সহজ ভাষায় প্রোগ্রামিং সি/সি++

আহমদ ওয়াহিদ মাসুদ

সি ল্যাঙ্গুয়েজের যত গভীরে যাওয়া যায় ততই ল্যাঙ্গুয়েজটি একদিকে যেমন সহজ হয়ে ওঠে, অন্যদিকে তেমন কঠিন হয়ে ওঠে। যেমন গত পর্বে দেখানো হয়েছে, কিভাবে ফাংশনের রিকার্সন ব্যবহার করে কোনো সমস্যা সমাধান করা যায়। রিকার্সন সি ল্যাঙ্গুয়েজের অ্যাডভান্সড ফিচারের একটি। কিভাবে রিকার্সন কাজ করে তা বোঝা এবং রিকার্সন নিয়ে লজিক করানো খুব কঠিন একটি ব্যাপার। কেউ যদি রিকার্সন দিয়ে কোনো সমস্যা সমাধান করে থাকেন তাহলে বুঝতে হবে তিনি একজন অ্যাডভান্সড সি প্রোগ্রামার। আর রিকার্সন দিয়ে তৈরি লজিকের কোড আকারে অনেক ছোট। অর্থাৎ সি দিয়ে কোড করা অনেক সহজ হয়ে যাচ্ছে।

ফাংশনের ওপর আলোচনা প্রায় শেষ। ছোট একটি টপিক দিয়ে ফাংশন সম্পর্কিত আলোচনা শেষ করা হবে। ফাংশনের জন্য বিভিন্ন ভেরিয়েবলের স্কোপ কী ধরনের হতে পারে তা নিয়ে এখন আলোচনা করা হবে।

ভেরিয়েবলের স্কোপ কী তা আমরা জানি। কোডের যে অংশ জুড়ে একটি ভেরিয়েবলের কর্মক্ষমতা বিস্তৃত, সে অংশকে ওই ভেরিয়েবলের স্কোপ বলে। স্কোপ অনুযায়ী ভেরিয়েবল দুই ধরনের হতে পারে। লোকাল এবং গ্লোবাল ভেরিয়েবল। এছাড়া এক বিশেষ ধরনের ভেরিয়েবল আছে, যাকে স্ট্যাটিক ভেরিয়েবল বলা হয়।

লোকাল ভেরিয়েবল

যখন কোনো ফাংশনের মধ্যে বা কোনো কোড ব্লকের মধ্যে কোনো ভেরিয়েবলকে ডিক্লেয়ার করা হয় তখন তাকে সংশ্লিষ্ট ফাংশনের বা ব্লকের লোকাল ভেরিয়েবল বলা হয়। ভেরিয়েবলটির অস্তিত্ব, কার্যপরিধি শুধু ওই ফাংশনের মধ্যে সীমাবদ্ধ থাকে। ফাংশনের বাইরে গেলে প্রোগ্রামে ওই ভেরিয়েবলের আর কোনো অস্তিত্ব থাকে না। অর্থাৎ ভেরিয়েবলটির স্কোপ হলো ওই নির্দিষ্ট ফাংশন। স্কোপের বাইরে ওই ভেরিয়েবলকে ব্যবহার করা হলে প্রোগ্রাম এরর দেখাবে। তাছাড়া লোকাল ভেরিয়েবলকে অন্য কেউ ব্যবহারও করতে পারে না। সহজ কথায়, লোকাল ভেরিয়েবলকে তার স্কোপের বাইরে ব্যবহার করা যায় না। আমরা জানি, প্রোগ্রামে একই নামে একাধিক ভেরিয়েবল থাকতে পারে না। কিন্তু স্কোপ অনুযায়ী একাধিক ভেরিয়েবল থাকতে পারে। যেমন কোনো প্রোগ্রামে দুটি ফাংশন আছে। একটি ফাংশন হলো outer() এবং এই আউটার ফাংশনের ভেতরে আরেকটি ফাংশন ডিক্লেয়ার করা আছে, যার নাম inner()। এখন এ দুটি ফাংশনের ভেতরেই যদি একই নামের ভেরিয়েবল ডিক্লেয়ার করা হয় এবং তাদের নিয়ে ভিন্ন ভিন্ন কাজ করা হয় তাহলে প্রোগ্রাম কোনো এরর দেখাবে না। কেননা প্রোগ্রাম যখন কোনো ফাংশন নিয়ে কাজ করে তখন স্ট্যাকে ওই ফাংশনের ভেরিয়েবল এবং ইনস্ট্রাকশনগুলো রেখে দেয়। আর যখন কোনো ফাংশনের কাজ শেষ হয়ে যায়, তখন প্রোগ্রাম স্ট্যাক খালি করে দেয়। অন্য কথায়, একটি ফাংশনের কাজ শেষ হয়ে যাওয়ার অর্থ হলো তার সব প্রোপার্টি ডিলিট করে দেয়া। প্রোগ্রাম যেহেতু একই সাথে একাধিক ফাংশনের কাজ করে না, একটি একটি করে এক্সিকিউট করে, তাই একই নামে ভিন্ন স্কোপবিশিষ্ট একাধিক লোকাল ভেরিয়েবল থাকলেও প্রোগ্রাম তাদেরকে আলাদা আলাদাভাবে এক্সিকিউট করে। তাই কোনো এরর দেখায় না। একই নিয়ম বিভিন্ন কোড ব্লকের জন্যও প্রযোজ্য।

