সহজ ভাষায় প্রোগ্রামিং সি/সি++

আহমদ ওয়াহিদ মাসুদ

প্রোগ্রামিংয়ে প্রথম কথা হলো, এর অ্যালগরিদম বানানোর কোনো নির্দিষ্ট নিয়ম নেই। অ্যালগরিদম হলো যেকোনো কাজ করার পদ্ধতি। শুনতে ভিন্ন ধরনের মনে হলেও আসলে এটিই সত্যি। একজন প্রোগ্রামার যেকোনো পদ্ধতিতে কোনো সমস্যার সমাধান দিতে পারেন অথবা একজন কোডার নিজের ইচ্ছেমতো কোনো প্রোগ্রামের কোড লিখতে পারেন। সবশেষে তা কাজ করে কি না সেটিই মূল কথা। তবে প্রোগ্রামের গুণগত মান যদি বিচার করতে বলা হয়, সেটি অবশ্য ভিন্ন ব্যাপার। সেক্ষেত্রে দেখা হয় কোন অ্যালগরিদম দিয়ে কোড করলে প্রোগ্রাম সবচেয়ে দ্রুত কাজ করে বা সবচেয়ে কম রিসোর্স ব্যবহার করে ইত্যাদি। এসবের জন্যই পয়েন্টার এবং আরও অনেক ফিচারের আবির্ভাব হয়েছে। গত পর্বে বিভিন্ন ধরনের পয়েন্টার নিয়ে আলোচনা করা হলেও শুধু নাল পয়েন্টারই ব্যাখ্যা করা হয়েছিল। এ পর্বে অন্যান্য পয়েন্টার, পয়েন্টার এবং অ্যারের মাঝে সম্পর্ক ও পার্থক্য নিয়ে আলোচনা করা হয়েছে।

ভয়েড পয়েন্টার

সোজা কথায়, ভয়েড পয়েন্টার হলো এমন এক ধরনের বিশেষ পয়েন্টার, যার ডাটা হিসেবে ক্যাস্টিং ছাড়াই অন্য যেকোনো টাইপের ভেরিয়েবলের অ্যাড্রেস নির্ধারণ করা যায়। অর্থাৎ এ ধরনের পয়েন্টারের মাধ্যমে যেকোনো টাইপের ভেরিয়েবল ক্যাস্টিং ছাড়াই পয়েন্ট করা যায়। এ ধরনের পয়েন্টার ডিক্লেয়ার করার নিয়ম হলো :

```
void *poiter_name;
```

দেখা যাচ্ছে এ ধরনের পয়েন্টার ডিক্লেয়ার করার জন্য ডাটা টাইপ হিসেবে ভয়েড কীওয়ার্ড ব্যবহার করতে হয়। আমরা জানি যে int *p মানে হলো p পয়েন্টারটি যেকোনো ইন্টিজারকে পয়েন্ট করবে। একইভাবে double *p মানে হলো p পয়েন্টারটি যেকোনো ডাবলকে পয়েন্ট করবে। একইভাবে void *p মানে হলো p এমন একটি পয়েন্টার, যা কি না ইন্টিজার কিংবা ফ্লোট কিংবা ডাবল সব ধরনের ভেরিয়েবলকেই ক্যাস্টিং ছাড়া পয়েন্ট করবে। পয়েন্টার যদি ভয়েড টাইপ না হতো, সেক্ষেত্রে ক্যাস্টিংয়ের মাধ্যমে এক টাইপের পয়েন্টার দিয়ে অন্য টাইপের ভেরিয়েবলকে পয়েন্ট করা যেত। খেয়াল রাখতে হবে, সাধারণ ভেরিয়েবলের ডাটা টাইপ হিসেবে ভয়েড ব্যবহার করলে তার মাঝে কোনো ডাটা না থাকা বোঝায় (যেটি সাধারণত দেখা যায় না, কারণ ভেরিয়েবলের মূল উদ্দেশ্যই হলো ডাটা রাখা)। আর কোনো ফাংশনের ডাটা টাইপ ভয়েড হলে তা কোনো মান রিটার্ন করে না। অর্থাৎ তখনই ভয়েড ডাটা টাইপটি ব্যবহার করা হয়, যখন কোনো মানের দরকার হয় না। কিন্তু পয়েন্টারের ক্ষেত্রে সম্পূর্ণ ভিন্ন একটি ঘটনা ঘটছে। এক্ষেত্রে যখন সব ধরনের ডাটার দরকার হয়, তখন ভয়েড ব্যবহার করা হয়। এবার ভয়েড পয়েন্টারের উদাহরণস্বরূপ ছোট একটি প্রোগ্রাম দেয়া হলো :

```
int x=10;
double y=3.12;
void *ptr;
ptr=&x;
```

