

১। প্রোগ্রামিং দিয়ে কি করা যায়?

আমাদের ফোন/ল্যাপটপ ট্যাব দিয়ে আমরা যাবতীয় যাই করে থাকি যেমন কোনো গেমস খেলা, গুগল ম্যাপ, বা ফাইট্রেনের ওয়েবসাইট সবকিছুই প্রোগ্রামিং দিয়ে তৈরী করা হয়।



২। প্রোগ্রামার কে? সে কী করে?

একজন প্রোগ্রামার হলো এমন একজন মানুষ, যে কম্পিউটারকে কাজ শেখায়। এইসে আমরা কিছু উদাহরণ দেখে নিয়েছি যেমন গেমস, এপস বা ওয়েবসাইট এগুলো যারা তৈরী করে থাকে কম্পিউটারকে বিভিন্ন নির্দেশনা দেয়ার মাধ্যমে তারাই আসলে প্রোগ্রামার।

৩। প্রোগ্রামিং ভাষা কি?

প্রোগ্রামিং ভাষা হলো এমন একটি বিশেষ ভাষা, যার মাধ্যমে মানুষ কম্পিউটারকে নির্দেশনা দেয়। যেমন আমরা বাংলায় বা ইংরেজিতে কথা বলি, ঠিক তেমনই কম্পিউটারও কিছু নির্দিষ্ট ভাষা বুঝতে পারে, যাকে বলে প্রোগ্রামিং ভাষা।

মানুষের ভাষা: বাংলা, ইংরেজি, হিন্দি, স্প্যানিশ

উদ্দেশ্য: মানুষে মানুষে যোগাযোগ

কম্পিউটারের ভাষা: সি, সি++, পাইথন, জাভা, জাভাস্ক্রিপ্ট

উদ্দেশ্য: মানুষ এবং কম্পিউটারের মধ্যে কমিউনিকেশন

প্রোগ্রামিং ভাষার প্রকারভেদ

প্রোগ্রামিং ভাষাগুলোকে সাধারণত দুইটি প্রধান ভাগে ভাগ করা হয়

১। **Low-Level Language** (লো-লেভেল ভাষা):

এই ভাষাগুলো কম্পিউটারের খুব কাছের ভাষা। মানে, এগুলো মানুষের জন্য কঠিন, কিন্তু কম্পিউটার সহজে বুঝতে পারে।

Machine Language (মেশিন ভাষা): শুধুই 0 আর 1 দিয়ে লেখা। যেমন: 10110000
01100001

Assembly Language: এটি মেশিন ভাষার চেয়ে কিছুটা উন্নত, যেখানে বাইনারি কোডের পরিবর্তে কিছু নির্দেশাবলী (যেমন ADD, SUB) ব্যবহার করা হয়।

বৈশিষ্ট্য:

- খুব দ্রুত কাজ করে
- কম্পিউটারের হার্ডওয়্যারের সাথে সরাসরি যোগাযোগ করে
- শেখা কঠিন, ভুল ধরা কঠিন

২। **High-Level Language** (হাই-লেভেল ভাষা):

এই ভাষাগুলো মানুষের জন্য সহজবোধ্য। আমরা সাধারণ ইংরেজির মতো শব্দ ব্যবহার করে কম্পিউটারকে নির্দেশনা দিই।

♦ উদাহরণ:

- Python
- C, C++
- Java
- JavaScript

বৈশিষ্ট্য:

- শেখা সহজ
- কোড লেখা ও বুঝা সহজ
- কম্পিউটার সরাসরি বুঝতে পারে না—প্রথমে অনুবাদ করতে হয় (Compiler বা Interpreter দিয়ে)

Machine vs Assembly vs High-Level

ভাষার ধরন	মানুষ বুঝতে পারে	কম্পিউটার বুঝতে পারে	উদাহরণ	অনুবাদ প্রয়োজন
Machine Language	খুব কঠিন	সরাসরি বুঝে	10110000 01100001	প্রয়োজন নেই
Assembly Language	কিছুটা কঠিন	সহজে বুঝে	ADD, SUB	Assembler
High-Level Language	সহজ	সরাসরি বুঝে না	a = 5 + 3	Compiler/Interpreter

আমরা সি এবং সি++ কেনো শিখবো?

১. Performance (দ্রুত কাজ করার ক্ষমতা):

C এবং C++ দিয়ে লেখা প্রোগ্রামগুলো খুব দ্রুত চলে। এরা কম্পিউটারের মেমোরি, প্রসেসর, সবকিছুর সাথে সরাসরি কাজ করতে পারে। যেমন: গেম, অপারেটিং সিস্টেম, রিয়েল-টাইম সফটওয়্যার—সব জায়গায় C/C++ ব্যবহার হয়।

উদাহরণ: PUBG বা Call of Duty-এর মতো গেমের C++ ব্যবহার হয় কারণ এতে মিলিসেকেন্ডের পারফরম্যান্স গুরুত্বপূর্ণ।