গ্লোবাল ভেরিয়েবল

লোকাল ভেরিয়েবল বোঝা গেলে গ্লোবাল ভেরিয়েবলও বোঝা কঠিন কিছু হবে না। অনেক দিক দিয়েই লোকাল ভেরিয়েবলের উল্টো হলো গ্লোবাল ভেরিয়েবল। লোকাল ভেরিয়েবলের স্কোপ যেমন একটি নির্দিষ্ট ফাংশন বা ব্লকের মধ্যে সীমাবদ্ধ, গ্লোবাল ভেরিয়েবলের স্কোপ হলো সম্পূর্ণ প্রোগ্রাম জুড়ে। অর্থাৎ প্রোগ্রামের যেখানেই বা যেকোনো ব্লকের ভেতরেই সেই গ্লোবাল ভেরিয়েবলকে ব্যবহার করা হোক না কেনো, প্রোগ্রাম কোনো এরর দেখাবে না। আর তাই একই নামে একাধিক গ্লোবাল ভেরিয়েবল থাকতে পারে না। একই নামে একাধিক গ্লোবাল ভেরিয়েবল ডিক্লেয়ার করলেই প্রোগ্রাম এরর দেখাবে। ব্যাপারটি বোঝা খুবই সহজ। যেহেতু এটি গ্লোবাল ভেরিয়েবল, তাই স্ট্যাকে সবসময় এই ভেরিয়েবলটি উপস্থিত থাকে। আর তাই একই নামে অন্য আরেকটি ভেরিয়েবল ডিক্লেয়ার করতে গেলে স্ট্যাকে একই সাথে দুটি ভেরিয়েবলের অস্তিত্ব তৈরি হয়, তাই প্রোগ্রাম এরর দেখায়।