এখানে ptr হলো ভয়েড টাইপের একটি পয়েন্টার এবং এর জন্য ইন্টিজার টাইপের ভেরিয়েবল x-এর অ্যাড্রেস নির্ধারিত হবে, অর্থাৎ ptr পয়েন্টারটি x-কে পয়েন্ট করবে। একইভাবে,

```
ptr=&y;
```

এক্ষেত্রে ptr-এর জন্য ডাবল টাইপের ভেরিয়েবল y-এর অ্যাড্রেস নির্ধারিত হবে, তথা ptr পয়েন্টারটি y-কে পয়েন্ট করবে। আবার ভয়েড পয়েন্টারের জন্য অন্য টাইপের ডাটাকেও অ্যাসাইন করা যায়। যেমন :

```
int x=20;
int *ip;
void *vp;
ip=&x;
vp=ip;
```

এখানে প্রথমে x একটি ইন্টিজার ভেরিয়েবল ডিক্লেয়ার করা হয়েছে, যার মান হিসেবে ২০ নির্ধারিত করা হয়েছে। এরপরই ইন্টিজার এবং ভয়েড টাইপের দুটি পয়েন্টার যথাক্রমে ip এবং vp ডিক্লেয়ার করা হয়েছে। সুতরাং ip দিয়ে x-কে পয়েন্টার করা যাবে। এখন vp যেহেতু ভয়েড টাইপের পয়েন্টার, তাই এটি দিয়ে সবাইকেই পয়েন্ট করার কথা। আবার ভয়েড পয়েন্টার যে শুধু যেকোনো ভেরিয়েবলকেই পয়েন্ট করবে এমনটি নয়, সেটি যেকোনো টাইপের পয়েন্টারের পয়েন্টেড ডাটাকেও পয়েন্ট করতে সক্ষম। তাই ওপরের কোডের একদম শেষে vp পয়েন্টারটি ip-এর ডাটা তথা x-কে পয়েন্ট করছে। এক্ষেত্রে কম্পাইলার কোনো এরর দেখাবে না। এখানে vp-এর জায়গায় অন্য কোনো পয়েন্টার ব্যবহার করলে ক্যাস্টিংয়ের প্রয়োজন হতো। কিন্তু ভয়েড পয়েন্টারের বেলায় কিছুরই দরকার হয় না। এভাবে ক্যাস্টিং ছাড়াই ভয়েড পয়েন্টারের জন্য অন্য টাইপের পয়েন্টারকে কিংবা অন্য ডাটা টাইপের পয়েন্টারের জন্য ভয়েড পয়েন্টারকে অ্যাসাইন করা যায়। তবে ভয়েড পয়েন্টারের মাধ্যমে পয়েন্টেড অ্যাড্রেসের ডাটা নিয়ে কাজ করতে চাইলে পয়েন্টেড ডাটাকে এভাবে ক্যাস্ট করতে হবে :

```
*(pointed_data_type *) void_ptr;
```

