প্রোগ্রামিং-এ নতুন দিগন্ত

জাহান আলী সানাই
প্রোগ্রামার
ইটিএসি প্রকল্প, জাতীয় রাজস্ব বোর্ড

অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং (OOP)

গত কয়েক বছরে কমপিউটার বিজ্ঞানের জগতে "অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং" বিষয় আলোচনায় ব্যাপক জনপ্রিয়তা লাভ করেছে। আলোচনার ফলাফল অবশ্য মিশ্র। কেউ কেউ এটাকে গভীরভাবে গ্রহণ করেছেন আবার কেউ কেউ "প্রোগ্রামিং-এ শুধুই নতুন অলংকরণ" (Fad) এই মন্তব্যে বিষয়টিকে উড়িয়ে দিয়েছেন। অবজেক্ট অরিয়েন্টেড সিস্টেমের ধারণা একেবারে নূতন নয়। ষাট এর দশকের প্রথম ভাগে ষ্ট্যানফোর্ড-এর Doug Engelber প্রত্যক্ষভাবে নিয়ন্ত্রণ ও পরিবর্তনযোগ্য বিভিন্ন মড্যুলের সমন্বয়ভিত্তিক সিস্টেম নিয়ে কাজ করেছিলেন। সত্তরের দশকের প্রথমভাগে নরওয়েতে প্রথম Simula প্রোগ্রামিং ল্যাংগুয়েজের সঙ্গে অবজেক্ট অরিয়েন্টেড প্রোগ্রামিং-এর ধারনাকে যুক্ত করা হয় আর এই আলোচনায় আমরা দেখব যে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং কেবলমাত্র "প্রোগ্রামিং-এ নতুন অলংকরণ" নয়, এটা প্রোগ্রামিং ভাষার উল্লেখযোগ্য উন্নয়ন যাতে প্রচুর ভালো ভালো ফিচার রয়েছে এবং প্রোগ্রামিং জগতে দীর্ঘস্থায়ী অবস্থান গ্রহন করেছে। আমরা একটি আদর্শ অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ভাষার ফিচার নিয়েও আলোচনা করব এবং দেখব যে নমনীয়তা এবং Bipartiate প্রোগ্রাম গঠন, যদিও অত্যাবশ্যকীয় নয়, তথাপি ব্যবহারিক দৃষ্টিকোন থেকে প্রত্যাশিত।

আলোচনাটি কয়েকটি অংশে বিভক্ত। প্রথম অংশে দেখব OOP এর চারিত্রিক বৈশিষ্ট্য এবং তার পক্ষের যুক্তি। পরবর্তী তিনটি অংশে আমরা তিনটি পুরানো OOP ভাষা নিয়ে আলোচনা করব। যেহেতু Simula 67 প্রথম OOP ভাষা, সেজন্য এটাকে প্রথম নির্বাচন। অন্য দু'টি ভাষা হল CLU এবং Smalltalk, তাদের প্রভাব এবং বৈচিত্র্যপূর্ণ বৈশিষ্ট্যের জন্য।

অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং

OOP এর মূল কেন্দ্রবিন্দু হল অবজেক্ট। এটা ঠিক যে, অবজেক্টকে এক কথায় সংজ্ঞায়িত করা বেশ কঠিন এবং সে রকম কোন সর্বজন গ্রাহ্য সংজ্ঞাও নেই। অবজেক্টকে অবজেক্ট অরিয়েন্টেড সিস্টেমের একক হিসেবে বিবেচনা করা যেতে পারে, যা কোন বাস্তব স্বত্বা (Entity) ও তার আচরণকে প্রতিনিধিত্ব করে। ব্যাপকার্থে অবজেক্ট হচ্ছে টাইম ও স্পেসের কাঠামোতে নির্দেশিত যে কোন স্বত্বা। অন্যান্য অবজেক্টের সঙ্গে পারস্পারিক ক্রিয়া প্রতিক্রিয়ারভিত্তিতে একটি অবজেক্টকে আরেকটি অবজেক্ট থেকে আলাদা করা যায়। আর সফটওয়্যারে অবজেক্ট হচ্ছে এই সকল বস্তুগত অবজেক্ট এর প্রতিমূর্তি।

