সহজ ভাষায় প্রোগ্রামিং সি/সি++

আহমদ ওয়াহিদ মাসুদ

সি ল্যাঙ্গুয়েজকে প্রোগ্রামিংয়ের জগতে মাদার অব ল্যাঙ্গুয়েজ বলা হয়। আধুনিক সব ল্যাঙ্গুয়েজের বেসিক স্ট্রাকচার সি থেকে নেয়া। কারণ, সি-তে একইসাথে যেমন একেবারে লো লেভেলে অর্থাৎ মেমরির অ্যাড্রেস লেভেলে কাজ করা যায়, তেমনি অনেক হাই লেভেল কোডিং করার সুবিধাও আছে। ইউজারের কোড করার সুবিধার্থে ও ইউজারের কষ্ট কমানোর জন্য সি-তে অনেক হাই লেভেল ফিচার আছে। সি ল্যাঙ্গুয়েজের একটি অন্যতম ফিচার হলো কাস্টম ডাটা টাইপ ব্যবহার করা। এ লেখায় কাস্টম ডাটা টাইপ কী, সি-তে কী কী কাস্টম ডাটা টাইপ আছে ও তা কীভাবে ব্যবহার করতে হয় ইত্যাদি উপস্থাপন করা হয়েছে।

কাস্টম ডাটা টাইপের প্রাথমিক ধারণা

কাস্টম ডাটা টাইপ হলো ইউজারের নিজের তৈরি করা ডাটা টাইপ। ইউজার নিজের সুবিধার জন্য প্রয়োজনমতো ডাটা টাইপ তৈরি করে নিতে পারেন। আমরা জানি, সি-তে কিছু বিল্টইন ডাটা টাইপ আছে, যেমন : int, float, char ইত্যাদি। তবে এই বিল্টইন ডাটা টাইপ দিয়েও অনেক সময় ঠিকমতো কাজ করা যায় না। যেমন : একটি প্রোগ্রাম লিখতে হবে, যার কাজ হলো একটি ক্লাসের ১০০ জন ছাত্রের রোল নাম্বার ও গ্রেড ইনপুট নেয়া। এখন রোল নাম্বার ইনপুট নেয়ার জন্য ইন্টিজার ভেরিয়েবল দরকার। আবার গ্রেড ইনপুট নেয়ার জন্য ক্যারেক্টার টাইপ ভেরিয়েবল দরকার (ধরা যাক, গ্রেড শুধু একটি অক্ষর দিয়ে হবে)। তাহলে স্বভাবতই ইউজার ১০০ + ১০০ = ২০০টি ভেরিয়েবল ডিক্লেয়ার করবে না। অবশ্যই এ ক্ষেত্রে অ্যারে ব্যবহার করতে হবে। কিন্তু দুই ধরনের ভেরিয়েবলের জন্য দুটি অ্যারে ডিক্লেয়ার করা দরকার। এ ধরনের ক্ষেত্রে কাস্টম ভেরিয়েবল ব্যবহার করা যায়। ইউজার যদি এখানে কাস্টম ভেরিয়েবল ব্যবহার করেন, তাহলে মাত্র একটি অ্যারে দিয়েই সব ইনপুট নেয়া যাবে। এখানে মনে হচ্ছে, অ্যারে হলেই কাজ হয়ে যেত। কিন্তু এ ধরনের প্রোগ্রাম যদি অনেক বড় কাজের জন্য হয়, তাহলে সাধারণ অ্যারে দিয়েও হয় না। যেমন : এই একই কাজ যদি আরও বড় অর্থাৎ মোট ৫০টি স্কুল থেকে ১০০ জন্য ছাত্রের নাম, রোল, গ্রেড, পিতার নাম, ফোন নাম্বার, বয়স ও উচ্চতা ইনপুট নেয়ার দরকার হয় তাহলে সাধারণ অ্যারে দিয়ে কাজটি করা অনেক কঠিন হয়ে যাবে। এ ধরনের সমস্যার জন্য কাস্টম ডাটা টাইপ ব্যবহার করা হয়।