যেমন :

```
int x=10,y;
void *ptr;
ptr=&x;
y=*(int*)ptr;
```

এখানে প্রথমে ptr-এর জন্য x-এর অ্যাড্রেস নির্ধারণ করা হয়েছে। পরে ptr-এর মাধ্যমে ইন্টিজার টাইপ ডাটা পড়ার জন্য ptr-কে int*-এ ক্যাস্ট করা হয়েছে। এভাবে ভয়েড পয়েন্টারের মাধ্যমে পয়েন্টেড অ্যাড্রেসের ডাটা নিয়ে কাজ করতে হলে পয়েন্টেড ডাটাকে উপযুক্ত পয়েন্টার টাইপে ক্যাস্ট করে নিতে হয়। তবে একটি উল্লেখযোগ্য বিষয় হলো, ভয়েড পয়েন্টারের মাধ্যমে পয়েন্টেড ডাটাকে যে টাইপে ক্যাস্ট করা হবে, আউটপুটে সেই টাইপ অনুযায়ী ডাটা পাওয়া যাবে। ভয়েড পয়েন্টারকে জেনেরিক পয়েন্টারও বলা হয়। ভয়েড পয়েন্টারকে অন্য কোনো টাইপে ক্যাস্ট না করা পর্যন্ত ইনক্রিমেন্ট, ডিক্রিমেন্ট কিংবা অন্য কোনো এক্সপ্রেশনের অপারেন্ড হিসেবে ব্যবহার করা যায় না।

কনস্ট্যান্ট পয়েন্টার

const কীওয়ার্ডকে পয়েন্টার ডিক্লেয়ার করার সময় বিভিন্নভাবে ব্যবহার করা যায়। তবে প্রথমে দেখা যাক নন-পয়েন্টার ভেরিয়েবল ডিক্লেয়ার করার সময় এ কীওয়ার্ড ব্যবহার করলে কী হয়,

```
const i=10;
```

এখানে const কীওয়ার্ডের মাধ্যমে কম্পাইলারকে জানিয়ে দেয়া হচ্ছে, i হলো একটি কনস্ট্যান্ট ভেরিয়েবল। তাই একই স্কোপের মাঝে প্রোগ্রামের অন্য কোথাও এ ভেরিয়েবলের ডাটা পরিবর্তন করলে কম্পাইলের প্রোগ্রাম কম্পাইল করার সময় এরর দেখাবে। অর্থাৎ কনস্ট্যান্ট ভেরিয়েবলের মান সাধারণত পরিবর্তন করা যায় না।

এভাবে কোনো ভেরিয়েবলের মান অপরিবর্তনীয় রাখতে হলে তাকে কনস্ট্যান্ট হিসেবে ডিক্লেয়ার করতে হয়। একইভাবে প্রোগ্রামে পয়েন্টারকে কনস্ট্যান্ট করার জন্যও const কীওয়ার্ড ব্যবহার করা হয়। কনস্ট্যান্ট কীওয়ার্ড দুইভাবে ব্যবহার হতে পারে। যেমন :

```
const datatype *pointerName = value;
```

A_ev

```
datatype * const pointerName = value;
```

অর্থাৎ const কীওয়ার্ডটি ডাটা টাইপের আগে বা পরে উভয় স্থানেই বসতে পারে। তবে এটি কিন্তু একই বিষয় নয়। দেখা যাক আগে বসালে কী হয় আর পরে বসালে কী হয়।

যদি ডাটা টাইপের আগে কীওয়ার্ড বসানো হয়, তাহলে পয়েন্টার যাকে পয়েন্ট করবে তার ▶

মান কনস্ট্যান্ট থাকবে। অর্থাৎ পয়েন্টারের নিজের মান পরিবর্তন করা যাবে, কিন্তু যাকে পয়েন্ট করা হচ্ছে তার মান পরিবর্তন করা যাবে না। অন্যভাবে বলতে গেলে, পয়েন্টারের মাধ্যমে পয়েন্টেড ডাটা পড়া যাবে, কিন্তু পরিবর্তন করা যাবে না। এভাবেই উইন্ডোজে কোনো ফাইলকে রিড অনলি করা হয়, যাতে ফাইলটি শুধু পড়া যাবে, কিন্তু পরিবর্তন করা যাবে না। যেমন :

```
int i=10,j;
const int *ptr;
ptr=&i;
j=*ptr;
*ptr=20;
```