গ্লোবাল ভেরিয়েবলকে মেইন ফাংশনের বাইরে ডিক্লেয়ার করতে হয়। ভেরিয়েবল ডিক্লেয়ার করার নিয়ম সাধারণ ভেরিয়েবলের মতোই। খেয়াল রাখতে হবে গ্লোবাল ভেরিয়েবল আর কনস্ট্যান্ট ভিন্ন জিনিস। গ্লোবাল ভেরিয়েবল সাধারণত ছোট প্রোগ্রামে দরকার হয় না। তবে বড় আকারের প্রোগ্রামে গ্লোবাল ভেরিয়েবল ব্যবহার করলে প্রোগ্রামের জটিলতা কমে না, কিন্তু ইউজারের কষ্ট অনেক অংশেই কমে যায়। ধরা যাক, কোনো প্রোগ্রামে দশটি ফর লুপ আছে। প্রথম লুপটি দুইবার চলবে এবং প্রতিটি লুপ আগের লুপ থেকে একবার বেশি চলবে। অর্থাৎ প্রথম লুপ দুইবার, দ্বিতীয় লুপ তিনবার, তৃতীয় লুপ চারবার ইত্যাদি। তাহলে সাধারণত ফর লুপের কন্ডিশনে সরাসরি সংখ্যা বসিয়ে ইউজার লুপ সংখ্যা নিয়ন্ত্রণ করবেন। এখন ধরা যাক, প্রোগ্রামটির একটু পরিবর্তন আনা দরকার। তাই প্রথম লুপকে চারবার চালাতে হবে এবং পরের লুপগুলো আগের নিয়মেই চলবে। তাহলে এই ছোট পরিবর্তন আনার জন্য ইউজারকে সব ফর লুপের কন্ডিশনকে এডিট করতে হবে। দশটি ফর লুপের জন্য হয়তোবা এটি খুব একটা কঠিন কিছু নয়। তবে এ ধরনের কাজ যদি কোনো সফটওয়্যারের কোডে করতে হয়, যেখানে হাজার হাজার লাইনের কোড থাকে, তাহলে সেটি খুবই কষ্টসাধ্য এবং সময়সাপেক্ষ একটি ব্যাপার হয়ে দাঁড়াবে। গ্লোবাল ভেরিয়েবল ব্যবহার করে এ ধরনের কাজ খুব সহজেই করা সম্ভব। এই দশটি ফর লুপের উদাহরণের কথাই ধরা যাক। এখানে গ্লোবাল ভেরিয়েবল ব্যবহার করে কোডে মাত্র একটি লাইন পরিবর্তন করলে দশটি লুপের কন্ডিশন পরিবর্তন হয়ে যেত। যেখানে সাধারণভাবে কাজটি করতে গেলে দশটি লাইনই পরিবর্তন করতে হবে। প্রতিটি লুপের কন্ডিশনে সরাসরি কোনো মান না দিয়ে যদি ওই গ্লোবাল ভেরিয়েবল ব্যবহার করা হয়, তাহলে যেকোনো সময় শুধু ওই ভেরিয়েবলের মান পরিবর্তন করলেই দশটি লুপের কন্ডিশন পরিবর্তন হয়ে যাবে। তাছাড়া যখন সফটওয়্যারের জন্য হাজার হাজার লাইনের কোড লেখা হয় তখন গুরুত্বপূর্ণ অনেক ভেরিয়েবলই থাকে, যেগুলো প্রায়ই পরিবর্তন করতে হয়। সেই ভেরিয়েবলগুলো গ্লোবাল হিসেবে ডিক্লেয়ার করলে সহজেই খুঁজে পাওয়া যায় এবং প্রয়োজনমতো পরিবর্তন করা যায়।

এবারে লোকাল এবং গ্লোবাল ভেরিয়েবলের ব্যবহার সম্পর্কে ভালোভাবে জানার জন্য ছোট একটি প্রোগ্রাম উদাহরণ হিসেবে দেয়া হলো :

```
#include<stdio.h>
#include<conio.h>
int x =30;

void outer()
{
    int x=10;
    inner();
    printf(“the outer x is %d\n”,x);
printf(“the global z is %d\n”,z);
```

►

```
}

void inner()
{
    int x=20;
    printf("the inner x is %d\n",x);
printf("the global z is %d\n",z);
}

void main()
{
    outer();
    printf(Òthe global z is %d\nÓ,z);
    getch();
}
```

প্রোগ্রামটির আউটপুট :

```
the inner x is 20
the global x is 30
the outer x is 10
the global x is 30
the global x is 30
```