কাস্টম ডাটা টাইপ

সি-তে মোট পাঁচভাবে নতুন ধরনের ডাটা টাইপ ডিক্লেয়ার করা যায়। যেমন : ০১. structure : এ ক্ষেত্রে বিল্টইন ডাটা টাইপগুলো ব্যবহার করে একটি হাইব্রিড ডাটা টাইপ ব্যবহার করা হয়। ০২. bit-field : এটি structure পদ্ধতিরই একটি ভিন্ন রূপ, যার মাধ্যমে মেমরির বিট লেভেলে কাজ করা হয়। ০৩. union : এই পদ্ধতিতে এমন ডাটা টাইপ তৈরি করা যায়, যার একাধিক ফিল্ডের জন্য মেমরির একই অংশ ব্যবহার হয়। ০৪. enumeration : এই পদ্ধতিতে তৈরি ডাটা টাইপের ভেরিয়েবলের মান একটি সিম্বল লিস্ট থেকে নির্ধারিত হয়। ০৫. typedef : এই পদ্ধতিতে বিল্টইন কিংবা কাস্টম ডাটা টাইপের নতুন নাম নির্ধারণ করা যায়।

এখানে প্রতিটি পদ্ধতিরই কিছু না কিছু সুবিধা-অসুবিধা আছে। আবার সব কাজ একই ধরনের নয়। যেমন : শেষের ডাটা টাইপ নতুন ডাটাকে নিয়ে কাস্টম টাইপ বিল্ড করে না, বরং অন্যকে নতুন নাম দিয়ে কাস্টম টাইপ হিসেবে তৈরি করে। আবার এখানে বিট-ফিল্ডই একমাত্র পদ্ধতি, যার মাধ্যমে লো লেভেলে কাজ করা যায়। প্রথমে structure নিয়ে আলোচনা করা হবে।

স্ট্রাকচার : প্রোগ্রাম চলার সময় সব ডাটাই মেমরিতে সেভ করা থাকে। অন্যান্য হাই লেভেল ল্যাঙ্গুয়েজের মতো সি-তেও ডাটা নিয়ে কাজ করার জন্য ভেরিয়েবল ব্যবহার করা হয়। আমরা জানি একই টাইপের অনেকগুলো ভেরিয়েবল নিয়ে যদি কাজ করার দরকার হয় তাহলে অ্যারে ব্যবহার করলে প্রোগ্রামের জটিলতা অনেক কমে যায়। কিন্তু এমন অনেক সময় আসতে পারে যখন একাধিক টাইপের একাধিক ভেরিয়েবলের দরকার হচ্ছে, সে ক্ষেত্রে কাস্টম ডাটা টাইপ খুবই সাহায্য করে। এবার আগের দেয়া উদাহরণকে এখানে আরেকটু বিস্তারিতভাবে দেয়া হলো। ধরা যাক, উক্ত ১০০টি স্কুলের ছাত্রের যেসব তথ্য ইনপুট নেয়া হবে তা হলো :

student name
department name
id
gpa
total credits
total course

এখন কাস্টম ডাটা টাইপ ব্যবহার না করলে প্রতিটি ভেরিয়েবলের জন্য আলাদা অ্যারে ডিক্লেয়ার করতে হবে। যেমন :

```
char name[30], dept_name[10];
long int id;
double gpa;
int credit, course;
```

লক্ষ করলে দেখা যাবে এখানে যেভাবে ভেরিয়েবলগুলো ডিক্লেয়ার করা হয়েছে, তাতে একটি স্কুলের ছাত্রদের সব তথ্য নেয়া সম্ভব। কিন্তু ১০০টি স্কুলের ছাত্রদের তথ্য নিতে বলায় এভাবে তা ইনপুট নেয়া সম্ভব হবে না।