এখানে j-এর জন্য i-এর ডাটা নির্ধারণ করা গেলেও *ptr=20; এই স্টেটমেন্টের মাধ্যমে i-এর জন্য ২০ নির্ধারণ করা যাবে না। কেননা পয়েন্টারকে ডিক্লেয়ার করার সময় কম্পাইলারকে জানিয়ে দেয়া হয়েছে পয়েন্টারটি যাকে পয়েন্ট করবে তার মান অপরিবর্তিত থাকবে। তাই শেষের লাইনে *ptr-এর মাধ্যমে পয়েন্টেড ডাটাকে পরিবর্তন করতে চাইলে কম্পাইলার এরর দেখাবে। সুতরাং শেষে বলা যায়, cons int *ptr; এর মানে হলো পয়েন্টারের জন্য যে ইন্টিজার ভেরিয়েবলের অ্যাড্রেস নির্ধারণ করা হবে, পয়েন্টারের মাধ্যমে সেই ভেরিয়েবলের ডাটা শুধু পড়া যাবে, কিন্তু পরিবর্তন করা যাবে না।

কিন্তু const কীওয়ার্ডটিকে যদি ডাটা টাইপের পরে ব্যবহার করা হয়, যেমন : Int * const ptr; তাহলে পয়েন্টারটি কনস্ট্যান্ট থাকবে। অর্থাৎ পয়েন্টারের নিজের মান পরিবর্তন করা যাবে না, কিন্তু পয়েন্টারটি যাকে পয়েন্ট করছে তাকে পরিবর্তন করা যাবে। একটি ছোট প্রোগ্রাম উদাহরণ হিসেবে দেয়া হলো :

```
int i=10,j;
int * const ptr=&i;
j=*ptr;
*ptr=20;
ptr=&j;
```

এখানে ব্যবহৃত পয়েন্টারটি কনস্ট্যান্ট, অর্থাৎ পয়েন্টারের জন্য অন্য কোনো ভেরিয়েবলের ডাটা নির্ধারণ করা যাবে না বা পয়েন্টারের ডাটা অপরিবর্তিত থাকবে। কিন্তু পয়েন্টারের মাধ্যমে পয়েন্টেড ডাটার মান পড়া যাবে (তৃতীয় লাইন) এবং প্রয়োজনে তা পরিবর্তন করা যাবে (চতুর্থ লাইন)। পরিশেষে বলা যায়, int * const ptr মানে হলো পয়েন্টার ভেরিয়েবলটি কনস্ট্যান্ট। তাই এ পয়েন্টার দিয়ে অন্য কাউকে পয়েন্টার করা যাবে না। আর const int *ptr মানে হলো পয়েন্টারটি যাকে পয়েন্ট করছে তার মান কনস্ট্যান্ট। তাই পয়েন্টেড ডাটাকে পরিবর্তন করা যাবে না, কিন্তু পয়েন্টারটি দিয়ে ইচ্ছে করলে অন্য কাউকে পয়েন্ট করা যাবে। কারণ পয়েন্টারটি কাকে পয়েন্ট করছে না করছে– সেটি পয়েন্টারের নিজের ডাটা, পয়েন্টেড ডাটা নয়। তবে এটিই শেষ নয়। ব্যবহারকারী প্রয়োজন হলে উভয় পাশেই const কীওয়ার্ড ব্যবহার করতে পারেন। যেমন const int * const ptr=&i; এক্ষেত্রে পয়েন্টারের মান এবং পয়েন্টেড ডাটার মান উভয়ই অপরিবর্তনীয় থাকবে। অর্থাৎ পয়েন্টার দিয়ে যেমন অন্য কাউকে পয়েন্ট করা যাবে না, তেমন পয়েন্টারটি দিয়ে যাকে পয়েন্ট করা হচ্ছে তার মানও পরিবর্তন করা যাবে না।

পয়েন্টারের পয়েন্টার

আমরা জানি, পয়েন্টার হলো একটি বিশেষ ধরনের ভেরিয়েবল, যা অন্য ভেরিয়েবলকে পয়েন্ট করে বিভিন্ন কাজে বিভিন্নভাবে ব্যবহার করে। ভেরিয়েবল যেমন বিভিন্ন টাইপের হয়, তেমনি পয়েন্টারও ইন্টিজার, ডাবল ইত্যাদি টাইপের হয়। যেমন int *i; এখানে i একটি পয়েন্টার, যা ইন্টিজার টাইপের ভেরিয়েবলকে পয়েন্ট করতে সক্ষম। এভাবে প্রোগ্রামে যেমন বিভিন্ন ভেরিয়েবলের জন্য পয়েন্টার ডিক্লেয়ার করা যায়, তেমনি বিভিন্ন পয়েন্টারের জন্যও পয়েন্টার ডিক্লেয়ার করা যায়। অর্থাৎ এক্ষেত্রে পয়েন্টার কোনো ভেরিয়েবলকে পয়েন্ট করবে না, বরং অন্য একটি পয়েন্টারকে পয়েন্ট করবে। যেমন int **p; এখানে p হলো একটি ইন্টিজার টাইপের পয়েন্টারের পয়েন্টার। অর্থাৎ p দিয়ে এমন পয়েন্টারকে পয়েন্ট করা যাবে, যা কি না কোনো ইন্টিজারকে পয়েন্ট করে। আবার char **c; এখানে c হলো একটি ক্যারেক্টার টাইপের পয়েন্টারের পয়েন্টার। লক্ষণীয়, একটি নির্দিষ্ট টাইপের ভেরিয়েবলকে যতগুলো ভেরিয়েবল পয়েন্ট করবে, এদের সবই একই টাইপের হতে হবে (ব্যতিক্রম ভয়েড টাইপ)। এমন কোনো পয়েন্টার হয় না যেটি নিজে ইন্টিজার টাইপের, কিন্তু পয়েন্ট করছে ডাবল টাইপের একটি পয়েন্টারকে। এভাবে একাধিক * সাইন ব্যবহার করে প্রয়োজনমতো পয়েন্টার বানিয়ে নেয়া যায়। যেমন int ***p; এখানে তিনটি * সাইন ব্যবহার করা হয়েছে। যার অর্থ p এমন একটি পয়েন্টার, যা ইন্টিজার টাইপের ভেরিয়েবলের পয়েন্টারের পয়েন্টারকে পয়েন্ট করে।

পয়েন্টার ও অ্যারে

সি-তে পয়েন্টার ও অ্যারের মাঝে একটি ঘনিষ্ঠ সম্পর্ক রয়েছে। অ্যারের অনেক কাজই পয়েন্টারের মতো। আবার পয়েন্টারের গঠন অনেকটা অ্যারের মতো বলা যায়। অ্যারে নিয়ে আগের সংখ্যায় আলোচনা করা হয়েছে। তবে পয়েন্টার ও অ্যারে নিয়ে আলোচনা করার জন্য অ্যারের সব ফিচার মনে রাখতে হবে। তাই আগে অ্যারে সম্পর্কে সংক্ষিপ্ত ধারণা দেয়া হলো।

অ্যারে সম্পর্কে সংক্ষিপ্ত ধারণা

অ্যারে মূলত হাই লেভেল ল্যাঙ্গুয়েজের অন্যতম ফিচার এবং এটি সি-এর অন্যতম প্রধান ফিচারগুলোর একটি। অ্যারে কী তা বোঝার জন্য একটু পেছনে ফিরে যেতে হবে। আমরা জানি, সি-তে ভেরিয়েবল ডিক্লেয়ার করা যায়। ভেরিয়েবল কী, তা ডিক্লেয়ার করলে কী হয় এবং তা কীভাবে কাজ করে তাও আমরা জানি। অ্যারের মূল ধারণা নতুন কিছুই নয়। অ্যারে হলো অনেকগুলো ভেরিয়েবল একসাথে ডিক্লেয়ার করার একটি পদ্ধতি। ধরুন, কোনো প্রোগ্রামে একইসাথে পাঁচটি ভেরিয়েবল ডিক্লেয়ার করার প্রয়োজন হলো। তাহলে ব্যবহারকারী সাধারণ নিয়মে পাঁচটি ভেরিয়েবল ডিক্লেয়ার করতে পারেন। এজন্য পাঁচটি স্টেটমেন্ট লেখার প্রয়োজন হবে, আবার একটি স্টেটমেন্টেও পাঁচটি ভেরিয়েবল ডিক্লেয়ার করা যায়। কিন্তু সেক্ষেত্রে স্টেটমেন্টটি অনেক বড় হবে। কিন্তু অ্যারে ব্যবহার করে পাঁচটি ভেরিয়েবল একই সাথে অর্থাৎ একটি স্টেটমেন্ট দিয়েই ডিক্লেয়ার করা সম্ভব। মাত্র পাঁচটি ভেরিয়েবলের ক্ষেত্রে হয়তো এটি তেমন গুরুত্বপূর্ণ বিষয় নয়, কিন্তু অনেক বড় প্রোগ্রামে একইসাথে যখন ১০০ বা ১০০০ ভেরিয়েবল ডিক্লেয়ার করার প্রয়োজন হবে, তখন অ্যারে ব্যবহার করলে কোডিং অনেক সহজ হয়ে যায়। অ্যারে হলো কতগুলো ভেরিয়েবলের সমষ্টি। সুতরাং ভেরিয়েবল ডিক্লেয়ারের মতো করেই অ্যারে ডিক্লেয়ার করতে হয়। ভেরিয়েবল ডিক্লেয়ারের আগে যেমন ডাটা টাইপ লেখার দরকার, অ্যারের জন্যও তেমনি দরকার। অ্যারে ডিক্লেয়ার করার সিনটেক্স : data_type array_name[array_size]| অ্যারের নাম হলো যেকোনো ভেরিয়েবলের নাম। অর্থাৎ ভেরিয়েবলের নামের নিয়মানুসারে অ্যারের নাম করতে হবে। এখানে নতুন বিষয় হলো অ্যারে সাইজ। যেহেতু অ্যারে হলো অনেকগুলো ভেরিয়েবলের সমষ্টি। সুতরাং কতগুলো ভেরিয়েবল নিয়ে অ্যারে ডিক্লেয়ার করা হচ্ছে সেটা বলতে হবে। এটিই হলো অ্যারে সাইজ। সাইজ যত হবে ততগুলো ভেরিয়েবল নিয়ে অ্যারে গঠিত হবে। অ্যারে ও ভেরিয়েবল ডিক্লেয়ারের মাঝে মূল পার্থক্য হলো সাইজ। যেমন int prime[10], valid[5]; ইত্যাদি। এখানে একই সাথে প্রাইম নামে দশটি, ভ্যালিড নামে পাঁচটি ইন্টেজার ডিক্লেয়ার করা হয়েছে। আর প্রাইম নামে যে ১০টি ভেরিয়েবল ডিক্লেয়ার করা হয়েছে তাদের আলাদাভাবেও অ্যাক্সেস করা যাবে। সেক্ষেত্রে prime[0], prime[1], prime[2] ইত্যাদি হবে একেকটি ভেরিয়েবলের নাম। প্রাইম নামের পর যে [] বন্ধনীর ভেতরে সংখ্যা ব্যবহার করা হয়েছে তাকে অ্যারের এলিমেন্টের ইন্ডেক্স বলে। এটি দিয়ে একটি অ্যারের সব এলিমেন্টকে অ্যাক্সেস করা যায়। অ্যারের ইন্ডেক্সিং ০ থেকে শুরু হয়।

পয়েন্টার বিভিন্ন ধরনের হয় এবং এদের কাজও বিভিন্ন ধরনের। যেমন শুধু কনস্ট্যান্ট পয়েন্টার ব্যবহার করেই ফাইল রিড অনলি হবে কি না তা নিয়ন্ত্রণ করা যায়। এ ধরনের প্রতিটি পয়েন্টারেরই নির্দিষ্ট ব্যবহার রয়েছে। আগামী পর্বে পয়েন্টার ও অ্যারে নিয়ে বিস্তারিত আলোচনা করার হবে কজ

ফিডব্যাক : wahid_cseaust@yahoo.com