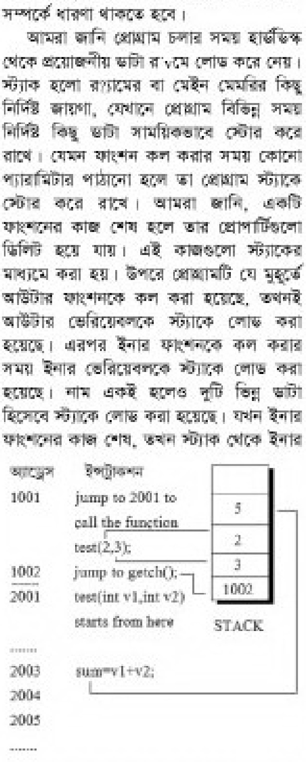
প্রোগ্রামটি দেখে সহজেই বোঝা যায়, লোকাল ও গ্লোবাল ভেরিয়েবলগুলো কিভাবে কাজ করছে। গ্লোবাল ভেরিয়েবল মেইন ফাংশনের বাইরে লেখা হয়। আর এখানে ফাংশনের প্রোটোটাইপ দেয়া হয়নি। কারণ, মেইন ফাংশনের আগে এই ইউজার ডিফাইন্ড ফাংশন দুটি লেখা হয়েছে। তবে ফাংশন দুটি যদি মেইন ফাংশনের পর লেখা হতো তাহলে মেইন ফাংশনের আগে এদের প্রোটোটাইপ না দিলে প্রোগ্রাম এরর দেখাত।

মেইন ফাংশনে প্রথমে আউটার ফাংশনকে কল করা হয়েছে। আউটার ফাংশনে একটি ভেরিয়েবলকে ডিক্লেয়ার করা হয়েছে। এটি একটি লোকাল ভেরিয়েবল, যার স্কোপ শুধু আউটার ফাংশনেই সীমাবদ্ধ। আউটার ফাংশনের এই ভেরিয়েবলটি কাজ করবে না। এরপর ইনার ফাংশনকে কল করা হয়েছে। লক্ষণীয়, যদিও ইনার ফাংশনকে আউটার ফাংশনের ভেতরে কল করা হয়েছে। তারপরও ইনার ফাংশন আউটার ফাংশনের কোনো প্রোপার্টি ব্যবহার করতে পারবে না। ইনার ফাংশনে আবার একই নামে আরেকটি ভেরিয়েবল ডিক্লেয়ার করা হয়েছে। এখানে প্রোগ্রাম কোনো এরর দেখাবে না, কারণ ভেরিয়েবল দুটির নাম এক হলেও স্কোপ ভিন্ন। এবার ইনার ভেরিয়েবলকে প্রিন্ট করা হয়েছে এবং সাথে সাথে গ্লোবাল ভেরিয়েবলকেও প্রিন্ট করা হয়েছে। আগেই বলা হয়েছে, গ্লোবাল ভেরিয়েবলকে পুরো প্রোগ্রামের সবাই ব্যবহার করতে পারে। প্রোগ্রামের যেকোনো জায়গা থেকে গ্লোবাল ভেরিয়েবলকে ব্যবহার করা যায়। ইনার ফাংশনের কাজ এখন শেষ। এবার প্রোগ্রাম আউটার ফাংশনে ফিরে গিয়ে আবার গ্লোবাল ভেরিয়েবল প্রিন্ট করবে। এবার প্রোগ্রাম মেইন ফাংশনে ফিরে গিয়ে আবারো গ্লোবাল ভেরিয়েবলকে প্রিন্ট করবে। এই প্রোগ্রামটিতে দেখানো হলো কোনো লোকাল ভেরিয়েবল তার নিজস্ব স্কোপ ছাড়া অন্য কোথাও কাজ করে না। আর গ্লোবাল ভেরিয়েবলকে সব জায়গা থেকেই এক্সিকিউট করা সম্ভব। আরেকটি খুব গুরুত্বপূর্ণ কথা জেনে রাখা ভালো যে লোকাল এবং গ্লোবাল ভেরিয়েবলেরও একই নাম হতে পারে। তবে কোনো ফাংশনে বা কোড ব-কে প্রিন্ট হওয়ার সময় লোকাল ভেরিয়েবলই প্রিন্ট হয়।