আবার ধরা যাক, এই প্রোগ্রামে অনেকগুলো ফাংশন আছে, যারা এই ডাটাগুলো নিয়ে বিভিন্ন ধরনের কাজ করেন। ফলে প্রতিবার ফাংশনগুলোতে এই ডাটাগুলো নির্দিষ্ট ক্রমানুসারে আর্গুমেন্ট হিসেবে পাঠানো হয়। কিন্তু ফাংশনে বেশিসংখ্যক আর্গুমেন্ট পাঠাতে গেলে প্রোগ্রামের জটিলতা অনেকাংশে বেড়ে যাবে। আর ভুল হওয়ার সম্ভাবনা তো থাকেই। এখন এখানে স্ট্রাকচার ব্যবহার করা মানে হলো সব ভেরিয়েবলগুলোকে গ্রুপ করে দেয়া। এরপর ইউজার এই ভেরিয়েবলগুলোকে সেই গ্রুপের মেম্বার হিসেবে ব্যবহার করতে পারবেন। আবার ইউজার ইচ্ছে করলে সেই গ্রুপের অ্যারেও তৈরি করতে পারবেন। এবার উপরের উদাহরণটিই স্ট্রাকচারের মাধ্যমে দেয়া হলো :

```
struct student
{
        char name[30];
        char dept_name[10];
        long int id;
        double gpa;
        int credit;
        int course;
};
```

এখানে স্টুডেন্ট নামে নতুন একটি ডাটা টাইপ তৈরি করা হলো। যার মোট ৬টি মেম্বার ভেরিয়েবল আছে। এখন ইউজার যদি স্টুডেন্ট টাইপের অ্যারে ডিক্লেয়ার করেন তাহলেই ১০০টি স্কুলের জন্য প্রোগ্রামটি লেখা যাবে।

স্ট্রাকচার ডিক্লেয়ার : কোনো ভেরিয়েবল কিংবা অ্যারেকে যেভাবে ডিক্লেয়ার করা হয়, প্রোগ্রামে একটি স্ট্রাকচারকেও অবশ্যই সেভাবে ডিক্লেয়ার করতে হবে। এটি ডিক্লেয়ারের নিয়ম হলো :

```
struct tag
{
        member1;
        member2;
        ........
        ........
        memberN;
};
```

লক্ষ রাখতে হবে ডিক্লেয়ারের শেষে একটি সেমিকোলন দিতে হয়।

উপরের ডিক্লেয়ারের নিয়ম থেকে দেখা যাচ্ছে এতে মূলত তিনটি অংশ আছে। যেমন : struct, tag, member ।

struct হলো একটি কিওয়ার্ড। এর মাধ্যমে প্রোগ্রামকে জানিয়ে দেয়া হয় যে একটি স্ট্রাকচার ডিক্লেয়ার করা হচ্ছে। tag হচ্ছে স্ট্রাকচারটির নাম। খেয়াল করতে হবে এটি কিন্তু কাস্টম ভেরিয়েবলের নাম নয়, বরং কাস্টম ডাটা টাইপের নাম। এখানে ভেরিয়েবলের নাম লেখার নিয়ম অনুযায়ী যেকোনো নাম লেখা যাবে, যা পরে একটি ডাটা টাইপ হিসেবে ব্যবহার হবে। অর্থাৎ tag-এর জায়গায় যদি student লেখা হয়, ▶

তাহলে ডিক্লেয়ারের পর বলা যাবে সেটি student টাইপ ভেরিয়েবল, ঠিক যেভাবে বলা হয় int টাইপ ভেরিয়েবল।

স্ট্রাকচার মেম্বার : { } বন্ধনীর ভেতরে ডিক্লেয়ার করা বিভিন্ন ভেরিয়েবল, অ্যারে, পয়েন্টার অথবা অন্য কোনো স্ট্রাকচার ভেরিয়েবলকে ওই স্ট্রাকচারের মেম্বার বলা হয়। যেমন :