একটি অবজেক্টে থাকে ডাটা এবং ডাটার উপর কাজ করবে এমন কোন সূত্র বা সমীকরণ। অথবা যেমন প্যাসকালে একটি রেকর্ড, বিশেষ একটি পদ্ধতির মাধ্যমে গঠিত হয়। পদ্ধতিটি নির্বাচন করে কি প্রক্রিয়া ডাটার উপর কাজ করবে।

OOP এর পক্ষে বহু যুক্তি দাড় করানো যায়। আমরা বাস্তব জগত থেকে অবজেক্ট সম্পর্কে ভৌতিক ও তাত্ত্বিক ধারনা পেতে পারি। ধরা যাক একটি হাতুড়ি। হাতুড়ির বিশেষ কয়েকটি কাজ আছে যা তার উপর কেউ প্রয়োগ করতে পারে। এই বিশেষ কাজের সেট হাতুড়ির জন্য স্বাতন্ত্র। অর্থাৎ যদি কেউ হাতুড়ি স্থান পরিবর্তন করায় তবুও এই কাজের সেটটির পরিবর্তন হবে না। হাতুড়ির স্থান পরিবর্তনের সাথে সাথে তার কাজের সেটটিও থাকছে। যদি হাতুড়িকে আমরা ডাটা হিসাবে গণ্য করি তবে তার কাজের সেটটি হবে প্রক্রিয়া। অর্থাৎ হাতুড়ি ও তার কাজের সেট একত্রে একটি অবজেক্ট। তার মানে ডাটার সাথে থাকবে তার উপর ক্রিয়া করতে পারে এমন প্রক্রিয়া।

প্রোগ্রামিং এ এই যুক্তিটিকে আরো বড় করে গ্রহণ করা হয়েছে। প্রত্যেক প্রোগ্রাম ডাটার উপর কাজ করে। ডাটা ছাড়া প্রোগ্রাম অর্থহীন, প্রোগ্রামের কেন্দ্রীয় ইস্যুই হল ডাটা। সেজন্য এটা অর্থপূর্ণ মনে হয় যখন আমাদের কার্যক্রমকে ডাটার সাথে যুক্ত করি, ঠিক বাস্তব জগতে যেভাবে করা হয়ে থাকে।

জটিল একটি সিস্টেমকে Hierarchical পদ্ধতিতে প্রোগ্রাম করা হয়ে থাকে। OOP এই পদ্ধতির জন্যও সুবিধাজনক।

প্রোগ্রামিং-এ Divide and Conquer নীতিটি খুবই কার্যকরী। একটি বড় সমস্যাকে ক্ষুদ্র ক্ষুদ্র ভাগে ভাগ করে নিলে সমাধান অনেক সহজ হয়। বলা হয়েছে—

"সুক্ষ্ণভাবে চিন্তা করা সম্ভব যদি স্বল্প সংখ্যক উপাদান একবারে নেয়া হয়।" সফ্টওয়্যার ইঞ্জিনিয়ারিং দৃষ্টিকোন থেকে এ বিষয়টিকে এভাবে ব্যাখ্যা করা যায়। "ভাল বিভাজন মানে প্রত্যেকটি ভাগকে স্বাধীনভাবে প্রোগ্রাম করা যাবে, প্রয়োজনে পুনঃ সম্পাদন করা যাবে সম্পূর্ণ সিস্টেমের উপর সামান্য বা কোন প্রভাব না ফেলেই।"

একটি অবজেক্টে একটি কাজের সেট জড়িত। এই সেট স্বনির্ভর অর্থাৎ এক অবজেক্টের কাজের সেট প্রোগ্রামের অন্য অংশের উপর প্রভাব ফেলে না। ফলে অবজেক্ট প্রোগ্রামের মধ্যে স্বনির্ভর এককে পরিণত হয় এবং তাদেরকে পৃথকভাবে পরিবর্তন করা যায়। অর্থাৎ একটি অবজেক্টের ভিতরের অংশ প্রোগ্রামের বাকী অংশের সাথে সম্পর্কযুক্ত নয়, যা সম্পর্কযুক্ত তা হল অবজেক্টের কাজের সেট। এই পদ্ধতি মডিউলার পদ্ধতির সাথে সঙ্গতিপূর্ণ যেখানে প্রোগ্রামের রক্ষণাবেক্ষণ সহজ হয়। বলা হয়েছে "OOP encapsulation কে সমর্থন করে, যার ফলে সফ্টওয়্যার পুনঃ ব্যবহার, শোধন, পরীক্ষা, রক্ষণাবেক্ষণ এবং পরিবর্ধনের ক্ষমতা বৃদ্ধি পায়।"