স্ট্যাক

স্ট্যাক সি ল্যাঙ্গুয়েজের একদম বেসিক এবং গুরুত্বপূর্ণ বিষয়গুলোর একটি। স্ট্যাক সব ল্যাঙ্গুয়েজেই ব্যবহার হয়। স্ট্যাক প্রোগ্রামিং ল্যাঙ্গুয়েজের কোনো বিষয় নয়, বরং এটি কমপিউটার আর্কিটেকচারের মধ্যে পড়ে। কমপিউটার আর্কিটেকচার বলতে প্রোগ্রাম কিভাবে বা কী কী নিয়মানুসারে কোড কম্পাইল করবে, তা বোঝানো হয়। যদিও স্ট্যাক প্রোগ্রামিং ল্যাঙ্গুয়েজের সাথে সরাসরি সম্পর্কযুক্ত নয়, তারপরও স্ট্যাক সম্পর্কে ভালোভাবে জানা অত্যন্ত জরুরি। তাহলে বোঝা যাবে প্রোগ্রামে ভেরিয়েবল, ফাংশনের বিভিন্ন প্যারামিটার, বিভিন্ন ইনস্ট্রাকশন কিভাবে কাজ করে। লোকাল ভেরিয়েবল এবং গ্লোবাল ভেরিয়েবলের মধ্যে যে মূল পার্থক্য, কিভাবে একই নামে ভিন্ন স্কোপবিশিষ্ট একাধিক লোকাল ভেরিয়েবল ব্যবহার হয়, তা ভালোভাবে বোঝার জন্য স্ট্যাক সম্পর্কে ধারণা থাকতে হবে।

আমরা জানি প্রোগ্রাম চলার সময় হার্ডডিস্ক থেকে প্রয়োজনীয় ডাটা র‍্যামে লোড করে নেয়। স্ট্যাক হলো র‍্যামের বা মেইন মেমরির কিছু নির্দিষ্ট জায়গা, যেখানে প্রোগ্রাম বিভিন্ন সময় নির্দিষ্ট কিছু ডাটা সাময়িকভাবে স্টোর করে রাখে। যেমন ফাংশন কল করার সময় কোনো প্যারামিটার পাঠানো হলে তা প্রোগ্রাম স্ট্যাকে স্টোর করে রাখে। আমরা জানি, একটি ফাংশনের কাজ শেষ হলে তার প্রোপার্টিগুলো ডিলিট হয়ে যায়। এই কাজগুলো স্ট্যাকের মাধ্যমে করা হয়। উপরে প্রোগ্রামটি যে মুহূর্তে আউটার ফাংশনকে কল করা হয়েছে, তখনই আউটার ভেরিয়েবলকে স্ট্যাকে লোড করা হয়েছে। এরপর ইনার ফাংশনকে কল করার সময় ইনার ভেরিয়েবলকে স্ট্যাকে লোড করা হয়েছে। নাম একই হলেও দুটি ভিন্ন ডাটা হিসেবে স্ট্যাকে লোড করা হয়েছে। যখন ইনার ফাংশনের কাজ শেষ, তখন স্ট্যাক থেকে ইনার ভেরিয়েবল ডিলিট করা হয়েছে। কিন্তু আউটার ভেরিয়েবলটি যেভাবে ছিল সেভাবেই আছে। কারণ আউটার ফাংশনের কাজ এখনো শেষ হয়নি। এভাবে একটি একটি কাজ বা ইনস্ট্রাকশন শেষ হয় আর সেই ইনস্ট্রাকশনের প্রোপার্টিগুলো স্ট্যাক থেকে ডিলিট করা হয়। ছোট একটি প্রোগ্রামের সাহায্যে স্ট্যাকের কাজ দেখানো হলো :



```
void test(int v1,int v2)
{
    int sum;
    sum=v1+v2;
    printf("%d + %d = %d",v1,v2,sum);
}
void main()
{
    test(2,3);
    getch();
}
```