- একই স্ট্রাকচারের একাধিক মেম্বার থাকলে তাদের সবার নাম ভিন্ন হতে হবে।
- প্রত্যেক মেম্বারের শেষে সেমিকোলন দিতে হবে।
- স্ট্রাকচার ডিক্লারেশনের সময় তথা নতুন ডাটা টাইপ ডিফাইনের সময় কোনো মেম্বারের মান নির্ধারণ করা যায় না। এখানে মান ডিক্লেয়ার করতে গেলে এরর দেখাবে। কারণ এটি কোনো ভেরিয়েবল ডিক্লারেশন নয়, শুধু ভেরিয়েবল টাইপ ডিক্লারেশন।

স্ট্রাকচার ডিক্লেয়ার করার সময় সেমিকোলনের ব্যবহার সতর্কতার সাথে খেয়াল করতে হয়। কোনো স্ট্রাকচারের মেম্বার ডিক্লেয়ার করার পর সেমিকোলন তো দিতে হবেই, সম্পূর্ণ স্ট্রাকচারটি ডিক্লেয়ার করার পরও আরেকটি সেমিকোলন দিতে হবে। অর্থাৎ সম্পূর্ণ স্ট্রাকচার ডিক্লারেশনই একটি স্টেটমেন্ট হিসেবে বিবেচিত হয়।

স্ট্রাকচার ভেরিয়েবল : যেকোনো প্রোগ্রামিং ল্যাঙ্গুয়েজে যেকোনো ধরনের ভেরিয়েবল ব্যবহার করার আগে তাকে ডিক্লেয়ার করে নিতে হয়। কাস্টম ডাটা অর্থাৎ স্ট্রাকচারের ক্ষেত্রেও একই নিয়ম প্রযোজ্য। প্রথমে স্ট্রাকচারটি ডিক্লেয়ার করতে হয় এরপর সাধারণ ভেরিয়েবলের নিয়মানুসারে ওই কাস্টম ডাটা টাইপের ভেরিয়েবলও ডিক্লেয়ার করতে হয়। যেমন : আগে স্টুডেন্ট নামে একটি স্ট্রাকচার ডিক্লেয়ার করা হয়েছে। এবার ওই স্টুডেন্ট টাইপের ভেরিয়েবল নিচের মতো ডিক্লেয়ার করতে হবে :

```
struct student var;
struct student var[50];
struct student* var;
```

এখানে স্ট্রাকচার ভেরিয়েবল, স্ট্রাকচার অ্যারে ও স্ট্রাকচার পয়েন্টার ডিক্লেয়ার করা দেখানো হলো। কোনো স্ট্রাকচারের ভেরিয়েবল ডিক্লেয়ার করতে হলে প্রথমে struct কিওয়ার্ড লিখে ওই স্ট্রাকচার টাইপের নাম লিখে ভেরিয়েবলের নাম লিখতে হবে। আর ভেরিয়েবলটি যদি অ্যারে হয়, তাহলে সাধারণ অ্যারের মতো ওই ভেরিয়েবলের নামের শেষে [] লিখে মাঝে মোট এলিমেন্ট সংখ্যা লিখতে হয়। আর পয়েন্টার ডিক্লেয়ার করতে হলে সাধারণ ভেরিয়েবলের মতোই লিখতে হয়, শুধু টাইপের ডান পাশে একটি * সাইন ব্যবহার করতে হয়। সুতরাং স্ট্রাকচার ভেরিয়েবল ডিক্লারেশনের সিনটেক্সকে নিচের মতো লেখা যায় :

```
struct type_name variable_name;
```

আরেকভাবে স্ট্রাকচারের ভেরিয়েবল ডিক্লেয়ার করা যায়। সেটি হলো স্ট্রাকচার ডিক্লেয়ার করার সময়ই তার ভেরিয়েবল ডিক্লেয়ার করা। নিচে একটি ডিক্লারেশনের উদাহরণ দেয়া হলো।

```
struct student
{
        char name[24];
        int id;
        double cgpa;
}var;
```