২. Efficiency (কম রিসোর্সে বেশি কাজ):

C/C++ দিয়ে এমন কোড লেখা যায়, যা কম মেমোরি ব্যবহার করে, দ্রুত চলে, এবং অপটিমাইজ করা যায়।

৩. Control over Hardware (হার্ডওয়্যারের উপর নিয়ন্ত্রণ):

C/C++ দিয়ে কম্পিউটারের ভেতরের অংশ, যেমন RAM, CPU, ডিস্ক, সবকিছুর সাথে সরাসরি কাজ করা যায়। এটা অন্য অনেক হাই-লেভেল ভাষায় সম্ভব নয়।

উদাহরণ: রোবট বানাতে হলে সেন্সর, মোটর সবকিছুর নিয়ন্ত্রণ দরকার। C ব্যবহার করে সেটা করা যায়।

৪. Beginning-friendly yet Powerful:

C এবং C++ তুলনামূলকভাবে শেখা সহজ, বিশেষ করে যারা প্রোগ্রামিংয়ে নতুন। এই ভাষাগুলো একদিকে বিগিনার-ফ্রেন্ডলি, আবার অন্যদিকে বেশ পাওয়ারফুল। পয়েন্টার, মেমোরি ম্যানেজমেন্ট, ডেটা স্ট্রাকচার—এইসব গুরুত্বপূর্ণ কনসেপ্ট C/C++ দিয়ে শেখা যায় খুব ভালোভাবে। এই ভিত্তি পরবর্তীতে Java, Python, JavaScript-এর মতো ভাষা শেখা অনেক সহজ করে তুলে।

অনুবাদক প্রোগ্রাম (Translator Program)

কম্পিউটার নিজে মানুষের লেখা কোড (Source Code) বুঝতে পারে না। তাই প্রয়োজন হয় অনুবাদক প্রোগ্রাম, যা এই কোডকে কম্পিউটারের ভাষায় অনুবাদ করে দেয়।

ধরন	ব্যাখ্যা
Source Code	প্রোগ্রামার যে কোড লেখে (যেমন: <code>a = b + c;</code>)
Object Code	অনুবাদিত কোড, যা কম্পিউটার সরাসরি বুঝতে পারে (যেমন: বাইনারি বা মেশিন কোড)

Types of Translator Program

1. **Assembler** : Assembly Language কে Machine Language-এ রূপান্তর করে
2. **Compiler** : পুরো Source Code একবারে অনুবাদ করে Object Code তৈরি করে

3. **Interpreter** : Source Code লাইন বাই লাইন পড়ে এবং সঙ্গে সঙ্গে চালায়

Compiler vs Interpreter

Subject	Compiler	Interpreter
অনুবাদের ধরন	পুরো কোড একবারে অনুবাদ	লাইন বাই লাইন অনুবাদ
গতি	দ্রুত (অনুবাদ একবারেই হয়)	ধীর (প্রতিটি লাইনে সময় লাগে)
ভুল ধরার সময়	সব ভুল একসাথে দেখায়	এক লাইনে ভুল হলে সেখানেই থামে
মেমোরি ব্যবহার	তুলনামূলক বেশি	তুলনামূলক কম

প্রোগ্রাম সংগঠনের ধাপসমূহ

1. Input (ইনপুট): ব্যবহারকারী বা অন্য কোনো উৎস থেকে তথ্য নেওয়া

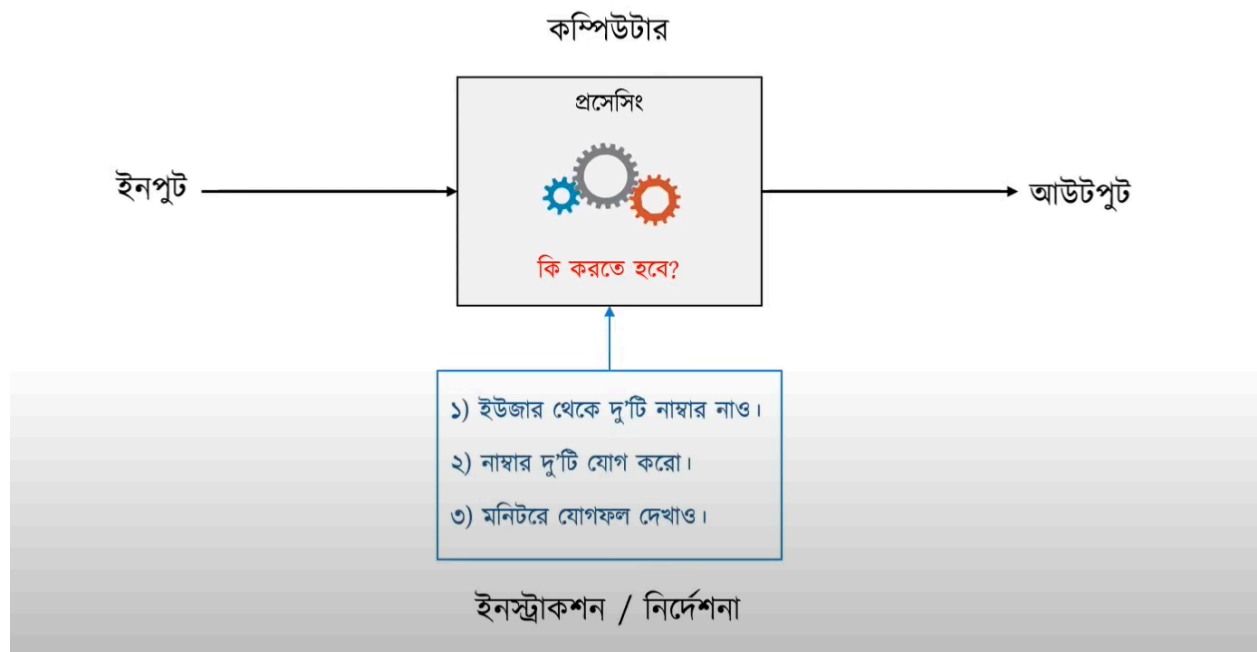
উদাহরণ: ক্যালকুলেটরে দুইটি সংখ্যা দেওয়া