উদাহরণের সাহায্যে বিষয়টি ব্যাখ্যা করা যায়। ধরা যাক একটি জ্যামিতিক পদ্ধতি যা কিছু সংজ্ঞায়িত জ্যামিতিক চিত্রকে প্রদর্শন করে। প্রত্যেকটি চিত্র প্রদর্শনের জন্য একটি করে পদ্ধতি থাকবে। প্রচলিত প্রোগ্রামিং পদ্ধতিতে আমাদের হয়ত একটি প্রদর্শনী রুটিন থাকবে যেখানে একটি বৃহৎ IF-THEN-ELSE পরীক্ষার মাধ্যমে জ্যামিতিক চিত্রকে সনাক্ত করা হয় এবং নির্দিষ্ট প্রদর্শনী রুটিনের সাহায্যে জ্যামিতিক চিত্রকে প্রদর্শন করা হয়। এই পদ্ধতির সবচেয়ে বড় অসুবিধা হল একটি জ্যামিতিক চিত্রের যোজন, বিয়োজন ও সম্পাদনের সাথে সাথে সম্পূর্ণ প্রোগ্রামকে পরিবর্তন করতে হয়। কিন্তু OOP—এ বিষয়টি বেশ সহজ। আমরা প্রত্যেকটি জ্যামিতিক চিত্রকে এক একটি অবজেক্ট হিসাবে নিতে পারি যাদের প্রত্যেকের নিজস্ব প্রদর্শনী রুটিন থাকবে। ফলে একটি অবজেক্ট বা জ্যামিতিক চিত্রকে প্রদর্শনের জন্য কেবলমাত্র তার প্রদর্শনী রুটিনকে ডাকতে হবে। ফলে প্রদর্শনী রুটিনের পরিবর্তন কেবলমাত্র নির্দিষ্ট জ্যামিতিক চিত্রের জন্য সীমিত থাকে।

নতুন কোন জ্যামিতিক চিত্র যদি যুক্ত করতে হয়, তবে সম্পূর্ণ প্রোগ্রামকে পরিবর্তন না করে কেবলমাত্র একটি অবজেক্ট যোগ করে নিলেই হবে। ডিবাগিং অনেক সহজ। কারণ কোন প্রদর্শনী রুটিনে সমস্যা হচ্ছে তা আমরা তাৎক্ষনিকভাবে জানি। আমাদের প্রদর্শনী রুটিনগুলোর কেবলমাত্র একটি নাম থাকতে পারে। এটাকে বলা হয় Name Overloading। ফলে নামকরণ কোন সমস্যার সৃষ্টি করে না।

কোড পুনঃ ব্যবহার সফ্টওয়্যার ইঞ্জিনিয়ারিং এর একটি বড় ইস্যু। OOP -এ একটি অবজেক্ট সহজেই আবার ব্যবহার করা যায়। কারণ একটি অবজেক্টের কাজের সেট প্রোগ্রামের অন্য অংশের উপর নির্ভর করে না। Inheritance পদ্ধতির মাধ্যমেও OOP—এ কোড পুনঃ ব্যবহার করা যায়।

সফ্টওয়্যার ইঞ্জিনিয়ারিং-এর আর একটি উপকারী ফিচার হল "তথ্য আড়াল করা" বা Information hiding. কয়েকটি OOP ভাষায় অবজেক্টের অভ্যন্তরীন বিষয়সমূহকে সংরক্ষিত করা হয় যাতে অবজেক্টের ডাটাকে কেবলমাত্র তার কাজের সেট দিয়ে ব্যবহার করা যায়। এর ফলে অবজেক্টের ডাটা সবসময় সঠিক থাকে, কারণ অন্য কোন কাজের সেট সেই ডাটার উপর কাজ করতে পারে না। ফলে প্রত্যেকটি অবজেক্ট স্বাধীনভাবে কাজ করতে পারে অর্থাৎ মডিউলারিটির দিকে ধাবিত হয়। একটি অবজেক্টে ভেরিয়েবল এর নামও পরিবর্তন করা যায় অন্যান্য বহিঃ রুটিনকে প্রভাবিত না করে।

একটি বিষয় পরিস্কার যে সোর্সকোড লেখা আরম্ভ করার আগে সুষ্ঠ পরিকল্পনা প্রয়োজন অন্ততঃ OOP এর ক্ষেত্রে। হয়ত এর পক্ষের এবং বিপক্ষের উভয় যুক্তিই দেখানো যেতে পারে। সুষ্ঠ পরিকল্পনার সুবিধা হল সমস্যা সম্পর্কে স্বচ্ছ ধারণা থাকে। ফলে ডিবাগিং কম করতে হয়। অপরপক্ষে এটাও বলা যেতে পারে যে কখনো কখনো পরিকল্পনা ছাড়াই প্রোগ্রাম লেখা আরম্ভ করতে হয় এবং কাজটা শেষ হয় একটি বড় প্রোগ্রাম লেখা শেষ করার মাধ্যমে। অবশ্য এই পদ্ধতিও OOP এর ক্ষেত্রে ব্যবহার করা যায়। যদিও কিছুটা কঠিন এবং অস্বাভাবিক।

অবজেক্ট সবসময় মেমোরীতে অবস্থান করে। ফলে একসময় হয়ত আমরা run out of memory পর্যায়ে চলে আসতে পারি। কিছু OOP ভাষা কখনো কখনো garbage সংগ্রহ করে। বর্তমানে Garbage সংগ্রহ দূর করার জন্য প্রচুর কাজ হচ্ছে।

Simula 67

Simula 67 এ সর্বপ্রথম OOP এর ধারনাটি যুক্ত করা হয়। এটা Algol 60 এর উপর ভিত্তি করে Simula I নামে পরিকল্পনা করা হয় সিস্টেম বর্ণনা ও সিস্টেম সিমুলেশন করার জন্য। অবজেক্ট সম্পর্কে ধারনা Simula I থেকে বর্ধিত হতে থাকে।

সিমুলা ৬৭ সর্বপ্রথম Class এর সূচনা করা হয়। অবজেক্ট হল Class এর একটি ক্ষণ (instance).

Class অনেকটা Procedure এর মত। প্রত্যেকটি Class এ থাকে আরম্ভকারী কোড এবং আরম্ভ করার পদ্ধতি। সিমুলা ৬৭ ডিজাইন করার সময় Class inheritance করার বিষয়টি গুরুত্ব দেয়া হয়। এতে পূর্বেই কিছু Class নির্দিষ্ট করে রাখা হয় যাতে সেগুলো hierarchical গঠনে ব্যবহার করা যায়। পূর্বেই নির্দিষ্ট করে রাখা class প্রোগ্রামাররা প্রয়োজনমত ব্যবহার করতে পারেন। detach এবং resume নামে দু'টি অপারেটর সিমুলা ৬৭ তে যুক্ত করা হয় যার মাধ্যমে নিয়ন্ত্রণ প্রবাহ অবজেক্ট ত্যাগ করে বা Caller এর কাছে ফিরে আসে। এভাবে দেখা যায় যে সিমুলা ৬৭ তে অনেক গুরুত্বপূর্ণ ফিচার যোগ করা হয়েছে, তার কিছু পরবর্তীতে আলোচনায় আসবে।

CLU

CLU তে তথ্য আড়াল করা এবং data abstraction এর বিষয়টি সবচেয়ে গুরুত্ব পায়। এতে একটি সমস্যাকে অনেকগুলো ক্ষুদ্র অংশে ভাগ করা হয় abstraction সনাক্ত করণের মাধ্যমে। অর্থাৎ প্রত্যেকটি ক্ষুদ্র অংশকে abstraction করা সম্ভব হয়।

Simula 67'র সাথে CLU'র পার্থক্য হল CLU তে নিরাপত্তার বিষয়টি প্রাধান্য পেয়েছে।

এখানে Class এর পরিবর্তে আছে Cluster। Cluster এর ভিতরের ভেরিয়েবল কেবলমাত্র ভিতরের ফাংশন দিয়ে ব্যবহার যোগ্য। বাইরের কোন রেফারেন্স এখানে ব্যবহার করা যায় না। সেজন্য বাইরের কেউ Cluster এর ভিতরের কিছু দেখতে পায় না। বাইরের সবাই যা জানে তা হল Cluster কে কিভাবে ব্যবহার করতে হবে তা। অর্থাৎ প্রত্যেকটি ডাটা অবজেক্ট এর চরিত্র তাদের কার্যক্রম দিয়ে বিশ্লেষিত হয়। CLU'র Cluster কিন্তু সিমুলা ৬৭'র class এর মত নয়। প্রত্যেকটি cluster এ থাকে create নামে একটি procedure যা cluster এর ভিতরের অবজেক্টকে initialize করে। একটি cluster এর ধরণ (type) কে এখানে parameter হিসাবে ব্যবহার করা যায়। ফলে একটি cluster বহুবিধ মাত্রা লাভ করে।

Smalltalk

Smalltalk কেবলমাত্র একটি প্রোগ্রামিং ভাষা নয়। এতে রয়েছে একটি এডিটর, উইনডো ম্যানেজার, একটি ডিবাগার, অবজেক্ট প্রদর্শনী রুটিন এবং একটি ইন্টারপ্রেটার। এটা সামগ্রিকভাবে প্রোগ্রামিং জগতে বিশাল প্রভাব ফেলে, কিন্তু আমরা কেবল এর ভাষাগত দিকটাই দেখব। স্মলটক OOP কে সর্বোচ্চমাত্রায় নিয়ে যায়। এখানে সবকিছুই অবজেক্ট। যেমন এখানে একটি Executing process অবজেক্ট হতে পারে, আবার সম্পূর্ণ environment টাই অবজেক্ট হতে পারে। প্রথম ভাবনায় মনে হয় বিষয়টি সহজ বলে সুন্দর কিন্তু দুর্ভাগ্যবশত স্মলটক এ প্রোগ্রামিং কম সহজ বা সামঞ্জস্যহীন। ব্যাকরণগত সামঞ্জস্যহীনতা থাকার কারণে স্মলটক প্রোগ্রামিং এর জন্য কম আকর্ষণীয় এবং আরো কিছু তত্ত্বগত বিষয় আছে যাতে স্মলটক এ কাজ করা কঠিন হয়ে দাঁড়ায়। স্মলটক এ সবকিছুকে অবজেক্ট হিসাবে গ্রহণ করারা ফলে OOP এর কিছু সীমাবদ্ধতা বেরিয়ে এসেছে যা আমরা এখন আলোচনা করব।

আলোচনা

এখন প্রশ্ন হল আমরা কোন উপস্থাপনাটি গ্রহণ করবো। স্মলটক এর উপস্থাপনা যেখানে প্রতিটি জিনিষই অবজেক্ট, নাকি Simula'র উপস্থাপনা যেখানে অবজেক্ট প্রচলিত প্রোগ্রামিং ভাষার নতুন একটি সংযুক্তি। Simula তে সক্রিয় এবং নিষ্ক্রিয় উভয় ধরনের অবজেক্ট থাকতে পারে। স্মলটক এ সবকিছু অবজেক্টে পরিণত হয়। ফলে প্রচলিত structured প্রোগ্রামিং-এর প্রয়োজনীয়তা এখানে দূর হয়। কিন্তু প্রোগ্রামিং এখানে জটিল। আমরা hierarchical ধরনের প্রোগ্রামিং সুবিধা হারাই, তার পরিবর্তে পাই অবজেক্ট যারা পরস্পরের সাথে যোগাযোগ করে বার্তা আদান প্রদানের মাধ্যমে। আমাদের যে সামগ্রিক দৃষ্টিকোন যেখানে আমাদের কাজকে আমরা Structured manner এ প্রোগ্রাম ও নিয়ন্ত্রণ করি তা এখানে অকার্যকর। স্মলটক কোন নিষ্ক্রিয় অবজেক্ট থাকে না যার উপর আমরা কাজ করতে পারি। নিষ্ক্রিয় অবজেক্ট না থাকার ফলে অসুবিধা হল ডাটা নিয়ে আলাদাভাবে কাজ করে যায় না। ধরা যাক সংখ্যা নিয়ে কাজ করব। এ ক্ষেত্রে সংখ্যার সাথে সাথে তার উপর কাজ করতে পারে প্রক্রিয়াও চলে আসে। ফলে যখন আমরা নতুন কোন সংখ্যা তৈরী করি তার সাথে সাথে প্রক্রিয়াও চলে আসে। এ বিষয়টি কিছুটা বাহুল্য মনে হয়। এর মূলে কারণ হল স্মলটক এ সবকিছুই অবজেক্ট। ফলে অবজেক্ট এবং Class এর মধ্যে পার্থক্য করা যায় না, OOP'র ধারণা কিসের উপর ভিত্তি করে তা বোঝা যায় না। দেখা যাচ্ছে স্মলটক অবজেক্টকে অতিমাত্রায় গ্রহণ করা হয়েছে যা simula তে করা হয়নি।

আমরা এতক্ষণ বলতে চেয়েছি OOP একটি অগ্রসর প্রোগ্রামিং। আমরা আরো দেখেছি যে স্মলটক সম্পূর্ণভাবে অবজেক্ট এর প্রতি নিবেদিত। তা হলে আমাদের কি প্রয়োজন ? এর উত্তর জানার আগে জানা প্রয়োজন অবজেক্ট থেকে কি কি সুবিধা আমরা পেতে চাই। আমরা অবজেক্টকে ব্যবহার করতে চাই মডিউলারিটি, রক্ষনাবেক্ষন এবং পরিবর্ধন সুবিধা পাবার জন্য। একই সাথে Structured প্রোগ্রামিং এবং hierarchy সুবিধা পেতে চাই যা দিয়ে অবজেক্টকে নিয়ন্ত্রণ করা যাবে। আবার এমনও হতে পারে যে একটি বিশেষ প্রোগ্রামের ক্ষেত্রে আমাদের অবজেক্ট ওরিয়েন্টেড উপস্থাপনা গ্রহনের প্রয়োজন নেই। এটা দেখা যায় যে ছোট প্রোগ্রাম খুব দ্রুত লেখা যায় এবং ডাটা প্যারামিটার ইচ্ছেমত পরিবর্তন করা যায়। প্রোগ্রামটি হয়ত মাত্র একবার ব্যবহৃত হবে। এক্ষেত্রে কি বৃহৎ পরিকল্পনার মাধ্যমে অবজেক্ট ওরিয়েন্টেড উপস্থাপনা গ্রহণ যুক্তিসঙ্গত হবে ? ব্যবহারিক অভিজ্ঞতা থেকে বলা যায় এক্ষেত্রে এত ঝামেলার প্রয়োজন নেই। প্রোগ্রাম রক্ষনাবেক্ষন বা পরিবর্ধন প্রয়োজন হবে না যদি প্রোগ্রামকে মাত্র কয়েকবার ব্যবহার করা হয়। এক্ষেত্রে আমরা প্রচলিত প্রোগ্রামিং পদ্ধতি ব্যবহার করতে চাইব।

এই নমনীয়তা Simula তে রয়েছে। অর্থাৎ Bipartiate প্রোগ্রাম গঠন সম্ভব। মজার ব্যাপার হল নতুন OOP ভাষা যেমন C++ এ এই নমনীয়তা রয়েছে। C++ এ প্রচলিত C প্রোগ্রামিং-এর পাশাপাশি OOP করা যায়। এই নমনীয় বেশীর ভাগ প্রোগ্রামার পছন্দ করবেন।

নিরাপত্তা আর একটি মজার ব্যাপার। CLU তে অবজেক্ট এর নিরাপত্তা সিরিয়াসলি নেয়া হয়। বাইরের কোন রুটিন দিয়ে ভিতরের ভেরিয়েবল কে ব্যবহার করা যায় না। কঠিন নিরাপত্তা থাকার কারণে CLU তে প্রকৃত inheritance পাওয়া যায় না। এর ফলে OOP এর প্রয়োজনীয় কিছু বিষয় বাদ পড়ে যায়। আবার Simula তে কোন নিরাপত্তা নেই। যে কেউ অবজেক্ট-এর ভিতরে প্রবেশ করতে পারে। তাহলে প্রকৃত সমাধান কি ? সম্ভাব্য সমাধান নিয়ে আলোচনা করার আগে আমরা দেখতে চাই নিরাপত্তার আদৌ কোন প্রয়োজন আছে কিনা। আমরা আগে বলেছি নিরাপত্তা থাকায় ডাটা সবসময় এক থাকে এবং অবজেক্টের সার্বভৌমত্ব নিশ্চিত হয়। মডিউলারিটির দৃষ্টিকোন থেকে এটা সুন্দর, কারণ কোন সমস্যা ছাড়াই অবজেক্টকে যে কোন স্থানে স্থাপিত করা যায়। C++ এ নিরাপত্তা ইস্যুর একটি সুন্দর সমাধান দেয়া হয়েছে। এখানে তিন ধরণের ভেরিয়েবল আছে। যথা সংরক্ষিত, ব্যক্তিগত এবং পাবলিক। সংরক্ষিত ভেরিয়েবল কেবল Class বা অবজেক্ট ব্যবহার করতে পারে। ব্যক্তিগত ভেরিয়েবল অবজেক্টের বাইরের কেউ ব্যবহার করতে পারে না। পাবলিক ভেরিয়েবল সবার জন্য উন্মুক্ত। ফলে C++ এ প্রোগ্রামকে সুবিধাজনকভাবে নিরাপদ করা যায়।

আমরা দেখেছি যে নিরাপত্তা কিভাবে CLU তে প্রভাব বিস্তার করেছে। অত্যধিক নিরাপত্তার কারণে এখানে inheritance প্রায় অসম্ভব। কিন্তু inheritance OOP'র একটি গুরুত্বপূর্ণ বিষয়। দুঃখজনকভাবে আলোচনা তিনটি ভাষার কোনটিই একাধিক inheritance কে সমর্থন করেনি। কিন্তু একটি hierarchical সিস্টেমে একাধিক inheritance এর প্রয়োজন হতে পারে। ধরা যাক একটি হার্ডওয়্যার বোর্ড যার সাথে আমরা সিরিয়াল পোর্ট ও প্যারালাল পোর্ট যুক্ত করতে চাই। আমাদের একটি অবজেক্ট থাকবে সিরিয়াল পোর্টের জন্য, একটি থাকবে প্যারালাল পোর্টের জন্য। এ ক্ষেত্রে আমরা চাইব উভয় অবজেক্টের ধর্ম হার্ডওয়্যার বোর্ডে inherit করতে। এজন্য আমাদের প্রয়োজন একাধিক inheritance.

Simula তে একাধিক inheritance এর অসুবিধা হল name conblict করতে পারে। আমরা যদি দু'টি Class inherit করি যাদের পদ্ধতির নাম একই তবে কোন অবজেক্টটি কাজ করবে তা পরিস্কার নয়। এটা যে কোন OOP ভাষার সমস্যা হতে পারে। সিমুলা ৬৭ , CLU এবং স্মলটক এ সমস্যার সমাধান দেয়া হয়েছে একাধিক inheritance না রেখে। তবুও আমাদের একাধিক inheritance প্রয়োজন। আমরা যদি C++ এর দিকে তাকাই তবে দেখব যে এর সমাধান পাওয়া যাচ্ছে। C++ এ দুই ধরনের পছন্দ আছে। আমরা non-virtual inheritance করতে পারি যেখানে যে Class গুলো inherit করা হয় তাদের নামের জায়গা পৃথক করে। আবার virtual inheritance করা যায় যেখানে name conflict করে। ফলে C++ এ প্রয়োজন অনুযায়ী inherit করা যায় এবং প্রোগ্রামিং এ নমনীয়তা বৃদ্ধি পায়।

এ পর্যন্ত যা বোঝা গেল তা হল Simula সঠিক দিকে এগুলো আরম্ভ করেছিল। CLU এবং স্মলটক তাদের নিজস্ব পথে চূড়ান্ত পর্যায়ে চলে গেছে যার ফলে কিছু অনাকাংখিত সমস্যা দেখা দিয়েছে। এ থেকে যৌক্তিক শিক্ষা যা পাওয়া যায় তা হল একধাপ পিছনে এসে পরীক্ষা করা যে আমরা এগিয়ে গিয়ে লাভবান হয়েছি কিনা। C++ সঠিক দিকে এগিয়েছে মনে হয় এবং আশা করা যায় ভবিষ্যতে আরো উন্নত হবে।

OOP থাকবে। এটা কোড পুনঃ ব্যবহার, রক্ষনাবেক্ষন মডিউলারিটি এবং পরিবর্ধনের প্রচুর সুবিধা দিয়েছে যা একে আরো জনপ্রিয় এবং গৃহীত করে তুলবে। এর অর্থ এই নয় যে প্রচলিত প্রোগ্রামিং শেষ হয়ে যাবে।

সেজন্য Bipartiate প্রোগ্রাম গঠন সুবিধাজনক যেখানে আমরা সক্রিয় ও নিষ্ক্রিয় ডাটা অবজেক্ট পেতে পারি। নতুন OOP ভাষাগুলো সবধরণের নমনীয় সুবিধা দিচ্ছে যেমন C++। পরিশেষে এটা বলা যায় যে প্রোগ্রামিং ভাষার উন্নয়ন তার নিজস্ব গতিতে এগিয়ে যাবে, কখনো বর্তমানকে সাথে নিয়ে, কখনো বর্তমানকে পিছনে ফেলে। ❂