এখানে স্ট্রাকচার ডিক্লেয়ার করার শেষে একই সাথে একটি ভেরিয়েবলও ডিক্লেয়ার করা হলো। ইউজার এই নিয়মে ইচ্ছে করলে একাধিক ভেরিয়েবল ডিক্লেয়ার করতে পারেন।

এভাবে দুটি নিয়মের মাধ্যমেই যেকোনো সংখ্যক স্ট্রাকচার ভেরিয়েবল, অ্যারে কিংবা পয়েন্টার ডিক্লেয়ার করা সম্ভব।

স্ট্রাকচার ডিক্লেয়ার করার আরেকটি নিয়ম হলো ট্যাগ না দেয়া। ইউজার যদি ট্যাগ না দেন তাহলে স্ট্রাকচার ডিক্লেয়ার করার সময়ই তার ভেরিয়েবল ডিক্লেয়ার করতে হবে। তবে ট্যাগবিহীন স্ট্রাকচার ডিক্লেয়ার করার পর যদি অন্য কোনো স্কোপে ওই একই টাইপের ভেরিয়েবল ডিক্লেয়ার করতে হয় তাহলে স্ট্রাকচারটিকে আবার ডিক্লেয়ার করতে হবে। তাই প্রতিটি স্ট্রাকচারের ট্যাগ লেখা উচিত। তাহলে পরে কোড পড়তে সুবিধা হয়।

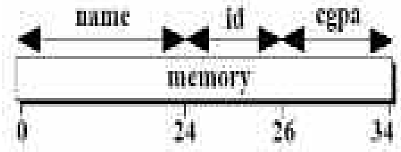
স্ট্রাকচার ভেরিয়েবলের জন্য মেমরিতে জায়গা নির্ধারণ : যেকোনো ভেরিয়েবলকে প্রোগ্রামে চলার উপযোগী করে তোলার জন্য অবশ্যই মেমরিতে তার জন্য নির্দিষ্টসংখ্যক জায়গা নির্ধারণ করতে হবে। স্ট্রাকচার ভেরিয়েবলের ক্ষেত্রেও একই নিয়ম প্রযোজ্য। একটি স্ট্রাকচার ভেরিয়েবল মেমরিতে কতটুকু জায়গা দখল করবে তা নির্ধারণ করে তার মেম্বারদের ওপর। সব মেম্বারের জন্য দখল করা জায়গা হবে ওই স্ট্রাকচারের একটি ভেরিয়েবলের দখল করা জায়গা। উপরের ডিক্লেয়ার করা স্টুডেন্ট স্ট্রাকচারের var ভেরিয়েবলটি কীভাবে মেমরিতে জায়গা দখল করে তা চিত্র-১-এ দেখানো হলো।

প্রথমে নেম অ্যারের জন্য ২৪ বাইট অ্যালোকেট করা হলো, কারণ একটি ক্যারেক্টার ১ বাইট জায়গা নেয় আর স্ট্রাকচারটিতে ২৪টি ক্যারেক্টারের একটি অ্যারে মেম্বার হিসেবে আছে। তারপর আইডির জন্য ২ বাইট, কারণ একটি ইন্টিজার ২ বাইট জায়গা নেয়। এরপর সিজিপিএ’র জন্য ৮ বাইট, কারণ এটি ডাবল টাইপের। আর ডাবল টাইপ ভেরিয়েবল ৮ বাইট জায়গা নেয়।

স্ট্রাকচার মেম্বার ও মেম্বার অপারেটর : উপরে দেখানো হলো কীভাবে স্ট্রাকচার ডিক্লেয়ার ও তার মেম্বার ডিক্লেয়ার করতে হয়। ইউজার ইচ্ছে করলে স্ট্রাকচার ভেরিয়েবল ডিক্লেয়ার করার পর তার প্রতিটি মেম্বারকে আলাদাভাবে ব্যবহার করতে পারেন। তবে এ ক্ষেত্রে খেয়াল রাখতে হবে সাধারণ ভেরিয়েবলের মতো স্ট্রাকচার ভেরিয়েবলের মেম্বারকে ব্যবহার করা যায় না। মেম্বারকে ব্যবহার করতে হলে আগে স্ট্রাকচার ভেরিয়েবলের নাম দিতে হবে। যেমন : ইউজার যদি উপরে ডিক্লেয়ার করা var ভেরিয়েবলের প্রতিটি মেম্বারের ভ্যালু অ্যাসাইন করতে চান তাহলে তা নিচের নিয়মানুসারে করা সম্ভব।

```
var.name=”this is a name”;
var.id=42;
var.cgpa=2.86965;
```

এখানে প্রতিটি মেম্বারকে আলাদাভাবে ব্যবহার করা হয়েছে। কিন্তু একটি ছোট সমস্যা হতে পারে। নেম ভেরিয়েবল একটি ক্যারেক্টার অ্যারে, যার মোট এলিমেন্ট সংখ্যা ২৪। কিন্তু মান অ্যাসাইনের সময় স্পেসসহ মোট ১৪টি ক্যারেক্টার অ্যাসাইন করা হয়েছে। সুতরাং বাকি ১০টি এলিমেন্টে গারবেজ মান থাকতে পারে। গারবেজ মান যাতে না থাকে সেজন্য ক্যারেক্টার অ্যারের সর্বশেষে একটি নাল ক্যারেক্টার অ্যাসাইন করা যায়। অথবা ক্যারেক্টার অ্যারেটি ডিক্লেয়ার করার সময় কোনো এলিমেন্ট সংখ্যা ব্যবহার না করলেও হয়। সে ক্ষেত্রে যতগুলো ক্যারেক্টার ওই অ্যারেতে অ্যাসাইন করা হবে, অ্যারেটির এলিমেন্ট সংখ্যা স্বয়ংক্রিয়ভাবে ততগুলো হয়ে যাবে।



চিত্র : মেমরিতে স্ট্রাকচারের গঠন

স্ট্রাকচার মেম্বার ইনপুট/প্রিন্ট : একটি স্ট্রাকচার ভেরিয়েবল ইনপুট নিতে হলে বা প্রিন্ট করতে হলে সাধারণ ভেরিয়েবলের নিয়মানুসারে তার প্রতিটি মেম্বারকে ব্যবহার করতে হবে। যেমন : var ভেরিয়েবলের ইনপুট ও প্রিন্ট করার পদ্ধতি নিচে দেখানো হলো :

```
scanf(“%s”,&var.name);
scanf(“%d”,&var.id);
scanf(“%f”,&var.cgpa);
printf(“%s”,var.name);
printf(“%d”,var.id);
printf(“%f”,var.cgpa);
```

সাধারণ ভেরিয়েবলের মতোই ইনপুট নেয়া ও প্রিন্ট করতে হয়। শুধু পার্থক্য হলো এ ক্ষেত্রে স্ট্রাকচার ভেরিয়েবলের নাম এবং সাথে মেম্বার অপারেটর ব্যবহার করতে হয়। এখানে (.) অপারেটর হলো মেম্বার অপারেটর।

স্ট্রাকচার তথা কাস্টম ডাটা টাইপের ব্যবহার সি ল্যাঙ্গুয়েজের একটি অন্যতম ফিচার। এটি ব্যবহারের মাধ্যমে ইউজারের কোড করা যেমন সহজ হয়ে যায়, তেমনি প্রোগ্রামের গুণগত মানও অনেক বেড়ে যায়। তাছাড়া বড় ধরনের প্রজেক্ট করার সময় স্ট্রাকচারের তথা কাস্টম ডাটা টাইপ ব্যবহারের কোনো বিকল্প নেই কজ

ফিডব্যাক : wahid_cseaust@yahoo.com