2. Process (প্রসেস): ইনপুট তথ্য নিয়ে হিসাব, বিশ্লেষণ বা সিদ্ধান্ত নেওয়া

উদাহরণ: দুই সংখ্যার যোগফল বের করা

3. Output (আউটপুট): প্রসেস করা তথ্য ব্যবহারকারীকে দেখানো

উদাহরণ: যোগফল স্ক্রিনে দেখানো



Algorithm (অ্যালগরিদম) কি?

একটি অ্যালগরিদম হলো এমন নির্দেশনার তালিকা, যা অনুসরণ করলে একটি নির্দিষ্ট কাজ বা সমস্যার সমাধান পাওয়া যায়।

উদাহরণ: আমরা যদি চা বানাতে চাই, তাহলে ধাপগুলো হতে পারে—

১. পানি গরম করা।
২. চা পাতা দেয়া।
৩. দুধ-চিনি মেশানো।
৪. ছেকে কাপে ঢালা।

এই ধাপগুলোই একটি অ্যালগরিদম!




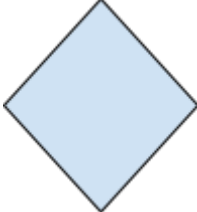
অ্যালগরিদমের বৈশিষ্ট্য:

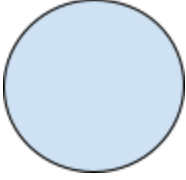

1. স্পষ্টতা (**Clarity**): প্রতিটি ধাপ পরিষ্কারভাবে লেখা থাকবে, যেন কোনো বিভ্রান্তি না হয়।
2. ধাপে ধাপে (**Step-by-step**): সমস্যার সমাধান ধাপে ধাপে এগোবে—একটি ধাপ শেষ হলে পরেরটি শুরু হবে।
3. শেষ হবে (**Finiteness**): অ্যালগরিদমের একটি শেষ থাকবে—অর্থাৎ, নির্দিষ্ট সময়ের মধ্যে সমাধান পাওয়া যাবে।

Flowchart (ফ্লোচার্ট) কি?

Flowchart হলো একটি চিত্রভিত্তিক উপস্থাপন, যেখানে অ্যালগরিদমের ধাপগুলো চিহ্ন ও তীরচিহ্নের মাধ্যমে দেখানো হয়। এটি দেখে সহজেই বোঝা যায়—কোন ধাপে কী হচ্ছে, এবং পরবর্তী ধাপে কী যাবে। অ্যালগরিদমকে যদি চিত্রে রূপ দেওয়া হয়, সেটাই Flowchart।

ফ্লোচার্ট অংকনে ব্যবহৃত প্রতীকসমূহ এবং এদের ব্যবহার:

চিহ্নের নাম	প্রতীক	ব্যবহার
Terminator		শুরু বা শেষ বোঝাতে (Start/End)
Input Output		তথ্য নেওয়া বা দেখানো বোঝাতে
Process		কোনো কাজ বা হিসাব বোঝাতে
Decision		শর্ত বা সিদ্ধান্ত বোঝাতে (Yes/No)

Connector		একাধিক প্রবাহের সংযোগ করার ক্ষেত্রে
Flow Line		এটি দুটি চিহ্নের মাঝে সংযোগ নির্দেশ করে।

Algorithm ও Flowchart এর উদাহরণ

দুটি সংখ্যার যোগফল নির্ণয়ের অ্যালগরিদম:

ধাপ ১: কাজ শুরু।

ধাপ ২: দুটি সংখ্যা a , b এর মান গ্রহণ।

ধাপ ৩: দুটি সংখ্যা a এবং b এর যোগফল s নির্ণয়। অর্থাৎ $s = a + b$ নির্ণয়।

ধাপ ৪: যোগফল s এর মান প্রদর্শন।

ধাপ ৫: কাজ শেষ।

দুটি সংখ্যার যোগফল নির্ণয়ের ফ্লোচার্ট:

