number of students: ");
9     scanf("%d", &n);
10
11    // allocate memory
```

১১১

কম্পিউটার প্রোগ্রামিং – বিতীয় খণ্ড

```

12 marks = (int *) malloc(sizeof(int) * n);
13 if (marks == NULL) {
14     printf("Memory allocation failed for marks\n");
15     return 1;
16 }
17 printf("Enter the marks for each student: \n");
18 for (i = 0; i < n; i++) {
19     scanf("%d", &marks[i]);
20 }
21
22 printf("Now here you can see the values:\n");
23 for (i = 0; i < n; i++) {
24     printf("%d\n", marks[i]);
25 }
26
27 // now free the memory
28 free(marks);
29
30
31
32 }

```

প্রোগ্রাম ৭-২

malloc এর মতো আরেকটি লাইব্রেরি ফাংশন রয়েছে, যার নাম হচ্ছে calloc। এটি malloc এর মতোই কাজ করতে, তবে একটি বাড়িতি কাজ হিসেবে মেমোরির যেটুকু অংশ বরাদ্দ করা হয়েছে সেখানে সব ০ দিয়ে ইনিশিয়ালাইজ করে দেয়। এই পার্থক্যটি মনে রাখবে, মাঝে-মধ্যে পরীক্ষায় এই প্রশ্নটি জিজ্ঞাসা করা হয়।

৩৫ ফাংশন free

প্রোটোটাইপ:

`void free (void * ptr)`

ইনপুট:

`void * ptr : malloc, calloc ফাংশন দিয়ে অ্যালোকেশন করা মেমোরি স্লেকেশনের ঠিকানা ইনপুট নেয়।`

রিটার্ন ভ্যালু:

কম্পিউটার প্রোগ্রামিং – ছ'তাম ১০

কাজ:

`void : কোনো মান রিটার্ন করে না।`

`ptr : মেমোরি স্লেকেশনের বরাদ্দ করা জারী হচ্ছে দেয় (deallocation)।`

এবন তোমাদের জন্য একটি কাজ। ওপরের প্রোগ্রামটি malloc এর বদলে calloc ব্যবহার করে আবার লিখ। তোমার কেউ কেউ ভাবতে পারো, calloc ফাংশনের ব্যবহার তুমি কীভাবে জানবে? নুটি সংজ্ঞ উপায় আছে। প্রথমটি হচ্ছে, তোমার কম্পাইলারের সঙ্গে যদি লাইব্রেরি ফাংশনের স্ক্রিপ্ট সম্পর্ক স্থাপন থাকে, তাহলে সেখানে খুঁজতে পারো। দ্বিতীয় উপায় হচ্ছে গুগলে সার্চ করা। এবন তোমার জন্য একটি কাজ। ওপরের প্রোগ্রাম থেকে পৃথক্ক প্রেসি পর্যন্ত সব শিক্ষার্থীর গণিত পরীক্ষার সময় একটি আরেতে রাখতে চাচ্ছে। আরেতে রেখে তারপরে কী করবে সেটি তুমি জান। আপাতত পয়েন্টারের ব্যবহার দেখানোই আমার উদ্দেশ্য। এখন বিভিন্ন ক্লাসে শিক্ষার্থীর সংখ্যা তিনি। তো তুমি যদি ডায়নামিক মেমোরি অ্যালোকেশন সম্পর্কে না জানতে, তাহলে হয়তো মার্কস [5] [1000]। কিন্তু যেহেতু তুমি ডায়নামিক মেমোরি অ্যালোকেশন শিখে গিয়েছ। তাই প্রতি ক্লাসে সর্বোচ্চ এক হাজার শিক্ষার্থী ধরে এরকম একটি আয়ারে ডিক্রেয়ার করতে: এই প্রোজেক্টে মেমোরি অ্যালোকেশন করতে পারো। আমি কোড লিখে দিলাম:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int *ara[5], num[5];
7     int i, j, n;
8
9     for(i = 0; i < 5; i++) {
10         printf("Enter the number of students for class %d : ", i + 1);
11         scanf("%d", &n);
12         num[i] = n;
13         ara[i] = (int *) malloc(sizeof(int) * n);
14         if (ara[i] == NULL) {
15             printf("Memory allocation failed\n");
16             return 1;
17         }
18         for(j = 0; j < n; j++) {
19             printf("Enter marks for student %d: ", j + 1);

```

```

22
23
24
25
26
27
28
29
30
31
32
33
34
35
    scanf("%d", &ara[i][j]);
}

// now print the results
printf("Output\n");
for(i = 0; i < 5; i++) {
    printf("Class %d : ", i + 1);
    for(j = 0; j < num[i]; j++) {
        printf("%4d", ara[i][j]);
    }
    printf("\n");
}
return 0;
}

```

প্রোগ্রাম ৭-৩

কোডটি মনোযোগ দিয়ে দেখ, তারপরে টাইপ করে কম্পাইল ও রান কর। এখানে তোমার কোনো বাড়ি মেমোরি খরচ করা লাগল না। তবে একটি অতিরিক্ত অ্যারে ব্যবহার করতে হয়েছে, কোন ক্লাসে শিক্ষার্থীর সংখ্যা কত, সেটি রাখার জন্য। এখন আমরা যদি এটিও আগে থেকে না জানতাম যে মোট ক্লাস কয়টি, তাহলে ডায়নামিক মেমোরি অ্যালোকেশন কি কোনোভাবে আমাদের সাহায্য করতে পারে? হ্যাঁ, পারে। ধরা যাক সর্বোচ্চ মোট ১২টি ক্লাস আছে, কিন্তু ঠিক কতটি ক্লাসের শিক্ষার্থীদের জন্য আমাদের গণিত পরীক্ষার নম্বর রাখতে হবে, সেটি আমরা জানি না। তাহলে আমরা পয়েন্টারের পয়েন্টার ব্যবহার করতে পারি। প্রথমে মোট কতটি ক্লাস আছে, সেটি ইনপুট নিয়ে ততটি পয়েন্টারের জন্য মেমোরি অ্যালোকেশন করব। তারপরে আবার প্রতিটি পয়েন্টারের জন্য ক্লাসের শিক্ষার্থীর সংখ্যা অনুযায়ী মেমোরি অ্যালোকেশন করব। নিচের মতো কোড করতে পারি:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int **ara, num[12];
7     int i, j, total_classes, n;
8
9     printf("Enter the total number of classes : ");
10    scanf("%d", &total_classes);
11    ara = (int **)malloc(sizeof(int *) * total_classes);
12
13    for(i = 0; i < total_classes; i++) {
14        num[i] = i + 1;
15        ara[i] = (int *)malloc(sizeof(int) * num[i]);
16        if(ara[i] == NULL) {
17            printf("Memory allocation failed\n");
18            return 1;
19        }
20        for(j = 0; j < num[i]; j++) {
21            printf("Enter marks for student %d: ", j + 1);
22            scanf("%d", &ara[i][j]);
23        }
24    }
25
26    // now print the results
27    printf("Output\n");
28    for(i = 0; i < total_classes; i++) {
29        printf("Class %d : ", i + 1);
30        for(j = 0; j < num[i]; j++) {
31            printf("%4d", ara[i][j]);
32        }
33        printf("\n");
34    }
35
36    return 0;
37 }

```

১২৪

```

12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
    if(ara == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }
    for(i = 0; i < total_classes; i++) {
        printf("Enter the number of students for class %d : "
               , i + 1);
        scanf("%d", &n);
        num[i] = n;
        ara[i] = (int *)malloc(sizeof(int) * n);
        if(ara[i] == NULL) {
            printf("Memory allocation failed\n");
            return 1;
        }
        for(j = 0; j < n; j++) {
            printf("Enter marks for student %d: ", j + 1);
            scanf("%d", &ara[i][j]);
        }
    }
    // now print the results
    printf("Output\n");
    for(i = 0; i < total_classes; i++) {
        printf("Class %d : ", i + 1);
        for(j = 0; j < num[i]; j++) {
            printf("%4d", ara[i][j]);
        }
        printf("\n");
    }
    return 0;
}

```

প্রোগ্রাম ৭-৪

কোডটি নিজে টাইপ করে কম্পাইল ও রান কর। তারপরে কোডটি মনোযোগ দিয়ে পড়, বুঝতে সহজ হওয়ার কথা নয়।

এখন তোমাদের যদি প্রয়োজন হয়, তখন ডায়নামিক মেমোরি অ্যালোকেশন করবে। আর প্রয়োজন না হলে অথবা ডায়নামিক মেমোরি অ্যালোকেশন করার দরকার নেই। আমি যেমন ওপরের দুটি কোডেই মেমোরি ফ্রি করি নি, তোমরাও বড় বড় প্রোগ্রামে এমন ভুল করে রেখে দিলে পরে সেটি খুঁজতে যথেষ্টই বেগ পেতে হবে।

১২৫

৭.১ - পয়েন্টারের হিসাব-নিকাশ

আমরা যদি একটি ইন্টিজার পয়েন্টার ডিক্রেয়ার করি, যেমন `int *p`, তাহলে `p`-তে আমরা একটি ইন্টিজারের মেমোরি অ্যাড্রেস রাখতে পারব। আর `*p` ব্যবহার করে, `p` ঠিকানায় যে ইন্টিজার ঠিকানায় যে ভারিয়েবলটি রাখা আছে, তার মান পাওয়া যাবে। এখন `*p + 1` করলে কী হবে? `*p` হচ্ছে `p` হয় 20 তাহলে `*p + 1` হবে 21। আর `*(p + 1)`-এর মান কী হবে? এটি হচ্ছে `p` যদি পরবর্তী ঠিকানা যেই ভ্যারিয়েবলটিকে নির্দেশ করে, তার মান। ধরা যাক, `p`-এর মান 100। মনে পয়েন্টারটি মেমোরি 100 নম্বর ঘরকে নির্দেশ করছে। এখন `p`-এর মান এক বাড়ালে, সেটি সেক্ষেত্রে পরবর্তী ঘর হবে 104, কারণ ওই সিস্টেমে ইন্টিজারটি 100, 101, 102 ও 103 - এই চারটি ঘর দখল করবে। এটি যদি ক্যারেক্টার টাইপের পয়েন্টার হতো, আর সিস্টেমে ক্যারেক্টারের করবে আর একটি ঘরের সাইজ হচ্ছে এক বাইট। চলো, এখন একটি প্রোগ্রাম লেখা যাক।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int ara[] = {100, 300, 500, 700, 900};
6     int *p;
7     p = ara;
8
9     printf("%p : %d\n", *p);
10    printf("%p + 1 : %d\n", *p + 1);
11    printf("%p + 2 : %d\n", *p + 2);
12    printf("%p + 3 : %d\n", *p + 3);
13
14 }
15
16 }
```

প্রোগ্রাম ৭-৫

তোমার প্রথম কাজ হবে, প্রোগ্রামটি না চালিয়ে আউটপুট কী আসবে, সেটি বের করা। তারপরে প্রোগ্রাম কম্পাইল করে চালাও। আউটপুট হবে এমন:

১২৬

<code>*p : 100</code>
<code>*p + 1 : 101</code>
<code>*p + 2 : 102</code>
<code>*p + 3 : 500</code>

তুমি যা হিসাব করেছিলে আর আউটপুট যা দেখছ, তা যদি না মেলে তাহলে তুমি প্রোগ্রাম লেখার আগে যেই আলোচনা করা হয়েছিল, অর্থাৎ পয়েন্টারের হিসাব-নিকাশ, সেই অংশটি আবার পড়। তারপর নিচের প্রোগ্রামটি দেখো এবং আউটপুট কী হবে, সেটি কাগজে-কলমে বের করার চেষ্টা কর।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     char *str = "Bangladesh";
6     printf("%c, %c, %c, %c\n", *str, *(str + 1), *(str + 2),
7            *(str + 3));
8     printf("%c, %c, %c, %c\n", *str, *str + 1, *str + 2, *str
9            + 3);
10
11    return 0;
12 }
```

প্রোগ্রাম ৭-৬

এই প্রোগ্রামের আউটপুট বোঝার জন্য তোমাদেরকে একটি জিনিস মনে করিয়ে দিই। একটি ইন্টিজার যেমন 6 এর সঙ্গে 1 যোগ করলে, যোগফল হবে 7। আর একটি ক্যারেক্টার, যেমন M-এর সঙ্গে 1 যোগ করলে, যোগফল কী হবে? M-এর আসকি মানের সঙ্গে এক যোগ হয়ে, সেটি যে অক্ষরের আসকি মান, সেই অক্ষরটি পাওয়া যাবে।

`'M' + 1 => 77 + 1 => 78 => 'N'`

আশা করি, আউটপুট বুঝতে তোমাদের আর সমস্যা হবে না। প্রোগ্রামের আউটপুট হবে এমন:

B, a, n, g
B, C, D, E

১২৭

p+1 করলে *p* এর মান কত বাঢ়বে সেটি নির্ভর করবে *p* কোন টাইপের ভ্যারিয়েবলের পদ্ধতি
করে তার ওপর। চলো আরও একটি কোড থেকে এই যোগ-বিয়োগটি দেখি:

```

1 #include <stdio.h>
2
3 int main()
4 {
5     char *p, a = 10;
6     int *q, b = 'F';
7     double *r, c = 302.64;
8
9     p = &a;
10    q = &b;
11    r = &c;
12
13    printf("Size of char: %d byte\n", sizeof(char));
14    printf("p : %p\n", p);
15    printf("p+1: %p\n", p+1);
16    printf("p+2: %p\n\n", p+2);
17
18    printf("Size of int: %d byte\n", sizeof(int));
19    printf("q : %p\n", q);
20    printf("q+1: %p\n", q+1);
21    printf("q+2: %p\n\n", q+2);
22
23    printf("Size of int: %d byte\n", sizeof(double));
24    printf("r : %p\n", r);
25    printf("r+1: %p\n", r+1);
26    printf("r+2: %p\n\n", r+2);
27
28    return 0;
29 }
```

প্রোগ্রাম ৭-৭

এই প্রোগ্রামের আউটপুট আমার কম্পিউটারে আসে এরকম:

```

Size of char: 1 byte
p : 0060FF03
p+1: 0060FF04
p+2: 0060FF05

Size of int: 4 byte
```

```

q : 0060FEFC
q+1: 0060FF00
q+2: 0060FF04
size of int: 8 byte
r : 0060FFF0
r+1: 0060FFF8
r+2: 0060FF00
```

এখানে *p* তে রয়েছে একটি ক্যারেক্টারের অ্যাড্রেস, *q* তে রয়েছে একটি ইন্টজারের অ্যাড্রেস ও *r* এ
রয়েছে একটি ডবল টাইপের ভ্যারিয়েবলের অ্যাড্রেস।

এখানে

৭.২ - ভয়েড পয়েন্টার (void pointer)

পয়েন্টার ভ্যারিয়েবল ডিক্লেয়ার করার সময় আমরা শুধুতেই কোন ধরনের ডেটার জন্য
পয়েন্টারটি ব্যবহার করা হবে, সেটি বলে দেই। ইন্টজার টাইপের ভ্যারিয়েবলের জন্য ইন্টজার
পয়েন্টার, ক্যারেক্টার টাইপের ভ্যারিয়েবলের জন্য ক্যারেক্টার টাইপের পয়েন্টার, ডবল টাইপের
ভ্যারিয়েবলের জন্য ডবল টাইপের পয়েন্টার ইত্যাদি। এখন, কখনো কখনো আমরা আগে থেকে
জানব না যে, কোন ধরনের ভ্যারিয়েবলের ঠিকানা আমাদের রাখতে হবে, কিন্তু পয়েন্টার
আগেতাগে ডিক্লেয়ার করা লাগবে। তখন আমরা ভয়েড টাইপের পয়েন্টার ব্যবহার করতে পারি।

ভয়েড পয়েন্টার ডিক্লেয়ার করতে হবে এভাবে: void *ptr;

ধরা যাক, *n* একটি ইন্টজার ভ্যারিয়েবল, তাহলে সেটির ঠিকানা আমরা পয়েন্টারে রাখতে পারব
এভাবে: ptr = &n;

কিন্তু আমরা যদি এখন *ptr*-কে ডিরেফারেন্স করতে চাই, তাহলে *ptr লিখলে হবে না, আমাদের
টাইপ কাস্ট করে নিতে হবে, এভাবে: *((int *)ptr)।

চলো একটি প্রোগ্রাম লিখে দেখি। তোমরা প্রোগ্রামটি দেখে দেখে টাইপ করে কম্পাইল ও রান
করবে।

```

1 #include <stdio.h>
2
3 int main()
4 {
5     void *vp;
6 }
```

```

7   int n = 20;
8
9   vp = &n;
10
11  printf("Address of n: %p\n", &n);
12  printf("Value of vp: %p\n", vp);
13  printf("Content of vp: %d\n",*((int *)vp));
14
15  return 0;
16 }
```

প্রোগ্রাম ৭-৮

এখন জোড়াদের কি মনে পড়ে যে আমরা malloc ফাংশন যখন ব্যবহার করেছি, তখন সেটি int টাইপ কস্ট করে নিয়েছি? এর কারণ হচ্ছে, malloc ফাংশনটি ভয়েড পয়েন্টার পয়েন্টারে) মেমোরি ব্যবহার করতে আলাদা আলাদা ফাংশন ব্যবহার করতে হতো।

৭.৩ - ফাংশন পয়েন্টার

আমরা এখন পর্যন্ত বিভিন্ন ধরনের ভারিমেবল ও স্ট্রাকচারের পয়েন্টার দেখেছি। এখন তোমাদের জানিবে রাখা দরকার যে ফাংশনেরও পয়েন্টার আছে। কারণ তুমি যখন কোনো ফাংশন তৈরি কর, সেটি প্রোগ্রাম চলার সময় কম্পিউটার মেমোরির কোনো একটি নির্দিষ্ট জায়গায় থাকবে, আর সেই জায়গার ঠিকানা ফাংশন পয়েন্টারে রাখা যায়।

আমি যদি একটি ফাংশন লিখি, যেটি প্যারামিটার হিসেবে দুটি ইন্টজার নেবে আর রিটার্ন করবে একটি ইন্টজার, তাহলে তার প্রোটোটাইপ লেখা যায় এভাবে:

```
int fnc (int, int);
```

আর একই ফাংশন, আমরা যদি ফাংশন পয়েন্টার ব্যবহার করতে চাই, সেক্ষেত্রে লিখব এভাবে:

```
int (*fnc) (int, int);
```

তাহলে fnc-তে আমি একটি ফাংশনের ঠিকানা (মেমোরি অ্যাড্রেস) রাখতে পারব। চলো একটি প্রোগ্রাম লিখে দেখি। তোমরা নিচের প্রোগ্রামটি টাইপ করে কম্পাইল ও রান কর:

১৩০

```

1 #include <stdio.h>
2
3 int add(int n1, int n2)
4 {
5     return n1 + n2;
6 }
7 int sub(int n1, int n2)
8 {
9     return n1 - n2;
10 }
11
12 int main()
13 {
14     int (*fnc)(int, int);
15     int n1 = 10, n2 = 5;
16
17     fnc = &add;
18
19     printf("Result : %d\n", fnc(n1, n2));
20
21     fnc = &sub;
22
23     printf("Result : %d\n", fnc(n1, n2));
24
25
26
27 }
28 }
```

প্রোগ্রাম ৭-৯

তোমরা যদি কোডটি মনোযোগ দিয়ে খেয়াল কর, তাহলে বুঝে যাবে কীভাবে ফাংশন পয়েন্টার ব্যবহার করত হয়। প্রোগ্রাম রান করলে আউটপুট আসবে এমন:

```
Result : 15
Result : 5
```

ফাংশন পয়েন্টার ব্যবহার করার একটি বড় সুবিধা হচ্ছে, এটি দিয়ে ফাংশনকে আরেকটি ফাংশনের প্যারামিটার হিসেবে পাঠানো যায়। চলো আমরা আরেকটি উদাহরণ দেখি:

```

1 #include <stdio.h>
2
```

১৩১

কম্পিউটার প্রোগ্রামিং - বিজীয় খণ্ড

```

3 int add(int n1, int n2)
4 {
5     return n1 + n2;
6 }
7
8 int sub(int n1, int n2)
9 {
10    return n1 - n2;
11 }
12
13 int operate(int (*op)(int, int), int a, int b)
14 {
15     return op(a, b);
16 }
17
18 int main()
19 {
20     int n1 = 10, n2 = 5;
21
22     printf("Result : %d\n", operate(&add, n1, n2));
23
24     printf("Result : %d\n", operate(&sub, n1, n2));
25
26     return 0;
27 }
```

প্রোগ্রাম ৭-১০

ওপরের প্রোগ্রামটি তোমরা টাইপ করে কম্পাইল ও রান করে দেখো। আগের প্রোগ্রামের মতো ফাংশনের পয়েন্টার পাঠাচ্ছি। `operate(&add, n1, n2)` না লিখে `operate(add, n1, n2)` লিখলেও ফাংশনটি একইভাবে কাজ করবে, কারণ ফাংশনের নামটিই তার ঠিকানা হিসেবে ব্যবহার করা যায় (যারের বেলাতে যেমন অ্যারের নামটি অ্যারের প্রথম ঘরের ঠিকানা হিসেবে ব্যবহার করা যায়)।

এখন প্রশ্ন হচ্ছে, ফাংশন পয়েন্টার আমি কেন শিখব এবং কখন ব্যবহার করব? আসলে তোমরা এখন পর্যন্ত যতটুকু প্রোগ্রামিং শিখেছ, তাতে তোমাদেরকে দুটি প্রশ্নের উভয় দেওয়া, আমার জন্য বেশ কঠিন কাজ। আপত্ত তোমরা কেবল জেনে রাখো যে ফাংশনের ওপয়েন্টার বলে একটি এর প্রয়োজন পরে (তোমাদের মধ্যে যারা সিরিয়াস প্রোগ্রামার হবে, তাদের অবশ্যই ফাংশন পয়েন্টারের দরকার পড়বে), তোমরা তখন হাতের কাছে যে বই থাকবে কিংবা ইন্টারনেট, সেটি

কম্পিউটার প্রোগ্রামিং - বিজীয় খণ্ড

ফাংশনের পয়েন্টার ব্যবহারের সুযোগ থাকবে না। এখন আমরা একটি উদাহরণ দেখব, যেখানে ভয়েত নামটির নিখে লিখে। তবে পরীক্ষা কিংবা ইন্টারনেটের সময় হয়তো তোমাদের বই কিংবা ফাংশনের পয়েন্টার ব্যবহার করা হয়েছে।

৭.৮ - qsort & bsearch ফাংশন

তোমরা তো সবাই নিচয়ই জানো যে আমাদের কাছে যদি একাধিক ডেটা থাকে, তাহলে সেগুলোকে আমরা কোনো একটি নির্দিষ্ট ক্রমে সাজাতে পারি। এই সাজানোর কাজটিকে বলে সার্টিং করার জন্য অনেক রকম অ্যালগরিদম আছে, যেগুলোর মধ্যে খুব জনপ্রিয় হচ্ছে কুইকসর্ট। তোমরা যখন অ্যালগরিদম পড়বে, তখন নিচয়ই কুইকসর্ট অ্যালগরিদম লিখবে এবং এর সৌন্দর্যে মুক্ত হবে। তার আগ পর্যন্ত তোমরা stdlib.h হেডার ফাইলের qsort নামে যে ফাংশন রয়েছে, সেটি ব্যবহার করতে পারো। ফাংশনটির প্রোটোটাইপ হচ্ছে এরকম:

```
void qsort (void* base, size_t nitems, size_t size, int
(*compar)(const void*, const void*));
```

ফাংশনটির নাম হচ্ছে qsort, রিটার্ন টাইপ হচ্ছে void (অর্থাৎ ফাংশনটি কোনো কিছু রিটার্ন করবে না)। এখন ফাংশনের প্যারামিটারগুলো দেখা যাক।

প্রথম প্যারামিটার হচ্ছে base নামক একটি ভয়েত পয়েন্টার। যেই আরেটি আমাদের সর্ট করতে হবে, তার প্রথম উপাদানের পয়েন্টার হচ্ছে এই base। qsort ফাংশন কিন্তু জানে না যে কোন ধরনের আরে সর্ট করতে হবে। তাই এখানে void পয়েন্টার ব্যবহার করা হচ্ছে যেন ইন্টিজার আরে, ফ্লোট অ্যারে, স্ট্রাকচারের আরে ইত্যাদি যেকোনো অ্যারেকেই আমরা সর্ট করতে পারি।

বিত্তীয় প্যারামিটার হচ্ছে nitems, যেটি নির্দেশ করে আরেতে মোট কয়টি উপাদান রয়েছে। তৃতীয় প্যারামিটার হচ্ছে size, যেই আরেটি সর্ট করা হবে, তার একটি উপাদান যত বাইট জমাগা দখল করে, সেটি size-এ রাখতে হবে।

চতুর্থ প্যারামিটার হচ্ছে একটি ফাংশনের পয়েন্টার। ফাংশনটি দুটি জিনিসের তুলনা করার জন্য ব্যবহার করা হবে। ফাংশনটি প্যারামিটার হিসেবে দুটি const void* নেবে এবং একটি ইন্টিজার রিটার্ন করবে। void* ব্যবহারের কারণ হচ্ছে, কী ধরনের জিনিস তুলনা করতে হবে, সেটি আমরা জানি না, তাই আমরা যেকোনো ধরনের ভারিয়েবল পাঠাতে পারব। আর তার আগে const দেওয়ার অর্থ হচ্ছে যেই মানগুলো আমরা পাঠাব, সেগুলো পরিবর্তন করা হবে না।

এখন চলো আমরা একটি প্রোগ্রাম লিখে দেখি, qsort কীভাবে ব্যবহার করতে হয়।

কম্পিউটার প্রোগ্রামিং - বিভাগ খণ্ড

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int comparefunc (const void * a, const void * b)
5 {
6     return (*(int*)a - *(int*)b);
7 }
8
9 int main()
10 {
11     int i, n=5;
12     int values[] = { 65, 6, 100, 1, 250 };
13
14     qsort(values, 5, sizeof(int), comparefunc);
15
16     for(i = 0 ; i < n; i++) {
17         printf("%d ", values[i]);
18     }
19
20     printf("\n");
21
22     return 0;
23 }

```

প্রোগ্রাম ৭-১১

প্রোগ্রামটি রান করলে তোমরা এমন আউটপুট দেখবে: 1 6 65 100 250।

একটি বিষয় খেয়াল কর যে, comparefunc-এর ভেতরে আমরা কিন্তু ইন্টজার পয়েন্টারে রূপান্তর করে নিয়েছি, কারণ আমরা এখন জানি যে, আমাদের অ্যারেটি ইন্টজার অ্যারে।

আমরা যদি চাই যে বড় থেকে ছোট ক্রমে সঠ করব, তাহলে comparefunc-এর ভেতরে a - b এর বদলে আমরা b - a লিখতাম (অবশ্যই পয়েন্টার কাস্ট করে নিতে হবে)।

এরকম আরেকটি ফাংশন আছে bsearch, যেটি দিয়ে বাইনারি সার্চ করা যায়। বাইনারি সার্চ নিয়ে আমার কম্পিউটার প্রোগ্রামিং ১ম খণ্ড বইতে বিস্তারিত আলোচনা আছে, ভুলে গেলে ওই বই থেকে দেখে নিতে পারো। ফাংশনটির প্রোটোটাইপ হচ্ছে এরকম:

```

void *bsearch(const void *key, const void *base, size_t nitems,
size_t size, int (*compar)(const void *, const void *));

```

কম্পিউটার প্রোগ্রামিং - বিভাগ খণ্ড

এখানেও দেখো তাইড পয়েন্টার ও ফাংশন পয়েন্টার ব্যবহার করা হচ্ছে। ফাংশনটির রিটার্ন ফাংশনটি রিটার্ন করবে, আর যেহেতু আগেভাগে জানার উপায় নেই যে, কোন ধরনের ফাংশনের প্রথম প্যারামিটার হচ্ছে একটি ভয়েড পয়েন্টার (key) যেটি আমরা অ্যারেতে খুঁজব বা সার্চ করব। আর পরের জিনিসগুলো ওপরের qsort ফাংশনের মতোই। এখানেও আমাদের একটি কম্পেয়ার ফাংশন লিখতে হবে। চলো একটি পূর্ণাঙ্গ উদাহরণ দেখি।

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int comparefunc (const void * a, const void * b)
5 {
6     return (*(int*)a - *(int*)b);
7 }
8
9 int main()
10 {
11     int key, *item, n=5;
12     int values[] = {1, 2, 5, 8, 10};
13
14     while (1) {
15         printf("Enter the value of the key (0 to exit): ");
16         scanf("%d", &key);
17         if (key == 0) {
18             break;
19         }
20         item = (int *) bsearch(&key, values, n, sizeof(int),
21                               comparefunc);
22         if (item != NULL) {
23             printf("Item found: %d\n", *item);
24         } else {
25             printf("Item not found in array\n");
26         }
27     }
28 }
29
30

```

অধ্যায় ৮ - মজাৰ কিছু প্ৰোগ্ৰাম

return ০;

প্ৰোগ্ৰাম ৭-১২

প্ৰোগ্ৰামটি যদি ভালোভাবে খেয়াল কৰে দেখো, যেই অ্যারেতে আমাদেৱ কোনো জিনিস খুঁজতে হবে, সেই অ্যারেটি কিন্তু আগে থেকেই সঁচ কৰা থাকতে হবে। নইলে বাইনাৰি সার্চ কৰা যাবে না। এখন তোমাদেৱ জন্য কাজ হচ্ছে এখন একটি প্ৰোগ্ৰাম লেখা যেটি প্ৰথমে একটি অ্যারেকে qsort আৰ আমি যেই উদাহৰণগুলো দেখিয়েছি, এগুলোতে ইন্টিজাৰ অ্যারে ব্যবহাৰ কৰেছি। কিন্তু তোমৰা যেকোনো ধৰনেৰ অ্যারে ব্যবহাৰ কৰতে পাৰো, এমনকি স্ট্রাকচাৰেৰ অ্যারেও।

এই অধ্যায়ে আমি কয়েকটি প্ৰোগ্ৰাম লিখে তোমাদেৱ দেখাব। তবে প্ৰোগ্ৰামগুলো ব্যাখ্যা কৰবো না। এগুলো আপত্তত তোমাৰ মজা পাওয়াৰ জন্য। বিজ্ঞাপিত বুৰাতে চাইলে গুগল সার্চ কৰে বুৰো নোব।

৮.১ - সময় পরিমাপ

তোমাদেৱ প্ৰায়ই জানতে ইচ্ছে কৰতে পাৰে যে তোমাৰ প্ৰোগ্ৰামটি চলতে কেমন সময় নিল। এজন তোমৰা time.h হেডাৰ ফাইলেৰ সাহায্য নিতে পাৰো। আমি এখন একটি প্ৰোগ্ৰাম লিখব, যেটি দেখে তুমি বুৰাতে পাৰবে যে কীভাৱে সময় মাপতে হয়।

```

1 #include <stdio.h>
2 #include <time.h>
3
4 void fnc(int x, int n) {
5     x = n * 2;
6 }
7
8 int main()
9 {
10    int i, j, x, n;
11    clock_t start_time, end_time;
12    double time_elapsed;
13
14    start_time = clock();
15
16    n = 12345678;
17    for (i = 0; i < 1000000000; i++) {
18        for (j = 0; j < 10; j++) {
19            x = n * 2;
20        }
21    }
22    end_time = clock();
23    time_elapsed = (double) (end_time - start_time) /
24    CLOCKS_PER_SEC;

```

১৩৭

```

25     printf("Time: %lf seconds\n", time_elapsed);
26     start_time = clock();
27
28     n = 12345678;
29     for (i = 0; i < 1000000000; i++) {
30         for (j = 0; j < 10; j++) {
31             fnc(x, n);
32         }
33     }
34
35     end_time = clock();
36     time_elapsed = (double) (end_time - start_time) /
37     CLOCKS_PER_SEC;
38
39     printf("Time: %lf seconds\n", time_elapsed);
40
41     return 0;
42 }
43 }
```

প্রোগ্রাম ৮-১

তবে তোমার কম্পিউটারে তো অনেক প্রোগ্রাম চলছে, যেমন তোমার অপারেটিং সিস্টেম। তাই এই পদ্ধতিটি শতকরা একশ ভাগ সঠিক, এই দাবি মোটেও করা যাবে না। তবে কিছুটা ধারণা জিনিস হাতেনাতে প্রমাণ পেলে। সেটি হচ্ছে, ফাংশন কল করা একটি সময়সাপেক্ষ কাজ। তাই একাধিক জায়গায় করতে হচ্ছে, তখন কাজটি একটি ফাংশনের ভেতরে করে সেই ফাংশনটি কল করবে।

৮.২ – র্যানডম নম্বর (random number) তৈরি

এখন আমরা দেখব কীভাবে র্যানডম নম্বর পাওয়া যায়। এজন্য stdlib.h হেডার ফাইলের ভেতরে rand() ফাংশনটি ব্যবহার করতে হবে।

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
```

```

1 int main()
2 {
3     for(int i = 0 ; i < 5; i++ ) {
4         printf("%d\n", rand());
5     }
6     return 0;
7 }
```

প্রোগ্রাম ৮-২

এখন, ওপরের প্রোগ্রামটি চলালে পাঁচটি সংখ্যা প্রিন্ট হবে। সমস্যা হচ্ছে তুমি বারবার রান করলে একই সংখ্যা প্রিন্ট হবে। এই সমস্যা থেকে পরিআনের উপায় হচ্ছে srand() ফাংশন ব্যবহার করা।

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 int main()
6 {
7     time_t t;
8     srand(0);
9
10    for(int i = 0 ; i < 5; i++ ) {
11        printf("%d\n", rand());
12    }
13
14    return 0;
15 }
```

প্রোগ্রাম ৮-৩

ওপরের প্রোগ্রামটি রান কর। দেখবে পাঁচটি র্যানডম নম্বর প্রিন্ট হয়েছে। প্রোগ্রামটি আরেকবার রান কর, দেখবে ওই পাঁচটি সংখ্যাই প্রিন্ট হয়েছে। আসলে আমাদেরকে যেটি করতে হবে, সংখ্যাটি কী হতে পারে? আমরা যদি কম্পিউটারের বর্তমান সময় পাঠাই, তাহলেই কিন্তু একেকবার একেক সংখ্যা পাঠানোর কাজটি হয়ে যাবে। এই সংখ্যাটিকে সিড (seed) বলে। তোমার র্যানডম নম্বর নিয়ে আরও পড়াশোনা করলে বিষয়টি বুরতে পারবে। আমি কোড দিয়ে দিলাম।

কম্পিউটার প্রোগ্রামিং – বিতীয় খণ্ড

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 int main()
6 {
7     time_t t;
8     srand((unsigned) time(&t));
9
10    for(int i = 0 ; i < 5; i++ ) {
11        printf("%d\n", rand());
12    }
13
14    return 0;
15 }
```

প্রোগ্রাম ৮-৮

তোমরা কিন্তু লুডু খেলার প্রোগ্রাম লেখার জন্য বিরাট অস্ত্র পেয়ে গেলে!

তোমাদের কাছে প্রশ্ন! আমি যদি চাই প্রতিটি সংখ্যা 0 থেকে 100-এর মধ্যে থাকবে, তাহলে কী করতে হবে?

৮.৩ – নিজে হেডার ফাইল তৈরি করা

এখন আমরা দেখব কীভাবে নিজস্ব হেডার ফাইল তৈরি করতে হয়। তোমরা dimik.h নামে একটি ফাইল তৈরি কর। তারপরে ফাইলে নিচের কোড লিখ (আর কিছু লিখবে না কিন্তু)

```

1 int add(int n1, int n2)
2 {
3     return n1 + n2;
4 }
5
6 int mul(int n1, int n2)
7 {
8     return n1 * n2;
9 }
```

প্রোগ্রাম ৮-৯

কম্পিউটার প্রোগ্রামিং – বিতীয় খণ্ড

এখার ফাইলটি সেভ করে বক্স করে দাও। এখন test.c নামে আরেকটি ফাইল তৈরি কর এবং নিচের কোড লিখ।

```

1 #include <stdio.h>
2 #include "dimik.h"
3
4 int main()
5 {
6     int n1 = 10, n2 = 5;
7     printf("%d + %d = %d\n", n1, n2, add(n1, n2));
8     printf("%d x %d = %d\n", n1, n2, mul(n1, n2));
9
10    return 0;
11 }
```

প্রোগ্রাম ৮-১০

কোডটি কম্পাইল ও রান করলে দেখবে নিচের মতো আউটপুট পাবে:

$$\begin{aligned} 10 + 5 &= 15 \\ 10 \times 5 &= 50 \end{aligned}$$

বাহ, তুমি নিজের হেডার ফাইল তৈরি করা শিখে গেলে!

অধ্যায় ৯ - বিবিধ

৯.১ - কনস্ট্যান্ট (constant) ও ম্যাক্রো (Macro)

সি প্রোগ্রামিং ভাষায় বিজ্ঞ প্রিপ্রেসর (preprocessor) আছে। যেমন #include, যা তোমরা ইতিমধ্যে ব্যবহার করেছ। এই প্রিপ্রেসরের পরে একটি হেডার ফাইলের নাম দিতে হয় আর তারপর তোমরা সেই হেডার ফাইলের ফাংশনগুলো তোমার প্রোগ্রামে ব্যবহার করতে পারো। কারণ প্রোগ্রাম কম্পাইল করার সময় যেসব হেডার ফাইল তুমি ইনক্লুড করেছ, তাদের সোর্স কোড তোমার সোর্স কোডের সঙ্গে চলে আসে (এই অধ্যায়ের শেষে প্রোগ্রাম কম্পাইলের ধাপসমূহ অংশটুকু পড়লে বিষয়টি আরও পরিকার হবে)। আরও বেশ কিছু প্রিপ্রেসর আছে, তবে আমি এখন তোমাদেরকে #define প্রিপ্রেসরের ব্যবহার দেখাব।

কনস্ট্যান্ট (constant)

ভারিয়েল মানে যার মান পরিবর্তন হয়, আর কনস্ট্যান্ট মানে যার মান একবার ঠিক করে দিলে আর পরিবর্তন হবে না। তোমরা সি প্রোগ্রামে #define CONSTANT_NAME Value এভাবে লিখে কাজ করতে পারো। এতে হবে কী, প্রোগ্রাম কম্পাইল করার সময় যেসব জায়গায় CONSTANT_NAME পাবে, সেগুলো Value দিয়ে পরিবর্তন করে দেবে। নিচের প্রোগ্রামটি দেখো:

```

1 #include <stdio.h>
2 #include <math.h>
3
4 #define MIN -1
5 #define PI (2 * acos(0))
6
7 int main()
8 {
9     printf("Value of MIN : %d\n", MIN);
10    printf("Value of PI : %lf\n", PI);
11
12    return 0;
13 }
```

প্রোগ্রাম ৯-১

কম্পিউটার প্রোগ্রামিং - ইতীয় খণ্ড

এটি টাইপ করে কম্পাইল ও রান করলেই ব্যবহার কীভাবে #define-এর সাহায্যে আমরা প্ল্যাটফর্ম তৈরি করতে পারি। আর তোমরা যারা পাইয়ের মানের সঙ্গে পরিচিত, তারা ভাবতে নিখেছি, সেট সিতে পাইয়ের মান বের করার সবচেয়ে তালো পদ্ধতি, নইলে তোমরা নানান ক্ষেত্রের পড়তে পার। তবে এটি মূলত প্রোগ্রামিং প্রতিযোগিতায় যারা অংশগ্রহণ কর, তাদের জন্মই বেশি গুরুত্বপূর্ণ। যারা ত্রিকোণমিতি পড়েছে, তারা বিষয়টি ব্যবহার করে পারবে, বাকিদের আপাতত এটি না ব্যবহার করতে পারবে।

এখন তোমরা নিচের প্রোগ্রামটি দেখো।

```

1 #include <stdio.h>
2
3 #define P 50
4 #define Q 60;
5
6 int main()
7 {
8     int a = P;
9     int b = Q;
10
11    printf("a = %d, b = %d\n", a, b);
12
13    return 0;
14 }
```

প্রোগ্রাম ৯-২

কোডটি তালোভাবে খেয়াল করলে দেখবে যে আমি এক জায়গায় সেমিকোলন দিতে ভুলে গিয়েছি। কিন্তু তোমরা যদি প্রোগ্রামটি কম্পাইল কর, ঠিকঠাক কম্পাইল হবে, আর আউটপুটও ঠিক আসবে। কারণ হচ্ছে আমি q-কে ডিফাইন করেছি [৬০] (সেমিকোলন সহ), তাই কম্পাইল হওয়ার সময় q-এর বদলে [৬০] বসে যাবে, তাই কম্পাইলার কোনো এরর পাবে না।

ম্যাক্রো

এখন আমরা ম্যাক্রো তৈরি করা শিখব। আসলে ম্যাক্রো বলতে এই ক্ষেত্রে আমরা ব্যবহ যে কিছু কাজের সমষ্টি, যা ওই ম্যাক্রো তৈরি করার পরে ম্যাক্রোকে কল করেই করা যাবে। তোমরা নিচের প্রোগ্রামটি দেখ। একটি ব্যাপার লক্ষ কর যে, define-এর ভেতর কোনো ডেটা টাইপ ব্যবহার করা হয়নি। এখন প্রোগ্রামটি টাইপ কর, কম্পাইল ও রান কর।

১৪৩

কম্পিউটার প্রোগ্রামিং – বিতীয় খণ্ড

```

1 #include <stdio.h>
2
3 #define MAX(a, b) (a > b? a : b)
4
5 int main()
6 {
7     int a = 83, b = 323;
8     double d1 = 8.32323, d2 = 3.33332323;
9
10    printf("maximum of %d and %d is %d\n", a, b, MAX(a, b));
11    printf("maximum of %lf and %lf is %lf\n", d1, d2, MAX(d1,
12          d2));
13
14    return 0;
15 }
```

প্রোগ্রাম ৯-৩

এখানে আমি টার্মিনাল অপারেটর ব্যবহার করেছি। এটি ব্যবহারের নিয়ম হচ্ছে :

কন্ট্রিভেশন ? স্টেটমেন্ট ১ : স্টেটমেন্ট ২

কন্ট্রিভেশন যদি সত্য হয়, তাহলে স্টেটমেন্ট ১ কাজ করবে, আর সত্য না হলে স্টেটমেন্ট ২। এখন $a > b$ যদি সত্য হয়, তাহলে a নইলে b , এভাবে আমরা a ও b -এর মধ্যে বৃহত্তমটি খুঁজে বের করতে পারি। আরেকটি মজার বিষয় হচ্ছে ম্যাক্রোটি কিন্তু ইন্টিজার ও ডবল, দুই ধরনের ডেটা টাইপের জন্যই কাজ করবে।

আমরা চাইলে আরও বড় ফাংশনও ম্যাক্রো আকারে লিখতে পারি। একাধিক লাইন হলে প্রতি লাইনের শেষে ব্যাকস্যাশ চিহ্ন () দিতে হবে। নিচের প্রোগ্রামটি খেয়াল কর। এখানে তোমার দুটি জিনিস শেখার আছে, কীভাবে মাস্টিলাইন (একাধিক লাইন) ম্যাক্রো তৈরি করতে হয় এবং কীভাবে এক্সর ব্যবহার করে দুটি ভ্যারিয়েবলের মান বদলে দেওয়া যায়। এক্সর ভুলে গেলে বিটওয়াইজ অপারেশন অধ্যায়টি দেখে নাও।

```

1 #include <stdio.h>
2
3 #define SWAP(a, b) { \
4     a ^= b; \
5     b ^= a; \
6     a ^= b; \
7 }
```

১৮৮

কম্পিউটার প্রোগ্রাম ১০-৪

```

8 int main()
9 {
10     int a = 83, b = 323;
11     printf("a = %d, b = %d\n", a, b);
12     SWAP(a, b);
13     printf("a = %d, b = %d\n", a, b);
14
15     return 0;
16 }
17
18 }
```

প্রোগ্রাম ১০-৪

একটি জিনিস কেবল মনে রাখবে। #define ব্যবহার করে কন্ট্যাক্ট বা ম্যাক্রো যাই তৈরি করি না কেন, এখানে কিন্তু ফাংশন কল করার মতো কোনো ব্যাপার ঘটে না, কম্পাইল করার সময় তোমার সোর্স কোডের কন্ট্যাক্ট বা ম্যাক্রোগুলো তাদের আসল কোড দিয়ে পরিবর্তিত হয়ে যায়। তাই ফাংশন কল করার সময় রানটাইমে মেইন সময় খরচ হয়, সেই সময় এখানে খরচ হয় না। যেমন ওপরের প্রোগ্রামে মেইন ফাংশনের মধ্যে তুমি SWAP(a, b) লিখেছ। কিন্তু কম্পাইল করার সময় কম্পাইলার করবে কী, ওখানে SWAP(a, b) এর বদলে নিচের তিনিটি লাইন বসিয়ে দেবে:

```

a ^= b;
b ^= a;
a ^= b;
```

এখন প্রশ্ন হচ্ছে, প্রোগ্রাম লেখার সময় কখন #define ব্যবহার করবে?

- ১) তোমাদের যদি কোনো কোড এমনভাবে লেখার দরকার হয় যে, সেটি ফাংশন ব্যবহার করলে সুবিধা, আবার ফাংশন ব্যবহারের কারণে প্রোগ্রাম ধীরগতির (slow - slow) হয়ে গিয়েছে, তখন #define ব্যবহার করে ম্যাক্রো তৈরি করে নেবে।
- ২) যেকোনো কন্ট্যাক্টের মান ও #define দিয়ে নির্ধারণ করে দেবে। এতে কোডটি পড়া সহজ হয় (মানে অন্য কেউ পড়ে সহজে বুঝতে পারে) আর ভুলবশত কন্ট্যাক্টের মান পরিবর্তন করার সম্ভাবনা করে যায়। আর কন্ট্যাক্টের জন্য সব বড় হাতের অঙ্কর ব্যবহার করাও একটি ভালো অভ্যাস।

১৪৫

৯.২ - এনিউমারেশন (enumeration)

প্রোগ্রামিং করা বা প্রোগ্রাম লেখার দুটি উদ্দেশ্য থাকে। প্রথম উদ্দেশ্য হচ্ছে কম্পাইলার থাণ্ডে প্রোগ্রাম টিকমতো বুঝতে পারে এবং কম্পাইল করতে পারে, যেটি কম্পিউটারে টিক্সটক রান করবে। দ্বিতীয় উদ্দেশ্য হচ্ছে আরেকজন প্রোগ্রামার যেন তোমার লেখা প্রোগ্রাম খুব সহজেই বুঝতে পারে। এই দ্বিতীয় উদ্দেশ্যটি কিন্তু কোনো অংশেই কম গুরুত্বপূর্ণ নয়। তোমরা যত বেশি সময় ধরে প্রোগ্রামিং করবে, যত বড় প্রোগ্রামার হবে, তত বেশি এই সহজবোধ্য কোড সেখার গুরুত্ব বুঝতে পারবে।

অধিকাংশ প্রোগ্রামিং ভাষাতেই এনিউমারেশন বলে একটি জিনিস রয়েছে। এটি আসলে একই রকম বেশি কিছু কনস্ট্যান্ট একসঙ্গে ডিক্রেয়ার করার একটি সহজ পদ্ধতি। যেমন ধরা যাক তুমি তিনটি রঙের জন্য তিনটি কনস্ট্যান্ট ডিক্রেয়ার করতে চাও। তাহলে সেটি এভাবে করতে পারো:

```
#define RED 1
#define GREEN 2
#define BLUE 3
```

এনিউমারেশন ব্যবহার করেও তুমি কাজটি করতে পারো এভাবে :

```
enum COLOR {RED, GREEN, BLUE};
```

এখনে COLOR হচ্ছে টাইপ আর তার কেবল তিনটি মান সম্পর্ক, সেগুলো হচ্ছে RED, GREEN, BLUE।

চলো একটি প্রোগ্রাম লিখে কম্পাইল ও রান করি। তাহলে বিষয়টি পরিষ্কার হয়ে যাবে।

```
1 #include <stdio.h>
2
3 enum COLOR {RED, GREEN, BLUE};
4
5 int main()
6 {
7     enum COLOR selected_color;
8
9     int num;
10
11    printf("Enter 1 for RED, 2 for GREEN, 3 for BLUE: ");
12    scanf("%d", &num);
13
14    if (num == 1) {
```

```
15         selected_color = RED;
16     } else if (num == 2) {
17         selected_color = GREEN;
18     } else if (num == 3) {
19         selected_color = BLUE;
20     }
21
22     printf("selected_color : %d\n", selected_color);
23
24     return 0;
25
26
27 }
```

প্রোগ্রাম ৯-৫

প্রোগ্রামটি রান করে ইনপুট 1, 2 বা 3 দিলে সেই অন্যান্য selected_color ভায়িরেবলের তেতরে নির্দিষ্ট রঙ অ্যাসাইন হবে।

আমরা এখন কনস্ট্যান্টের লিন্ট তৈরি করব, প্রথম কনস্ট্যান্টের মান হবে 0, দ্বিতীয়টির 1, তৃতীয়টির 2, এভাবে ত্রৈমিক মান অ্যাসাইন হতে থাকবে। একটি প্রোগ্রাম লিখে সেটি পরীক্ষা করা যাক।

```
1 #include <stdio.h>
2
3 enum COLOR {NO_COLOR, RED, GREEN, BLUE};
4
5 int main()
6 {
7     printf("value : %d\n", NO_COLOR);
8     printf("value : %d\n", RED);
9     printf("value : %d\n", GREEN);
10    printf("value : %d\n", BLUE);
11
12    return 0;
13 }
```

প্রোগ্রাম ৯-৬

ওপরের প্রোগ্রামটি টাইপ করে কম্পাইল ও রান কর। আউটপুট আসবে এমন:

```
value : 0
value : 1
```

১৪৭

```
value : 2
value : 3
```

আমরা চাইলে বিজ্ঞ কম্প্যাক্টের মানও ঠিক করে দিতে পারি। নিচের প্রোগ্রামটি লক্ষ কর:

```
1 #include <stdio.h>
2
3 enum COLOR {NO_COLOR = 0, RED = 40, GREEN = 44, BLUE = 60};
4
5 int main()
6 {
7     printf("value : %d\n", NO_COLOR);
8     printf("value : %d\n", RED);
9     printf("value : %d\n", GREEN);
10    printf("value : %d\n", BLUE);
11
12    return 0;
13 }
```

প্রোগ্রাম ১-৭

এটি রান করলে নিচের মতো আউটপুট আসবে:

```
value : 0
value : 40
value : 44
value : 60
```

প্রোগ্রাম মানুষের জন্য সহজবোধ্য করে লেখার বিষয়টি যেহেতু আসল, তাই আরেকটি জিনিসের সঙ্গে তোমাদের পরিচয় করিয়ে দিতে চাই, সেটি হচ্ছে সুইচ (switch)। অনেক সময় অনেকগুলো if-else না লিখে switch ব্যবহার করলে কোড দেখতে সুন্দর হয় ও পড়তে সহজ হয়। তাই অনেক প্রোগ্রামিং ভাষাতেই এর প্রচলন রয়েছে। নিচের উদাহরণটি খেয়াল কর। নিজে টাইপ করে কম্পাইল ও রান করতে ভুলবে না।

```
1 #include <stdio.h>
2
3 enum DAY {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
4           FRIDAY, SATURDAY};
5
6 int main()
7 {
8     enum DAY today;
```

```
8     today = THURSDAY;
9
10    switch (today) {
11        case SUNDAY:
12            printf("Today is Sunday. It is a work day\n");
13            break;
14        case MONDAY:
15            printf("Today is Monday. It is a work day\n");
16            break;
17        case TUESDAY:
18            printf("Today is Tuesday. It is a work day\n");
19            break;
20        case WEDNESDAY:
21            printf("Today is Wednesday. It is a work day\n");
22            break;
23        case THURSDAY:
24            printf("Today is Thursday. It is a work day\n");
25            break;
26        case FRIDAY:
27            printf("Today is Friday, today is holiday\n");
28            break;
29        case SATURDAY:
30            printf("Today is Saturday, today is holiday\n");
31            break;
32    }
33
34    return 0;
35 }
36 }
```

প্রোগ্রাম ১-৮

এখন তোমরা একটি পরীক্ষা কর। ওপরের প্রোগ্রামটি থেকে break স্টেটমেন্টটি বাদ দিয়ে দাও। এখন তোমরা একটি পরীক্ষা কর। ওপরের প্রোগ্রামটি থেকে break স্টেটমেন্টটি বাদ দিয়ে দাও। এখন তোমরা একটি পরীক্ষা কর। ওপরের প্রোগ্রামটি থেকে break স্টেটমেন্টটি বাদ দিয়ে দাও। আমি তারপরে প্রোগ্রাম চালিয়ে দেশো কী হয়। এখন চিন্তাবন্ধনা করে বিষয়টি বোঝার চেষ্টা কর। আর তোমরা চাইলেই এক লাইন লিখে বলে দিতে পারতাম, কিন্তু একটি চিন্তা করার অভ্যাস কর। আর তোমরা যদি চাইতে যে কোনো কেইসের সঙ্গে না মিললে বিশেষ কোনো কিছু প্রিন্ট করবে, সেটিও করা সম্ভব। একটি হোট উদাহরণ দিই, বুবাতে পারবে। তবে তার আগে একটি মজার তথ্য। অনেক অনেক বছর আগে (ঠিক কত বছর, সেটি বলতে পারছি না), মানুষ যখন ওলন্তে শেখা শুরু করল, সম্ভব। একটি হোট উদাহরণ দিই, বুবাতে পারবে। কোনো কিছু একটি দেখলে বুবাত যে একটি জিনিস অনেক বছর আগে (ঠিক কত বছর, সেটি বলতে পারছি না), মানুষ যখন ওলন্তে শেখা শুরু করল, তখন মানুষ তিনটি জিনিস বুবাতে পারত। কোনো কিছু একটি দেখলে বুবাত যে একটি জিনিস রয়েছে, দুটি জিনিস দেখলে দুটি জিনিস বুবাতে পারত। কিন্তু যখনই দুইয়ের বেশি জিনিস দেখত,

তখন তার কাছে অনেক মনে হতো!

১৪৯

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int n;
6
7     printf("Enter a positive integer (> 0): ");
8     scanf("%d", &n);
9
10    switch(n) {
11        case 1:
12            printf("One item\n");
13            break;
14        case 2:
15            printf("Two items\n");
16            break;
17        default:
18            printf("Many items\n");
19    }
20
21    return 0;
22 }
```

প্রোগ্রাম ৯-৯

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[])
5 {
6     int n1, n2, sum;
7
8     n1 = atoi(argv[1]);
9     n2 = atoi(argv[2]);
10
11    sum = n1 + n2;
12
13    printf("%d + %d = %d\n", n1, n2, sum);
14
15    return 0;
16 }
```

প্রোগ্রাম ৯-১০

ধরা যাক, তুমি প্রোগ্রামটি cmdarg.c নামে সেভ করেছ। এখন টার্মিনালে `gcc cmdarg.c -o sum.out` কমান্ড দিয়ে প্রোগ্রামটি কম্পাইল কর। তারপরে `./sum.out 5 7` লিখে কিবোর্ডে এন্টার কি চাপ। তাহলে আউটপুটে দেখবে $5 + 7 = 12$ । ধাপগুলো হবে নিচের মতো:

```
$ gcc cmdarg.c -o sum.out
$ ./sum.out 5 7
5 + 7 = 12
```

তাহলে আমরা দেখলাম, প্রোগ্রামে টার্মিনাল থেকে কোনো আর্গুমেন্ট পাঠাতে হলে, সেগুলো পাঠাতে হয় মেইন ফাংশনের মাধ্যমে। মেইন ফাংশনের প্যারামিটারে সবসময় দুটি জিনিস থাকবে: `int argc, char *argv[]`। নাম যে argv আর argc এর argv দিতে হবে এমন কোনো কথা নেই, তবে সবাই এরকম নামই দেয়। argc হচ্ছে আর্গুমেন্ট কাউন্ট (argument count) আর argv হচ্ছে আর্গুমেন্ট ভেক্টর (argument vector)। আর্গুমেন্ট কাউন্টে থাকে তুমি কয়াটি আর্গুমেন্ট পাঠালে আর আর্গুমেন্ট ভেক্টরে থাকে সেই আর্গুমেন্টগুলো। তুমি যদি কোনো কিছু না পাঠাও, তাহলেও argc এর মান হবে 1, কারণ সবসময় প্রথম আর্গুমেন্ট হচ্ছে এক্সেকিউটিবেল ফাইলটি চলানোর জন্য যে কমান্ড আমরা দিই, সেটি। তাই আমরা আগের প্রোগ্রামে প্রথম আর্গুমেন্ট পেয়েছি argv[1]-এ আর দ্বিতীয়টি পেয়েছি argv[2]-তে। চলো একটি প্রোগ্রাম লিখে দেখি।

কম্পিউটার প্রোগ্রামিং – বিতীয় খণ্ড

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[])
5 {
6     printf("Argument count : %d\n", argc);
7     printf("First argument : %s\n", argv[0]);
8     return 0;
9 }

```

প্রোগ্রাম ৯-১১

ওপরের প্রোগ্রামটি cmdarg.c নামে সেভ করে রান করি। তাহলে এরকম আউটপুট আসবে:

```

$ gcc cmdarg.c -o prog.out
$ ./prog.out
Argument count : 1
First argument : ./prog.out

```

এখন প্রশ্ন হচ্ছে মেইন ফাংশনের দ্বিতীয় প্যারামিটারটি আমরা অ্যারে না বলে ভেষ্টের বলছি কেন? কারণ অ্যারে হলে তো একটি নির্দিষ্ট সাইজ দিতে হবে, আর ভেষ্টের হলে যতগুলো খুশি আমরা ইনপুট দিতে পারব (এই ভেষ্টের কিন্তু পদাৰ্থবিজ্ঞানের ভেষ্টের নয়, এটি মনে রাখবে)।

চলো, আমরা আরেকটি প্রোগ্রাম লিখে দেখি।

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     if (argc != 2) {
6         printf("Please enter one argument (your name)\n");
7         return 1;
8     }
9     printf("Welcome %s\n", argv[1]);
10
11 }
12

```

প্রোগ্রাম ৯-১২

প্রোগ্রামটি এবারে রান করি :

কম্পিউটার প্রোগ্রামিং – বিতীয় খণ্ড

```

$ gcc name.c
$ ./a.out Tamim Shahriar
Please enter one argument (your name)

```

এই আউটপুটের কী কারণ? চিন্তা করে বের কর। আমরা আবার প্রোগ্রাম রান করার চেষ্টা করি।

```

$ ./a.out Tamim
Welcome Tamim

```

এখন নিচয়ই বুঝতে পারছে যে আগেরবার কী সমস্যা ছিল। এখন আমি যদি প্রোগ্রামকে

যোৱাতে চাই যে আমার নাম Tamim Shahriar, তাহলে ডবল কোটেশন ব্যবহার করতে হবে।

```

$ ./a.out "Tamim Shahriar"
Welcome Tamim Shahriar

```

আমি এখন পর্যন্ত যেসব ভাষায় প্রোগ্রাম লিখেছি, সব ভাষাতেই কমান্ড লাইন আর্ডারেট নেওয়ার

ব্যবহা আছে। তাই এটি নিয়ে স্বচ্ছ ধারণা থাকা জরুরি।

এখন একটি মজার কথা বলি। তোমরা তো একটু আগে প্রোগ্রাম কম্পাইল করার জন্য নিচের

কমান্ডটি দিয়েছে:

```
gcc cmdarg.c -o prog.out
```

এখনে কিন্তু gcc একটি প্রোগ্রাম আর এই প্রোগ্রামে তুমি কমান্ড লাইন আর্ডারেট হিসেবে

cmdarg.c -o prog.out পাঠাচ্ছ। gcc প্রোগ্রামটি করে কী, তার আর্ডারেট হিসেবে যে সি

প্রোগ্রাম পায়, ধরে নেয় যে তাকে কম্পাইল করতে হবে। আর a.out নামে এক্সিকিউটেবল ফাইল

তৈরি করে। আর যদি -o আর্ডারেট পায়, তাহলে তার পরের যেই আর্ডারেট থাকবে, সেই নামে

তৈরি হচ্ছে অপশনাল বা ঐচিক। ggc-তে এরকম এক্সিকিউটেবল ফাইল তৈরি করাবে। এই অংশটি হচ্ছে অপশনাল বা ঐচিক।

আবারও কিছু ঐচিক আর্ডারেট আছে, যেগুলো এই বইতে আলোচনা করা সম্ভব হচ্ছে না। আবার

যদি ggc প্রোগ্রামে কোনো আর্ডারেট না দিই, তাহলে কী হবে? চলো সেটি করে দেখি:

```
$ gcc
clang: error: no input files
```

gcc প্রোগ্রাম তোমাকে একটি এর মেসেজ দিচ্ছে, কারণ প্রোগ্রামের ভেতরে কোথাও বলা আছে।

যে অবশ্যই একটি ইনপুট ফাইল (মানে সি প্রোগ্রাম) থাকতে হবে।

কম্পিউটার প্রোগ্রামিং – দ্বিতীয় খণ্ড

তোমরা দেখলে তো, কমান্ড লাইন আর্ডেমেন্ট কত শুরুত্বপূর্ণ একটি জিনিস। এখন তোমাদের কাজ হবে এমন একটি প্রোগ্রাম লিখা, যেখানে কমান্ড লাইন আর্ডেমেন্ট হিসেবে আমি যতক্ষণে সংখ্যাই দিই না কেন, সেগুলো যোগ করে যোগফল প্রিন্ট করবে।

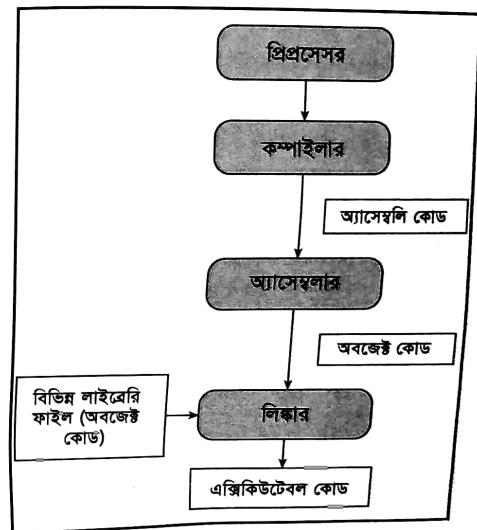
১.৪ – প্রোগ্রাম কম্পাইল হওয়ার ধাপসমূহ

তোমাদের অনেকের মনেই পশ্চ এসেছে যে আমরা যখন একটি প্রোগ্রাম কম্পাইল করি, তখন আসলে কী হয়? অনেকেই আমাকে জিজাসা করেছে যে এই বিষয়ে কম্পিউটার প্রোগ্রামিং - ১ম খণ্ড বইতে কিছু লেখা নেই কেন? উত্তর হচ্ছে, আগে বললে তোমাদেরকে বিষয়টি বোানো বেশ কষ্টসাধ্য ছিল। কিন্তু এখন তোমরা প্রস্তুত হয়ে গিয়েছ, তাই এখন আর জানতে কোনো বাধা নেই। সি প্রোগ্রাম কম্পাইল হওয়ার সময় মোট চারটি কাজ হয় :

- 1) **প্রিপ্রেসিং (Preprocessing)**: এটি হচ্ছে প্রথম ধাপ। এখানে তুমি তোমার সোর্স কোডে মেসব প্রিপ্রেসর, যেমন include, define ইত্যাদি ব্যবহার করেছ, সেগুলোর কোড তোমার কোডে চলে আসে। যেমন তুমি যদি তোমার প্রোগ্রামে #include <stdio.h> লিখ, তাহলে stdio.h হেডার ফাইলের সোর্স কোড তোমার প্রোগ্রামে চলে আসবে। তাই printf, scanf এসব ফাংশনকে চিনতে কম্পাইলারের কোনো সমস্যা হবে না। আবার তুমি যদি #define ব্যবহার করে কোনো ম্যাক্রো তৈরি করে থাকো, সেক্ষেত্রেও প্রতিটি ম্যাক্রোর জায়গায় তার সোর্স কোড ব্যবহার করে থাকো। এভাবে আরও যত প্রিপ্রেসর আছে (যদি তুমি তোমার সোর্স কোডে ব্যবহার করে থাক), সেগুলো সব চলে গিয়ে তাদের সোর্স কোড তোমার প্রোগ্রামে বসে যাবে। এই কাজটি আসলে প্রিপ্রেসর নামে আলাদা একটি প্রোগ্রাম করে, যেটি তোমার কম্পাইলারের স্বয়ংক্রিয়ভাবেই চালায়, তোমার নিজের কিছু করতে হয় না। তাহলে এই ধাপের ইনপুট হচ্ছে সি সোর্স কোড আর আউটপুট হচ্ছে ওই সোর্স কোড আর সেটিতে যত প্রিপ্রেসর ব্যবহার করা হয়েছে, সেসবের বদলে তাদের সোর্স কোড।
- 2) **কম্পাইল (Compile)** : প্রিপ্রেসিং করার পরে মেই সোর্স কোড পাওয়া যায়, সেটি এখন কম্পাইল হয়। এই ধাপে তোমার সি প্রোগ্রামের সিনট্যাক্স (syntax) পরীক্ষা করে দেখা হয় যে সব ঠিকঠাক আছে কি না। আবার ধর তুমি printf ফাংশন ব্যবহার করেছ, কিন্তু stdio.h হেডার ফাইলটি ইনক্লুড করনি, সেটিও এই ধাপে এসে ধরা পরে যাবে। কারণ প্রিপ্রেসিং ধাপেই stdio.h হেডার ফাইলের সোর্স কোড তোমার প্রোগ্রামে চলে আসার কথা। যদি তোমার কোডে কোনো ভুল না থাকে (লজিক্যাল ভুল নয়, সি প্রোগ্রামিং ভাষার ভুল), তাহলে তোমার কোড কম্পাইল হবে এবং অ্যাসেম্বলি ল্যাঙ্গুয়েজের কোড তৈরি হবে।

কম্পিউটার প্রোগ্রামিং – দ্বিতীয় খণ্ড

- 3) **অ্যাসেম্বলি (Assembly)** : এই ধাপে অ্যাসেম্বলি ল্যাঙ্গুয়েজের কোডকে মেশিন ল্যাঙ্গুয়েজের সোর্স কোডে রূপান্তর করা হয় এবং তার ফলে অবজেক্ট ফাইল তৈরি হয়। তোমরা অনেক সময় .o (ডট ও) এক্সটেনশনের যেসব ফাইল দেখো, সেগুলো হচ্ছে অবজেক্ট ফাইল এবং সেখানেই থাকে মেশিন ল্যাঙ্গুয়েজের কোড। এতে তোমার কম্পিউটারের মাইক্রোপ্রসেসর বুবাতে পারে, এমন ভাষায় কোড লেখা থাকে।



ছবি ২১: প্রোগ্রাম কম্পাইল হওয়ার ধাপসমূহ

- 4) **লিঙ্কিং (linking)** : একটি প্রোগ্রাম যখন চলে, তার মানা প্রকারের রিসোর্স দরকার হয়। যেমন বিভিন্ন সিলেক্টেম রিসোর্স ব্যবহারের দরকার হতে পারে। কিংবা কোনো লাইব্রেরি ফাংশন ব্যবহারের দরকার হতে পারে। তো এই ধাপে এসে প্রোগ্রামের জন্য অযোজনায় রিসোর্সগুলো লিঙ্ক বা যুক্ত করে মূড়ান্ত এক্সিকিউটেবল ফাইল তৈরি হয়, যেটি রান করলে আলাদা একটি অবজেক্ট ফাইলে আছে যার সঙ্গে লিঙ্ক করলে ফাংশনটি ব্যবহার করা যাবে। আলাদা একটি অবজেক্ট ফাইলে আছে যার সঙ্গে লিঙ্ক করলে ফাংশনটি ব্যবহার করা যাবে। আলাদা একটি অবজেক্ট ফাইলে কেবল ফাংশনটির কথা বলা আছে, ইমপ্রিমেট করা নেই। এই আর stdio.h হেডার ফাইলে কেবল ফাংশনটির কথা বলা আছে, ইমপ্রিমেট করা নেই। এই ধরনের ফাংশনগুলো আসলে লাইব্রেরি আকারে থাকে আর তার জন্য আলাদা অবজেক্ট

ফাইল আগে থেকেই তৈরি করা থাকে, যাতে যেসব প্রোগ্রাম ওই ফাঁশন ব্যবহার করবে, তাদের প্রত্যেকের জন্য প্রতিবার একই অবজেক্ট ফাইল তৈরি করতে না হয়। printf.o নামক অবজেক্ট ফাইলে printf ফাঁশনের ইমপ্লিমেটেশন আছে। তো তোমার প্রোগ্রামের অবজেক্ট ফাইল (যেটি দ্বিতীয় ধাপে তৈরি হয়েছে)-এর সঙ্গে এসব অবজেক্ট ফাইলগুলোকে যুক্ত করে একটি এক্সেকিউটিভেল ফাইল তৈরি করার মাধ্যমেই সি প্রোগ্রাম কম্পাইল করার শেষ ধাপটি সম্পন্ন হয়।

ছবি ২১ এর মাধ্যমে ধাপগুলো আবার তুলে ধরা হলো।

৯.৫ - #typedef ও #define নিয়ে কিছু কথা

আমি অনেককে দেখেছি #typedef ও #define এই দুটি বিষয়ের মধ্যে গুলিয়ে ফেলে। লক্ষ কর #define হচ্ছে একটি প্রিপ্রসেসর টোকেন, অর্থাৎ, প্রিপ্রসেসিংয়ের ধাপে কোডে যত ডিফাইন করা কি (key) আছে তা তার ভালু দিয়ে অপসারিত হয়ে তারপর কম্পাইলারের কাছে পাঠানো হবে। অর্থাৎ, প্রিপ্রসেসিংয়ের পরে কম্পাইলারের কাছে যে কোডটি কম্পাইল করার জন্য পাঠানো হবে তাতে #define বলে কোনো টোকেন থাকবে না।

নিচের উদাহরণটি লক্ষ কর:

```

1 #include <stdio.h>
2 #define ip int *
3
4 int main(int argc, char *argv[])
5 {
6     int * a; // a is an integer pointer
7     ip b; // b is also an integer pointer
8     ip c, d; // c is an integer pointer
9     // BUT, d is only an integer
10
11    return 0;
12 }
```

প্রোগ্রাম ৯-১৩

এখনে আমরা ip কথাটিকে ডিফাইন করলাম int * মান দিয়ে, এখন প্রিপ্রসেসর প্রসেসিং করার প্রিপ্রসেসিংয়ের পরে কোডটি হবে অনেকটা এমন।

```

4 int main(int argc, char *argv[])
5 {
6     int * a; // a is an integer pointer
7     int * b; // b is also an integer pointer
8     int * c, d; // c is an integer pointer
9     // BUT, d is only an integer
10
11    return 0;
12 }
```

প্রোগ্রাম ৯-১৪

অর্থাৎ, আট নম্বর লাইনে গিয়ে c কে ডিক্রেয়ার করা হবে ইন্টিজার পয়েন্টার হিসেবে কিন্তু d কে ডিক্রেয়ার করা হবে ইন্টিজার হিসেবে।

আমারা যদি এই কাজটি typedef ব্যবহার করে করি তাহলে কোডটি হবে এমন:

```

1 #include <stdio.h>
2
3 typedef int * ip;
4
5 int main()
6 {
7     int * a; // a is an integer pointer
8     ip b; // b is also an integer pointer
9     ip c, d; // c and d both are integer pointers
10
11    return 0;
12 }
```

প্রোগ্রাম ৯-১৫

এখানে ip কথাটিকে ইন্টিজার পয়েন্টার টাইপেরই আরেকটি নাম হিসেবে ডিফাইন করা হয়েছে। এখন আমরা যত ভ্যারিয়েবলকে ip টাইপ হিসেবে ডিক্রেয়ার করব, সবগুলোই আসলে ইন্টিজার এখন আমরা যত ভ্যারিয়েবলকে ip টাইপ হিসেবে ডিক্রেয়ার করা হবে। #typedef কে আরও জটিল টাইপ পয়েন্টার টাইপের ভ্যারিয়েবল হিসেবে ডিক্রেয়ার করা যায়। #typedef কে আরও জটিল টাইপ ডিফাইন করার কাজেও ব্যবহার করা যায়।

```

1 #include <stdio.h>
2
3 typedef int * ip;
4 typedef int a100[100];
5 typedef int (*fp) (int);
```

১৫৭

```

6
7 int main()
8 {
9     ip a, b, c; // a, b, c are integer pointers
10    a100 p, q; // p, q both are array of 100 integers
11    fp f1, f2; // f1, f2 both are pointers to functions
12        // which takes one integer parameter as input
13        // and returns an integer
14
15    return 0;
16 }

```

প্রোগ্রাম ৯-১৬

এছাড়া #typedef অন্যান্য ভারিয়েবলের মতো ক্ষেপিং নীতি মেনে চলে যেখানে #define এর কোনো ক্ষেপিং নীতি নেই। যেখানে ডিফাইন করা হয় তার পরে থেকে যেকোনো জায়গায় ব্যবহার করা যায়।

৯.৬ - main() ফাংশন ও return 0

আমরা যখন সি ল্যাঙ্গুয়েজে কোড লেখি তখন কোডের main() ফাংশন এর প্রথম লাইন থেকে এক্সিকিউশন শুরু হয়। main() ফাংশন যদি এইভাবে ডিক্রেয়ার করা হয় [int main()] তাহলে এর অর্থ হচ্ছে যে ফাংশনটি যখন এক্সিকিউশন শেষ হবে তখন সে একটি ইন্টিজার রিটার্ন করবে। অর্থাৎ, ফাংশনের শেষে আমাদের কোনো একটা ইন্টিজার রিটার্ন করতে হবে। প্রচলিত নিয়মে ০ রিটার্ন করা হয়, প্রোগ্রামটি ঠিকভাবে কোনো সমস্যা ছাড়াই চলেছে সেটি বোধানের জন্য। তবে ০ ই যে রিটার্ন করতে হবে এমন কোনো কথা নেই। চাইলে যেকোনো ইন্টিজারই রিটার্ন করা যায়।

সি প্রোগ্রামের ক্ষেত্রে সাধারণত ISO নীতিমালা অনুসরণ করে কম্পাইলার তৈরি করা হয়ে থাকে। ISO/IEC 9899:1989 (C90) আদর্শ নীতিমালা অনুযায়ী, নিচে উল্লেখিত তিনটি উপায়ে main() ফাংশন ডিক্রেয়ার করা যাবে।

```

int main(void)
int main(int argc, char **argv)
int main(int argc, char *argv[])

```

যেখানে দ্বিতীয় ও তৃতীয় লাইনটি সমতুল্য। কম্পাইলার থেকে ইনপুট নেওয়ার ক্ষেত্রে দ্বিতীয় অথবা তৃতীয়টি ব্যবহার করা হয়। রিটার্নের ব্যাপারে আদর্শ নীতিমালা হচ্ছে একটি প্রোগ্রাম তিন ধরনের মান রিটার্ন করতে পারবে :

```

0
EXIT_SUCCESS
EXIT_FAILURE

```

যেখানে দ্বিতীয় ও তৃতীয় মানটি stdlib.h ফাইলে #define করা আছে। আবার, ISO/IEC 9899:1999 (C99) আদর্শ নীতিমালায় আরও যে শর্ত আরোপ করা হয়েছে তা হলো,

- main() ফাংশনের ডিক্রেয়ারেশনে int শব্দটি অবশ্যই উল্লেখ করতে হবে। অর্থাৎ, main() না লিখে অবশ্যই int main() লিখতে হবে। যদি এই লাইনটি না লেখা হয় ডিফল্ট হিসেবে main() ফাংশন এক্সিকিউশনের শেষে 0 রিটার্ন করা হবে।

আমরা যদি চাই যে কিছুই রিটার্ন করব না, সেক্ষেত্রে আমরা int main() এর পরিবর্তে void main() ব্যবহার করতে পারি। তবে int main() ব্যবহার করাটাই বেশি সমর্থিত ও উৎসাহিত।

সি++ ল্যাঙ্গুয়েজেও C99 এর মতো তিনভাবে main() ফাংশন ডিক্রেয়ার করা যায় :

```

int main(void)
int main(int argc, char **argv)
int main(int argc, char *argv[])

```

সি++ এর আদর্শ নিয়মে void main() ব্যবহার করা একেবারেই নিষিদ্ধ, সি++ এর আদর্শ নিয়মে void main() ব্যবহার করতে পারি। তা না হলে কোড কম্পাইল হবে অর্থাৎ main() ফাংশনকে অবশ্যই ইন্টিজার রিটার্ন করতে হবে, তা না হলে কোড কম্পাইল হবে না।

এখন প্রশ্ন হচ্ছে main() ফাংশনের কিছু রিটার্ন করার প্রয়োজন কী? আমরা যখন কোনো একটি কোড লিখে প্রোগ্রামটি চলাই, আমরা অপারেটিং সিস্টেমকে নির্দেশ দেই এবং অপারেটিং কোড প্রোগ্রামটিকে চলাই। প্রোগ্রামের এক্সিকিউশন যখন শেষ হবে তখন সে ০ মানটি সিস্টেম সিস্টেমের কাছে রিটার্ন করবে। ০ নির্দেশ করে যে প্রোগ্রামটি সঠিকভাবে চলেছে। অপারেটিং সিস্টেম অর্থাৎ কোনো কম্পাইল লাইনের মাধ্যমে যেভাবেই প্রোগ্রাম চলাই না আমরা অপারেটিং সিস্টেম অর্থাৎ কোনো কম্পাইল লাইনের মাধ্যমে যেভাবেই প্রোগ্রাম চলাই না কেন, প্রোগ্রাম কত মান রিটার্ন করল সেটি গুরুত্বপূর্ণ নয়। যেটি জানা গুরুত্বপূর্ণ সেটি হলো প্রোগ্রামটি ঠিকভাবে চলে শেষ হয়েছে কিনা। আদর্শ নিয়ম হচ্ছে,

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

- প্রোগ্রাম ঠিকভাবে চলে এক্সিট করলে 0 রিটুর্ন করা, এবং
- প্রোগ্রাম চলাকালে কোনো এরর (মেমোরি ওভারফ্লো অথবা অ্যাড্রেসিং সংক্রান্ত এরর অথবা যেকোনো এরর) হয়ে যদি প্রোগ্রামটি মধ্যপথেই বন্ধ হয়ে যায় তাহলে, 0 ছাড়া যেকোনো মান রিটুর্ন করা।

আবার, কিছু কিছু ক্ষেত্রে রিটুর্নকৃত মানটি শুরুত্বপূর্ণ হতে পারে। তবে সেসব আলোচনা এখন না করাই ভালো। ভবিষ্যতে তোমরা আরও প্রোগ্রামিং করলে ও প্রোগ্রামিং নিয়ে পড়াশোনা করলে জানতে পারবে।

৯.১ – lvalue এবং rvalue

এলভ্যালু এবং আরভ্যালু কি বোঝার আগে শুরুতেই নিচের কোডটি টাইপ করে তোমার কম্পিউটারে কম্পাইল কর:

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int a;
6     a = 2;
7     3 = a;
8
9     return 0;
10 }
```

প্রোগ্রাম ৯-১৭

কি দেখতে পেলে? এই কোডটি কম্পাইল করতে গেলে কম্পাইলার এরকম একটি এরর দেবে:

error: lvalue required as left operand of assignment

তাহলে দেখো আমরা যখন লিখছি, $a = 2$, তখন a ভ্যারিয়েবলের মধ্যে 2 মানটি অ্যাসাইন হচ্ছে কিন্তু, যদি, আমি লিখি $3 = a$, তখন a ভ্যারিয়েবলের মধ্যে 3 মানটি অ্যাসাইন করতে পারছি না। এখানে 6 ও 7 নম্বর লাইনে যা লেখা হয়েছে এগুলো হলো এক্সপ্রেশন। এক্সপ্রেশন লিখতে হলে সমান সমান চিহ্নের বামে যে অংশটি থাকে তাকে বলে এলভ্যালু (lvalue)। এখানে সাধারণত এমন রাশি থাকতে হয় যার একটি নির্দিষ্ট মেমোরি লোকেশন থাকে এবং যেই লোকেশন আরক্সেস করার অনুমতি থাকে, যেমন কোনো ভ্যারিয়েবল। আরভ্যালু (rvalue)

কম্পিউটার প্রোগ্রামিং - দ্বিতীয় খণ্ড

বলা হয় সাধারণত সমান সমান চিহ্নের ডানদিকের রাশিকে। এই রাশিমালার একটি নির্দিষ্ট মান থাকত হয়। আরভ্যালুও নির্দিষ্ট মেমোরি লোকেশন থাকে, তবে এই মেমোরি লোকেশনটি অহারণী এবং কোনো ভ্যারিয়েবলকে নির্দেশ করে না। সাধারণত এই লাইনের অপারেশন শেষ হয়ে যাওয়ার পরে এই মেমোরি লোকেশনটি ইনভ্যালিড বা অর্থহীন হয়ে যায়। নিচের উদাহরণগুলো দেখে বলো তো কোন লাইনগুলো ঠিকমতো কাজ করবে আর কোন লাইনগুলো কাজ করবে না?

```

1 int a,b;
2 enum color {RED, GREEN, BLUE};
3 color c;
4 a      = 4;
5 b      = 2;
6 4      = a;
7 (a+2) = 3;
8 b      = (a+2);
9 a+b   = a-b;
10 c    = red;
11 green = c;
```

প্রোগ্রাম ৯-১৮

আগে নিজেরা চিন্তা কর, পরে নিচের সমাধান দেখো:

লাইন নম্বর	বামপক্ষ	ডানপক্ষ	ফলাফল
4	a এটি একটি ভ্যারিয়েবল যার একটি মেমোরি লোকেশন আছে এবং একটি মানও আছে। সুতরাং এটি এলভ্যালু হিসেবে ব্যবহার করা যায়।	4 এটি একটি ধ্রুবক মান, সুতরাং এটি একটি আরভ্যালু।	ঠিক আছে
5	b এটি একটি ভ্যারিয়েবল যার একটি মেমোরি লোকেশন আছে এবং একটি মানও আছে। সুতরাং এটি এলভ্যালু হিসেবে ব্যবহার করা	2 এটি একটি ধ্রুবক মান, সুতরাং এটি একটি আরভ্যালু	ঠিক আছে

১৬১

কম্পিউটার প্রোগ্রামিং – দ্বিতীয় খণ্ড

	যায়।		
6	4 এটি একটি ধ্রবক মান, সুতরাং এটি একটি আরভ্যালু।	a এটি একটি ভ্যারিয়েবল যার একটি মেমোরি লোকেশন আছে এবং এর একটি মানও আছে। সুতরাং এটি এলভ্যালু এবং আরভ্যালু দুইভাবেই ব্যবহার করা যায়।	ঠিক নেই
7	a+2 a একটি ভ্যারিয়েবল যার মেমোরি লোকেশন ও মান আছে। a+2 একটি এক্সপ্রেশন যার কেবল মান আছে, একে এলভ্যালু হিসেবে ব্যবহার করা যায় না।	3 এটি একটি ধ্রবক মান যাকে আরভ্যালু হিসেবে ব্যবহার করা যায়।	ঠিক নেই
8	b এটি একটি ভ্যারিয়েবল যার একটি মেমোরি লোকেশন আছে এবং একটি মানও আছে। সুতরাং এটি এলভ্যালু হিসেবে ব্যবহার করা যায়।	a+2 a একটি ভ্যারিয়েবল যার মেমোরি লোকেশন ও মান আছে। a+2 একটি এক্সপ্রেশন যার মান আছে, একে আরভ্যালু হিসেবে ব্যবহার করা যায়।	ঠিক আছে
9	a+b a ও b দুটি ভ্যারিয়েবল যাদের মেমোরি লোকেশন ও মান আছে। a+b একটি এক্সপ্রেশন যার মান আছে, একে এলভ্যালু হিসেবে ব্যবহার করা যায়।	a-b a ও b দুটি ভ্যারিয়েবল যাদের মেমোরি লোকেশন ও মান আছে। a-b একটি এক্সপ্রেশন যার মান আছে, একে আরভ্যালু হিসেবে ব্যবহার করা যায়।	ঠিক নেই
10	c এটি একটি ভ্যারিয়েবল যার একটি মেমোরি লোকেশন আছে এবং একটি মানও আছে। সুতরাং এটি এলভ্যালু হিসেবে ব্যবহার করা	red এটি একটি ধ্রবক মান যাকে আরভ্যালু হিসেবে ব্যবহার করা যায়।	ঠিক আছে

১৬২

কম্পিউটার প্রোগ্রামিং – দ্বিতীয় খণ্ড

	যায়।		
11	green এটি একটি ধ্রবক মান যাকে এলভ্যালু হিসেবে ব্যবহার করা যায়।	c এটি একটি ভ্যারিয়েবল যার মেমোরি লোকেশন ও মান আছে একে এলভ্যালু ও আরভ্যালু দুইভাবেই ব্যবহার করা যায়।	ঠিক নেই

তাহলে তোমরা কী বুঝতে পারলে? এলভ্যালু হচ্ছে সেই সব এক্সপ্রেশন যাদের একটি মেমোরি লোকেশন থাকে এবং সেই লোকেশনে কোনো একটি মানকে অ্যাসাইন করা যায়। এদের মেহেতু মেমোরি লোকেশন থাকে সেহেতু এদের একটি মানও থাকে। সেটি অর্থবোধক কোনো মান হতে পারে, অথবা হতে পারে গারবেজ (garbage) মান। অপরপক্ষে আরভ্যালু হচ্ছে সেই সব এক্সপ্রেশন যাদের একটি মান থাকে। এদেরও একটি মেমোরি লোকেশন থাকে, কেননা, সব মানই তো কোনো না কোনো মেমোরি লোকেশনেই আছে। তবে এই মেমোরি লোকেশন কোনো ভ্যারিয়েবলকে নির্দেশ করে না এবং এখানে কোনো মান অ্যাসাইন করা যায় না।

১৬৩

অধ্যায় ১০ - প্রোগ্রাম ডিবাগিং

১০.১ - ডিবাগিং কী?

ডিবাগিং শব্দটার সঙ্গে তোমরা অনেকেই হ্যাতো পরিচিত। ডিবাগিং বিষয়টা আসলে কী? ধৰা যাক, তুমি একটি প্রোগ্রাম লিখেছ, সেটি ঠিকঠাক মতো কম্পাইলও হয়েছে। অর্থাৎ, কোডের সিনট্যাক্সে কোনো তুল নেই। কিন্তু, তুমি যেমনটি চাচ্ছ, প্রোগ্রামটি ঠিক সেরকমভাবে কাজ করছে না। হ্যাতো যেখানে ৫ আউটপুট দেওয়ার কথা সেখানে ১৫ আউটপুট দিচ্ছে। তাহলে বুবুতে হবে, তুমি প্রোগ্রামে যেসব পাণিতিক বা লজিক্যাল অপারেশনগুলো করেছ তাতে তুমি যেমনটি ভাবছ বাগ। তোমাকে এখন খুঁজে বের করতে হবে, তোমার প্রোগ্রামের কোনো একটি লাইনে তুমি এমন কোনো কাজ করেছ যা তোমার ফলাফলকে পালটে দিচ্ছে। এই ত্রুটি খোঁজ করার কাজটিকেই বলে ডিবাগিং।

১০.২ - সাধারণ ডিবাগিং

ডিবাগিংয়ের প্রাথমিক উপায় কী? সেটি কিন্তু আমরা ইতিমধ্যে দেখেছি অনেকবার। সেটি হচ্ছে কেমন আছে। চলো একটি উদাহরণ দেখি।

ধর, আমি একটি প্রোগ্রাম লিখলাম যেটি কিছু সংখ্যা ইনপুট নেবে ও তাদের গড় প্রিন্ট করবে।

```

1 #include <stdio.h>
2
3 int add(int a, int b)
4 {
5     return a * b;
6 }
7
8 int main()
9 {
10    int length, i, sum = 0, tmp;
11    printf("Enter number of integers: ");
12    scanf("%d", &length);

```

168

```

14 for(i = 0; i < length; i++)
15 {
16     printf("Enter Number %d: ", i+1);
17     scanf("%d", &tmp);
18     sum = add(sum, tmp);
19 }
20
21
22
23
24
25 }

printf("The average is %lf\n", (double)sum / length);

return 0;
}

```

প্রোগ্রাম ১০-১

এটি চালানোর পরে আমি এরকম আউটপুট পেলাম:

```

Enter number of integers: 5
Enter Number 1:45
Enter Number 2:27
Enter Number 3:98
Enter Number 4:112
Enter Number 5:97
The average is 0.000000

```

কী আশ্চর্য! ৫টি সংখ্যা ইনপুট দিলাম, অথচ এদের গড় শূন্য হলো কীভাবে? নিশ্চয়ই প্রোগ্রামে কী আশ্চর্য! ৫টি সংখ্যা ইনপুট দিলাম, অথচ এদের গড় শূন্য হলো কীভাবে? নিশ্চয়ই প্রোগ্রামে কী আশ্চর্য! ৫টি সংখ্যা ইনপুট দিলাম, অথচ এদের গড় শূন্য হলো কীভাবে? নিশ্চয়ই প্রোগ্রামে কী আশ্চর্য! ৫টি সংখ্যা ইনপুট দিলাম, অথচ এদের গড় শূন্য হলো কীভাবে? নিশ্চয়ই প্রোগ্রামে কী আশ্চর্য! ৫টি সংখ্যা ইনপুট দিলাম, অথচ এদের গড় শূন্য হলো কীভাবে? নিশ্চয়ই প্রোগ্রামে কী আশ্চর্য! ৫টি সংখ্যা ইনপুট দিলাম, অথচ এদের গড় শূন্য হলো কীভাবে? নিশ্চয়ই প্রোগ্রামে কী আশ্চর্য! ৫টি সংখ্যা ইনপুট দিলাম, অথচ এদের গড় শূন্য হলো কীভাবে? নিশ্চয়ই প্রোগ্রামে কী আশ্চর্য!

```

1 #include <stdio.h>
2
3 int add(int a, int b)
4 {
5     return a * b;
6 }
7
8 int main()
9 {
10    int length, i, sum = 0, tmp;
11    printf("Enter number of integers: ");
12

```

165

```

13     scanf("%d", &length);
14
15     for(i = 0; i < length; i++)
16     {
17         printf("Enter Number %d: ", i+1);
18         scanf("%d", &tmp);
19         printf("Number %d = %d\n", i+1, tmp);
20         sum = add(sum, tmp);
21     }
22
23     printf("The average is %lf\n", (double)sum / length);
24
25     return 0;
26 }
```

প্রোগ্রাম ১০-২

এবারে আউটপুট এল এরকম:

```

Enter number of integers: 5
Enter Number 1:45
Number 1=45
Enter Number 2:27
Number 2=27
Enter Number 3:98
Number 3=98
Enter Number 4:112
Number 4=112
Enter Number 5:97
Number 5=97
The average is 0.000000
```

তার মানে বোধা গেল সংখ্যা ইনপুট নেওয়ার কাজ ঠিকমতোই হচ্ছে। তাহলে এখন চলো দেখি যোগফল বের করার কাজটি ঠিকমতো হচ্ছে কি না। কোডটিকে এবার এভাবে পরিবর্তন করলাম:

```

1 #include <stdio.h>
2
3 int add(int a, int b)
4 {
5     return a * b;
6 }
7
```

```

8 int main()
9 {
10     int length, i, sum = 0, tmp;
11
12     printf("Enter number of integers: ");
13     scanf("%d", &length);
14
15     for(i = 0; i < length; i++)
16     {
17         printf("Enter Number %d: ", i+1);
18         scanf("%d", &tmp);
19         sum = add(sum, tmp);
20         printf("Current Number is = %d, Current Sum = %d\n",
21                tmp, sum);
22
23     printf("The average is %lf\n", (double)sum / length);
24
25     return 0;
26 }
```

প্রোগ্রাম ১০-৩

এবারে আউটপুট পেলাম এরকম:

```

Enter number of integers: 5
Enter Number 1: 45
Current Number is = 45, Current Sum = 0
Enter Number 2: 27
Current Number is = 27, Current Sum = 0
Enter Number 3: 98
Current Number is = 98, Current Sum = 0
Enter Number 4: 112
Current Number is = 112, Current Sum = 0
Enter Number 5: 97
Current Number is = 97, Current Sum = 0
The average is 0.000000
```

তাহলে দেখা যাচ্ছে আমাদের যোগফল প্রতিবার শূন্য থাকছে। এখন যদি আমরা যোগ করার ফাংশনটি দেখি তাহলে দেখতে পাব যে, আমরা add ফাংশনের মধ্যে ভুল করে যোগ চিহ্নের বদলে গুণ চিহ্ন ব্যবহার করেছিলাম। সেটি যদি ঠিক করে ফেলি তাহলেই আমাদের কোড ঠিকমতো কাজ করবে।

```

1 Enter number of integers: 5
2 Enter Number 1: 45
3 Current Number is = 45, Current Sum = 45
4 Enter Number 2: 27
5 Current Number is = 27, Current Sum = 72
6 Enter Number 3: 98
7 Current Number is = 98, Current Sum = 170
8 Enter Number 4: 112
9 Current Number is = 112, Current Sum = 282
10 Enter Number 5: 97
11 Current Number is = 97, Current Sum = 379
12 The average is 75.800000

```

এখন আমরা অপ্রয়োজনীয় প্রিন্ট ফাংশনগুলো সরিয়ে দিতে পারি। বারবার এভাবে প্রিন্ট লেখা আবার সরিয়ে ফেলাটা খানিকটা পরিশ্রমের কাজ তাই তোমরা চাইলে নিচের পদ্ধতিও ব্যবহার করতে পারো:

```

1 #include <stdio.h>
2 #define DEBUG 1
3
4 int add(int a, int b)
5 {
6     return a + b;
7 }
8
9 int main()
10 {
11     int length, i, sum = 0, tmp;
12
13     printf("Enter number of integers: ");
14     scanf("%d", &length);
15     if(DEBUG) printf("----\nDEBUG\nNumber of integers:
16 %d\nENDDEBUG\n----\n", length);
17
18     for(i = 0; i < length; i++)
19     {
20         printf("Enter Number %d: ", i+1);
21         scanf("%d", &tmp);
22         if(DEBUG) printf("----\nDEBUG\nNumber %d:
23 %d\nENDDEBUG\n----\n", i+1, tmp);
24         sum = add(sum, tmp);
25         if(DEBUG) printf("----\nDEBUG\nCurrent Sum:
26 %d\nENDDEBUG\n----\n", sum);
27
28     }
29 }

```

```

24     }
25     printf("The average is %lf\n", (double)sum / length);
26
27     return 0;
28 }
29

```

শ্রেণীম ১০-৮

এতে করে তুমি যদি 2 নং লাইনের DEBUG এর মান 0 করে দাও, তাহলে আর ডিবাগ মেসেজগুলো প্রিন্ট হবে না। আর যদি সেটি 1 করে দাও তাহলে আবার ডিবাগ মেসেজগুলো প্রিন্ট হবে। সফটওয়্যার ডেভেলপমেন্ট ও ওয়েব ডেভেলপমেন্টের ক্ষেত্রেও আমরা এখন পদ্ধতি হবে। নিজেরা সফটওয়্যার টেস্টিং করার সময় ডিবাগ অন করে দিয়ে চালাই, কোনো ব্যবহার করি। নিজেরা সফটওয়্যার টেস্টিং করার সময় ডিবাগ অন করে দিয়ে চালাই, কোনো ভুল হুটি হলো কি না দেখার জন্য আর বাইরের ব্যবহারকারীদের ব্যবহারের সময় ডিবাগ অফ করে দেই।

এই তো দেখলাম একটি পদ্ধতি। যদি তোমার কোডটি ২৫-৩০ লাইন না হয়ে, ৭০০-৮০০ লাইনের হয় এবং সেখানে যদি ১২-১৩ টি ফাংশন থাকে এবং ইনপুট যদি এত সরল না হয়ে আরও জটিল হয় তবে কিন্তু প্রতিটি লাইনে এভাবে প্রিন্ট করে দেখা প্রয়োজন হবে। তাই প্রোগ্রাম ডিবাগ করার আরও একটি উপায় আছে।

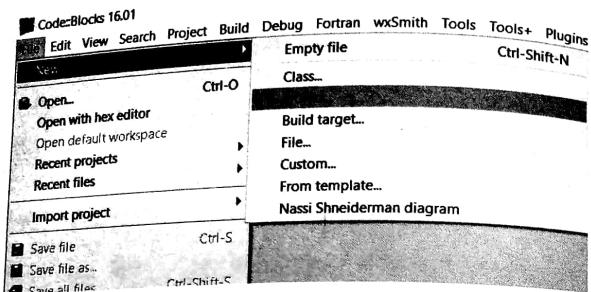
১০.৩ – কোডরুক্সে ডিবাগিং

আমরা যদি লাইনে লাইনে প্রিন্ট না করে লাইন-বাই-লাইন প্রোগ্রামটি চালিয়ে দেখতে পারি কোন লাইনে কেন ভ্যারিয়েবলের মান কেমন তাহলেই কিন্তু সহজে আমরা প্রোগ্রামের বাগ খুঁজে বের করতে পারব। অনেক সময় ভালো আইডিয়ের সঙ্গে একটি ডিবাগার টুল দেওয়া থাকে যেটি করে দিয়ে প্রোগ্রাম ডিবাগ করা যায়। আমরা এখন দেখব কোডরুক্সের ডিবাগিং টুল ব্যবহার করে দিয়ে প্রোগ্রাম ডিবাগ করা যায়। আগের অংশে আমরা যেই প্রোগ্রামটি লিখেছি, সেটিই আবার কীভাবে প্রোগ্রাম ডিবাগ করা যায়। আগের অংশে আমরা যেই প্রোগ্রামটি লিখেছি, সেটিই আবার এখানে উদাহরণ হিসেবে ব্যবহার করব।

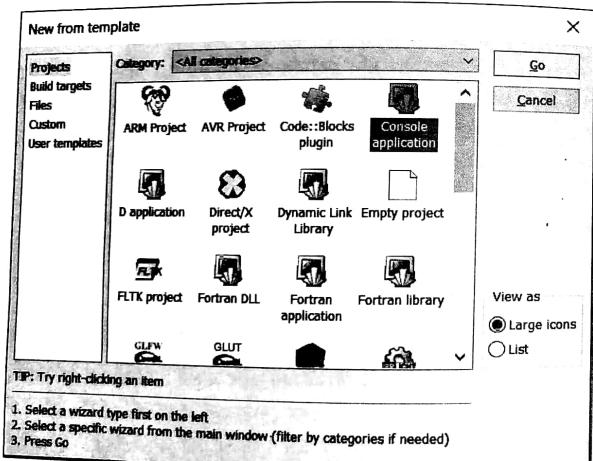
১। অথবে আমাদের কোডরুক্সে একটি প্রজেক্ট তৈরি করতে হবে। তার জন্য প্রথমে আমরা ক্লিক করব ফাইল (File) → নিউ (New) → প্রজেক্ট (Project) (ছবি ২২)। এরপর একটি উইন্ডো করব ফাইল (File) → সিলেক্ট করতে হবে কনসোল অ্যাপ্লিকেশন (Console Application) (ছবি ২৩)। এরপরে আরেকটি উইন্ডো আসবে, সেখান থেকে নেক্সট (Next) Application (ছবি ২৪)। এরপরে আরেকটি উইন্ডোতে সি এবং সি প্লাস প্লাস নামে দুটি অপশন থাকবে, সেখান থাটনে ক্লিক করব। পরবর্তী উইন্ডোতে সি এবং সি প্লাস প্লাস নামে দুটি অপশন থাকবে, সেখান থেকে আমরা সি (C) সিলেক্ট করে আবারও নেক্সট বাটনে ক্লিক করব। (ছবি ২৪)।

১৬৯

কম্পিউটার প্রোগ্রামিং – বিতীয় খণ্ড



ছবি ২২



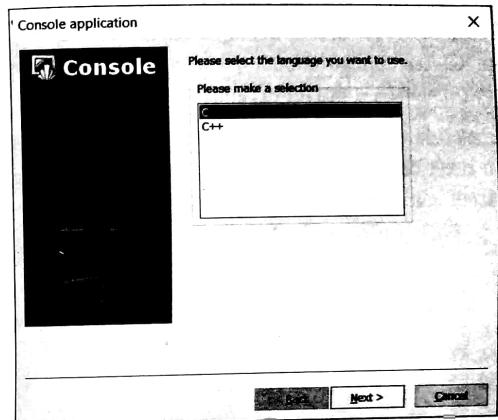
ছবি ২৩

২। এরপরের উইন্ডো একটি ফর্ম দেখানো হবে (ছবি ২৫)। এখান থেকে আমরা প্রজেক্টের টাইপেল ফিল্ডে লিখব আমাদের প্রজেক্টের নাম। আমি এখানে লিখেছি: "Average"। বিতীয় ফিল্ডে আমরা লিখে দিব কোন ডারেক্টরি আমাদের প্রজেক্টটি সেভ হবে। এখানে আমি দিয়েছি: "D:\\"। এর পরের একটি ফিল্ড হচ্ছে প্রজেক্ট ফাইলনেম ও রেজাল্টিং ফাইলনেম। এই দুটি ফিল্ড

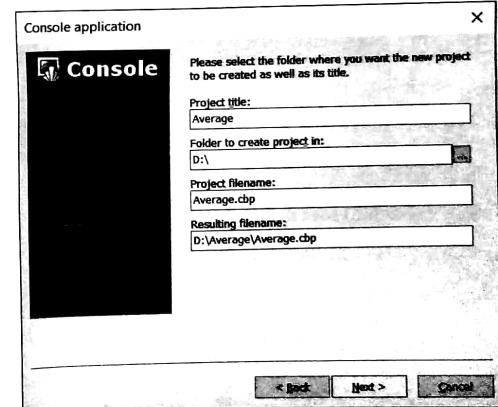
১৭০

কম্পিউটার প্রোগ্রামিং – বিতীয় খণ্ড

আপনাআপনি তৈরি হবে। এ দুটি নিয়ে আপাতত চিন্তা করতে হবে না। এবারে নেক্সট বাটনে ক্লিক কর।



ছবি ২৪



ছবি ২৫

১৭১

কম্পিউটার প্রোগ্রামিং - বিতীয় খণ্ড

৩। এরপরে উইন্ডোতে প্রথমে থাকবে একটি ড্রপডাউন মেনু যেখানে অনেক রকম সি / সি++ কম্পাইলারের নাম লিস্ট করা থাকবে। চাইলে এর যেকোনো একটি কম্পাইলার ব্যবহার করা যায়। আমরা কম্পিউটারে আমি ব্যবহার করছি GNU GCC Compiler। এর পরে থাকবে দুটি অপশন: একটিতে সেখা থাকবে "Create Debug Configuration" এবং পরেরটিতে সেখা থাকবে "Create Release Configuration"। দুটি অপশনই ডিফল্টভাবে চেক করা থাকবে। আমরা এখানে কোনো পরিবর্তন না করে ফিনিশ (Finish) বাটনে ক্লিক করব।

৪। আমাদের প্রজেক্ট তৈরি করা হয়ে গেল। এবার আমরা কম্পিউটারের নির্ধারিত ফোল্ডার (যেই ফোল্ডারে আমরা প্রজেক্ট তৈরি করেছি) "D:\\" এ গেলে দেখতে পাব Average নামে একটি ফোল্ডার তৈরি হয়েছে এবং সেখানে main.c নামে একটি ফাইল তৈরি হয়েছে। এবারে আমরা ফাইল (File) → ওপেন (Open) মেনু থেকে main.c ফাইলটি ওপেন করে নিচের কোডটি টাইপ করব।

```

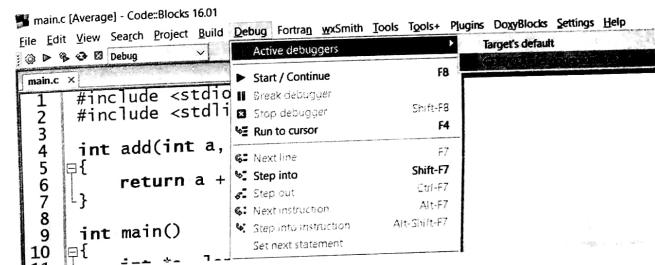
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int add(int a, int b)
5 {
6     int s = a + b;
7     return s;
8 }
9
10 int main()
11 {
12     int *array, length, i, sum = 0;
13
14     printf("Enter number of integers: ");
15     scanf("%d", &length);
16     array = (int*)malloc(sizeof(int)*length);
17
18     for(i = 0; i < length; i++)
19     {
20         printf("Enter Number %d: ", i+1);
21         scanf("%d", &array [i]);
22         sum = add(sum, array [i]);
23     }
24
25     printf("The average is %lf\n", (double)sum / length);
26 }
```

কম্পিউটার প্রোগ্রামিং - বিতীয় খণ্ড

27 }
28 }

প্রোগ্রাম ১০-৫

৫। কোডটি প্রথমে কম্পাইল ও রান করে দেখো ঠিকমতো চলছে কি না। এরপরে আমরা একে ডিবাগিং করব। কোডকসের মেনুবারে দেখবে ডিবাগ (Debug) নামে একটি মেনু আছে। সেখানে প্রথমে থাকবে Active debuggers নামে একটি সাবমেনু। সেখান থেকে আমরা সিলেক্ট করাব GDB/CDB Debugger (ছবি ২৬)।



ছবি ২৬

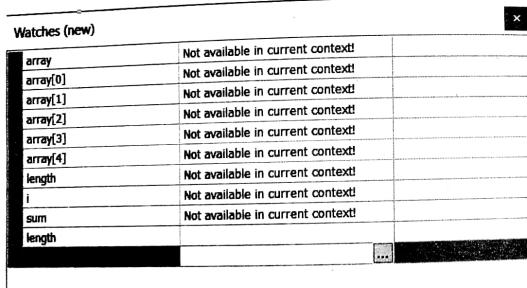
৬। ডিবাগ মেনুর একটি অংশে থাকে Start/ Continue, Break Debugger, Stop Debugger, Run to cursor। পরের অংশে থাকে Next Line, Step into, Step out ইত্যাদি অপশন। ডিবাগিং শুরু করার আগে প্রথমে আমরা ডিবাগ মেনু থেকে Watches উইজেন্ট অপশন। ডিবাগিং শুরু করতে হবে ডিবাগ (Debug) → Debugging Windows → চালাব। এর জন্য ক্লিক করতে হবে ডিবাগ (Debug) → Debugging Windows → Watches। এরপর দেখবে নতুন একটি উইজেন্ট আসবে। সেখানে আমরা আমাদের বিভিন্ন Watches। এরপর দেখবে নতুন একটি উইজেন্ট আসবে। সেখানে আমরা আমাদের ভ্যারিয়েবল যাদের মান আমরা পর্যবেক্ষণ করতে চাই, সেগুলো লিখব। এখানে আমি সবগুলো ভ্যারিয়েবল যাদের মান আমরা পর্যবেক্ষণ করতে চাই, সেগুলো লিখব। এখানে আমি সবগুলো ভ্যারিয়েবলের নামই দিচ্ছি। Watches উইজেন্ট আমরা একটি করে ভ্যারিয়েবলের নামই দিচ্ছি।

৭। এবারে আমরা ডিবাগিং শুরু করব। প্রথমে আমরা আমাদের প্রোগ্রামের 12 নম্বর লাইনে ক্লিক করব, যে লাইনে আমরা সবগুলো ভ্যারিয়েবল ডিক্রেয়ার করেছি। এরপরে ডিবাগ → Run to Cursor মেন্যুতে ক্লিক করব। এরপরে তোমরা দেখবে ঠিক 12 নম্বর লাইনের পাশে একটি Cursor মেন্যুতে ক্লিক করব। এরপরে তোমরা দেখবে ঠিক 12 নম্বর লাইনের পাশে একটি Cursor মেন্যুতে ক্লিক করব।

১৭৩

কম্পিউটার প্রোগ্রামিং - বিতীয় খণ্ড

ত্রিভুজাক্ষর চিহ্ন দেখা যাবে। এটি নির্দেশ করে আমরা এখন কোন লাইনে আছি। আরও লক্ষ কর Watches উইজেটটি খানিকটা পরিবর্তিত হয়ে গেছে (ছবি ২৮)। array ভ্যারিয়েবলটির মান দেখা যাচ্ছে ০x2a। যেহেতু এটি একটি পয়েন্টার, এই মান দিয়ে তার মেমোরি লোকেশনটি বোঝাচ্ছে। আরও লক্ষ কর **length, sum, i** ইত্যাদি ভ্যারিয়েবলে কিছু মান দেখা যাচ্ছে। আমরা তো এখনো এই ভ্যারিয়েবলগুলোতে কোনো মান দেইনি, তাহলে এই মানগুলো কোথা থেকে এল? এই মানগুলো হচ্ছে এই ভ্যারিয়েবলগুলোর মেমোরি লোকেশনে আগে থেকেই যে মান ছিল সেই মান। আমাদের প্রোগ্রামের সঙ্গে এই মানের কোনো সম্পর্ক নেই। আমাদের কাছে এগুলো হচ্ছে গারবেজ মান। **array[0], array[1]** এসব ভ্যারিয়েবলে দেখো কোনো মান দেখাচ্ছে না। কারণ আমরা এখনো array পয়েন্টারের জন্য কোনো মেমোরি অ্যালোকেশন করিনি।



ছবি ২৭

```

10 int s = a + b;
11 return s;
}
12
13 int main()
14 {
15     int *array, length, i, sum = 0;
16     printf("Enter number of integers: ");
17     scanf("%d", &length);
18     array = (int*)malloc(sizeof(int)*length);
19     for(i = 0; i < length; i++)
20     {
21         printf("Enter Number %d: ", i+1);
22         scanf("%d", &array[i]);
23         sum = add(sum, array[i]);
24     }
25     printf("The average is %lf\n", (double)sum);
26     return 0;
}

```

ছবি ২৮

কম্পিউটার প্রোগ্রামিং - বিতীয় খণ্ড

৮। এবারে আমরা পরের লাইন এক্সেকিউট করব। এর জন্য আমাদের ক্লিক করতে হবে ডিবাগ → Next Line। এবারে দেখবে ত্রিভুজ চিহ্নটি 14 নম্বর লাইনে যেখানে প্রিন্ট ফাংশন আছে সেখানে চলে এসেছে। আবারও Next Line ক্লিক করি। এবারে দেখবে প্রিন্ট ফাংশনের কথাটি কমান্ড ক্লিনে প্রিন্ট হয়েছে এবং সেই সঙ্গে ত্রিভুজ চিহ্ন পরের scanf লাইনে (15 নম্বর লাইনে) নির্দেশ করছে। আবারও Next Line ক্লিক করে আমরা কমান্ড ক্লিনে ইনপুট দেব ৫। এবারে দেখবে length ভ্যারিয়েবলের মান এখন পরিবর্তন হয়ে 5 হয়ে গেছে। অর্থাৎ, আমরা যেই ইনপুট দিলাম তা length ভ্যারিয়েবলে অ্যাসাইন হয়েছে। আমাদের ত্রিভুজ চিহ্ন এখন আছে 16 নম্বর লাইনে। এভাবে আবারও Next Line দেই। এবারে দেখো array ভ্যারিয়েবলে মান পরিবর্তন হয়ে গেল। আমরা যখন **malloc** ফাংশনটিকে **sizeof(int)*length** দিয়ে কল করলাম, ফাংশনটি মেমোরি থেকে 5 টি ইন্টিজার রাখার মতো জায়গা বের করে তার মেমোরি ঠিকানাটি array ভ্যারিয়েবলে অ্যাসাইন করেছে। সেই সঙ্গে দেখো **array[0]**, **array[1]** ইত্যাদি ভ্যারিয়েবলগুলো এখন গারবেজ মান দেখাচ্ছে (ছবি ২৯)।

Watches (new)	
Function arguments	
Locals	
array	0xb41680
length	5
i	34
sum	0
array	(int *) 0xb41680
array[0]	-1163005939
array[1]	-1163005939
array[2]	-1163005939
array[3]	-1163005939
array[4]	-1163005939
length	5
i	34
sum	0
length	5

ছবি ২৯

৯। এখন আমরা চলে এসেছি 18 নম্বর লাইনে for ল্যাপের শুরুতে। এবারে আমরা আরেকবার Next Line ক্লিক করলে i এর মান সেট হয়ে যাবে ০ এবং আমরা চলে যাব 20 নম্বর লাইনের প্রিন্ট ফাংশনে। আবারও Next Line ক্লিক করে আমরা scanf এর লাইনে যাব এবং আবারও Next Line ক্লিক করে কমান্ড ক্লিনে ইনপুট দিব 45। এবারে দেখো array[0] এর মান 45 সেট হয়ে পিয়েছে।

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

১০। ২২ নম্বর লাইনে এসে আমরা আর Next Line ক্লিক করব না। এই লাইনে আমরা যোগ করার কাজটি করেছি add মানের ফাংশনে। আমরা দেখতে চাই ফাংশনটি কীভাবে কাজ করছে। আমরা এই ফাংশনটির ভেতরে চুক্তি দেখতে চাই।

১১। এর জন্য আমরা প্রথমে ক্লিক করব ডিবাগ → Debugging Windows → Call Stack। তোমাদেরকে অফিসিয়াল অধ্যায়ে বলেছিলাম যখন নতুন ফাংশন কল হয় তখন এই মহুর্তে যে ফাংশনের যত নম্বর লাইনে আছি সেসব তথ্য ও সব লোকাল ভ্যারিয়েবলের মান স্ট্যাক সেগমেন্টে সংরক্ষণ করে রাখা হয়। এই কল স্ট্যাক উইজেটে আমরা এই তথ্য দেখতে পাব। এখন কল স্ট্যাকে শুধুমাত্র একটি লাইন রয়েছে, যেখানে লেখা রয়েছে আমরা main ফাংশনের 22 নম্বর লাইনে রয়েছি। এবাবে আমরা ক্লিক করব ডিবাগ → Step Into মেনুতে।

Call stack					
Nr	Address	Function	File	Line	
1	0x401404	main()	D:\Average\main.c	22	

Function arguments	
Locals	
s	6356708
array	Not available in current context
array[0]	Not available in current context
array[1]	Not available in current context
array[2]	Not available in current context
array[3]	Not available in current context
array[4]	Not available in current context
length	Not available in current context
i	Not available in current context
sum	Not available in current context
length	Not available in current context

ছবি ৩০

১২। ছবি ৩০ এ দেখো আমরা এখন add ফাংশনের মধ্যে 6 নম্বর লাইনে চলে এসেছি। Watches উইজেটে add ফাংশনের লোকাল ভ্যারিয়েবল a ও b এর মান দেখা যাচ্ছে যথাক্রমে 0 ও 45। অন্যান্য ভ্যারিয়েবলের মানগুলো এখন আর দেখা যাচ্ছে না, কারণ এই ফাংশনের মধ্যে থেকে আমরা main ফাংশনের ভ্যারিয়েবলগুলো আয়কসেস করতে পারছি না। কারণ ওই ভ্যারিয়েবলগুলো ক্লোপ ওই main ফাংশনের মধ্যেই সীমাবদ্ধ। কল স্ট্যাক উইজেটে দেখো এখন দুটি লাইন আছে। প্রথম লাইনটি নির্দেশ করে আমরা কোন ফাংশনের কোন লাইনে আছি। বিত্তীয় লাইন নির্দেশ করে আমরা কোন ফাংশনের কোন লাইন থেকে এখানে এসেছি। যদি আরও গভীরে যাই তাহলে এখানে আরও বেশি লাইন থাকবে। তোমরা কোনো রিকার্সিভ প্রোগ্রাম লিখে ডিবাগ করে দেখলে কল স্ট্যাকটি আরও ভালোভাবে বুঝতে পারবে।

১৩। এভাবে আমরা যদি প্রতিবার Next Line কমান্ড দিতে থাকি আমাদের প্রোগ্রাম এক এক লাইন করে এক্সেকিউট হবে এবং আমরা দেখতে পাব কখন কোন ভ্যারিয়েবলে মান কত হচ্ছে

১৭৬

কম্পিউটার প্রোগ্রামিং - বিত্তীয় খণ্ড

এবং কীভাবে পরিবর্তন হচ্ছে। যদি কোনো ফাংশনের ভেতরে চুক্তি চাই তাহলে Step Into এবং কোনো ফাংশনের বাইরে চলে আসতে চাইলে Step Out কমান্ড দিব। তোমরা এবার বাকিটা নিজেরা কর এবং দেখো sum ভ্যারিয়েবলে মান প্রতিবারে কীভাবে আপডেট হচ্ছে। এই হলো আমার কম্পিউটারের পাওয়া সর্বশেষ অবস্থা (ছবি ৩১)।

Function arguments	
Locals	
array	0xb41680
length	5
sum	379
array[0]	(int *) 0xb41680
array[1]	45
array[2]	27
array[3]	98
array[4]	112
length	97
i	5
sum	379
length	5

ছবি ৩১

এই হলো মোটামুটি প্রাথমিক ডিবাগিং। বড় বড় ফাইল ও ফাংশন ডিবাগিংয়ের জন্য আরও অনেক অপশন রয়েছে। আমরা সম্পূর্ণ ফাইল ডিবাগ না করে বিশেষ বিশেষ লাইন ডিবাগ করার জন্য breakpoint ব্যবহার করতে পারি। সেটি কীভাবে করতে হয়, আশা করি সেটি তোমরা নিজেরা একটু ঘাঁটাঘাঁটি করলেই শিখে ফেলবে।

১৭৭

অধ্যায় ১১ – যেতে হবে বহুদুর

বইটি যদি তুমি ধূমি ধূমী ধূমী বাইকভাবে পড়ে এতদূর এসে থাক, তাহলে তোমাকে অভিনন্দন! আমার দৃঢ় বিশ্বাস, প্রোগ্রামিং জিনিসটি তুমি এখন থেকে আরও বেশি উপভোগ করবে। বইটি যদি কেবল একবার পড়ে থাক, তাহলে তোমার প্রথম কাজ হবে বইটি আবার পড়া।

যারা আমার কম্পিউটার প্রোগ্রামিং ১ম খণ্ড, ৫২টি প্রোগ্রামিং সমস্যা ও সমাধান, কম্পিউটার প্রোগ্রামিং ২য় গুণ (অর্থাৎ এই বইটি) - ভিনটি বইই মনোযোগ দিয়ে বুরো বুরো পড়েছ, সবগুলো উন্নাসন দেখে দেখে টাইপ করে কম্পাইল ও রান করেছে এবং যেসব প্রোগ্রামিং সমস্যা দেওয়া হয়েছে, সেগুলো সমাধানের চেষ্টা করেছে, তাদের ইতিমধ্যে প্রোগ্রামিংয়ের বেসিক শক্ত হয়ে গিয়েছে। এখন তুমি দৃঢ় জিনিস শিখতে পার, সেগুলো হচ্ছে –

অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং - আমরা এখন পর্যন্ত যতটুকু প্রোগ্রামিং শিখেছি, সেটুকু হচ্ছে স্ট্রাকচার প্রোগ্রামিং। অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং হচ্ছে প্রোগ্রাম অ্যানলাইসিস ও লেখার আরেক ধরনের পদ্ধতি। এই পদ্ধতি শেখার জন্য তোমরা সি প্লাস প্লাস (C++), কিংবা জাভা (Java) শিখে। এগুলো হচ্ছে প্রোগ্রামিং ভাষা, যা দিয়ে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং করা যায়। এছাড়াও অনেক ভাষা রয়েছে, কিন্তু এগুলোই বেশি প্রচলিত। যেকোনো একটি ভাষা শিখলেই চলবে। আর ভুল গেলে চলবে না যে, প্রোগ্রামিং ভাষা শেখার সঙ্গে সঙ্গে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিংয়ের ধারণা খুব ভালোভাবে বুরো নিতে হবে। বাংলা ভাষায় এসব বিষয়ে আলো কেনো বই আমার জানা নেই, তাই তোমরা ইংরেজি বইয়ের সাহায্য নিতে পার। যেকোনো দুই বা তিনটি বই কিনে একটি একটি করে পড়ে ফেলবে। আর বাংলা ভাষায় এসব বিষয়ে বই ও অনলাইন কোর্সের জন্য খোঁজবৰ করতে পার যথাক্রমে দ্বিমিক প্রকাশনী (<http://dimik.pub>) ও দ্বিমিক কম্পিউটিংয়ের (<http://dimikcomputing.com>) ওয়েবসাইট। অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং শেখার জন্য ছয় মাস থেকে এক বছর নিরলস পরিশ্রম ও অধ্যাবসায়ের প্রয়োজন হবে। প্রোগ্রামিং প্রতিযোগিতায় অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিংয়ের তেমন গুরুত্ব না ধাকলেও দক্ষ সফটওয়্যার প্রকৌশলী হওয়ার পূর্বশর্ত হচ্ছে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিংয়ে দক্ষতা।

ডেটা স্ট্রাকচার ও অ্যালগরিদম - তুমি এখন পর্যন্ত যতটুকু প্রোগ্রামিং শিখেছি, তাতে করে তুমি ডেটা স্ট্রাকচার ও অ্যালগরিদম শেখার জন্য তৈরি হয়ে গিয়েছ। তবে তোমাকে একটুখানি বিচ্ছিন্ন গণিত (ডিসক্রিট ম্যাথ) শিখে নিতে হবে। এজন্য দ্বিমিক কম্পিউটিংয়ের ওয়েবসাইট থেকে বিচ্ছিন্ন গণিতের ওপর ফি অনলাইন কোস্টি করে নিতে পার। তারপরে বাজারে যতগুলো বই পাওয়া যায়, সবগুলো জোগাড় করে ডেটা স্ট্রাকচার ও অ্যালগরিদম শেখা শুরু করে দাও। মনে

মনে ধরে রাখবে যে, এটি প্রায় দেড় থেকে দুই বছরের প্রজেক্ট। ডেটা স্ট্রাকচার ও অ্যালগরিদম তোমাকে যে কেবল প্রোগ্রামিং প্রতিযোগিতায় ভালো করতে সাহায্য করবে তা নয়, সেই সঙ্গে একজন দক্ষ সফটওয়্যার প্রকৌশলী হিসেবে গড়ে উঠতেও সহায়তা করবে।

যতদিনে তুমি ওপরের জিনিসগুলো শিখবে, ততদিনে তুমি জেনে যাবে যে, এরপর তোমাকে কী করতে হবে, কী পড়তে হবে। তাই এখনই অঙ্গীর হওয়ার কিছু নেই। তবে আমার ওয়েবসাইটে (<http://subeen.com>) তুমি নিয়মিত দুঁ মারতে পার বিভিন্ন প্রযুক্তিবিষয়ক লেখা ও পরামর্শমূলক (গাইডলাইন) লেখা পড়ার জন্য।

প্রোগ্রামিং মোটেও সহজ কোনো কাজ নয়। যদি খুব সহজ হতো, তাহলে সারা পৃথিবীজুড়ে দক্ষ প্রোগ্রামারদের এত চাহিদা থাকত না। তবে আশা কথা হচ্ছে, প্রোগ্রামিং শিখতে খুব বেশি মেধার প্রয়োজন নেই। তোমাদের প্রোগ্রামিং শেখার পেছনে যথেষ্ট শ্রম ও সময় দিতে হবে। যে যত বেশি চৰ্চা করবে, সে তত ভালো প্রোগ্রামার হবে। আর ভালো প্রোগ্রামারদের জন্য সারা পৃথিবীর দরজা খোলা।

আমাদের চারিদিকে সব মানুষরা ছুটে চলেছে উন্নতি করার আশায়। সেসব মানুষের ভিত্তে কিছু কিছু মানুষ নিজেদের উন্নত করার চেষ্টা করছে। উন্নতি করার চেষ্টে উন্নত হওয়া কিন্তু হাজারগুণ কঠিক কাজ। কিন্তু প্রোগ্রামিং শেখার একমিষ্ঠ চেষ্টার মাধ্যমে তুমি প্রকারাত্মে নিজেকে উন্নত করার কঠিন ও শ্রমসাধ্য কাজটি চালিয়ে যাচ্ছ। তাই তোমাকে আমার পক্ষ থেকে অভিনন্দন। পরিশীলনী মানুষ হও, সৎ মানুষ হও, উন্নত মানুষ হও। বিজয় সুনিশ্চিত। তোমাদের জন্য শুভকামনা।