এই প্রোগ্রামটা চালানোর সময় যখন টেস্ট ফাংশনটিকে কল করা হবে তখন স্ট্যাকের সাহায্যে নিচের কাজগুলো সম্পন্ন হবে :

০১. স্ট্যাকে প্রথমে রিটার্ন অ্যাড্রেসটি রাখা হবে। অর্থাৎ এই ফাংশন কল করার পর যে ইনস্ট্রাকশনটি আছে, সেই ইনস্ট্রাকশনের অ্যাড্রেসটি রাখা হবে। এক্ষেত্রে getch() ফাংশনের অ্যাড্রেসটি রাখা হবে।

০২. এরপর ফাংশনের প্যারামিটার হিসেবে যে ভেরিয়েবলগুলো নির্ধারণ করা হয়েছে সেগুলো স্ট্যাকে রাখা হবে।

০৩. সবশেষে ফাংশনের লোকাল ভেরিয়েবলের জন্য স্ট্যাকে জায়গা রাখা হবে।

চিত্রে দেখানো হয়েছে কিভাবে স্ট্যাকে ডাটাগুলো রাখা হবে।

একটি প্রোগ্রাম স্ট্যাকের জন্য মেমরিতে কতটুকু জায়গা রাখবে তা অপারেটিং সিস্টেম, কম্পাইলার ইত্যাদির ওপর নির্ভর করে। স্ট্যাকের জায়গা নির্ধারণের জন্য ভিন্ন ভিন্ন আর্কিটেকচার আছে। তাছাড়া কমপিউটারের মেমরির ওপরও নির্ভর করে কিভাবে এবং কতটুকু জায়গা স্ট্যাকের জন্য নির্ধারিত হবে। অনেক সময় বিভিন্ন প্রোগ্রাম ঠিকমতো চলে না বা হ্যাং করে। এটি মূলত স্ট্যাকের জন্য হয়ে থাকে। মেইন মেমরি যদি ছোট হয় তাহলে প্রয়োজনীয় ডাটা স্ট্যাকে স্টোর করা সম্ভব হয় না। অর্থাৎ স্ট্যাক ওভারফ্লো হয়ে যায়। আবার মেমরি বেশি থাকলেও এ ধরনের সমস্যা হতে পারে। যদি প্রোগ্রাম কোনো ইনফিনিটি লুপ চালায়, যেখানে ইনফিনিটিসংখ্যক ডাটা স্ট্যাকে স্টোর করার প্রয়োজন হয়, সেক্ষেত্রে স্ট্যাক ফুল হয়ে যায় অর্থাৎ স্ট্যাক ওভারফ্লো হয়ে যায়। তাই প্রোগ্রাম হ্যাং করে। ফাংশনের ব্যবহারে যেহেতু অনেক ক্ষেত্রে রিকার্শন ব্যবহার হয়, তাই স্ট্যাকের ব্যাপারটি খুব ভালোভাবে লক্ষ করতে হয়। রিকার্শনের লজিকে কোনো ভুল থাকলে স্ট্যাক ওভারফ্লো হওয়ার সম্ভাবনা থাকে, আর একবার ওভারফ্লো হলে প্রোগ্রাম হ্যাং করবে। ইউজার যদি টার্বো সি দিয়ে কোড ►

করেন তাহলে প্রোগ্রাম হ্যাং করার ব্যাপারে খুব সতর্ক থাকতে হবে। অন্যান্য ৩২ বিটের কম্পাইলার সাধারণত কোড কোনো ফাইলে সেভ করার পর তা রান করে। কিন্তু টার্বো সি কোড কোনো ফাইলে সেভ না করেই তা কম্পাইল এবং রান করে। সুতরাং কোড সেভ না থাকা অবস্থায় যদি প্রোগ্রাম একবার হ্যাং করে তাহলে টার্বো সি ক্লোজ করে আবার নতুন করে চালাতে হবে। অর্থাৎ কোডগুলো ডিলিট হয়ে যাবে এবং আবার নতুন করে শুরু থেকে কোড লিখতে হবে। তাই যেকোনো কম্পাইলার ব্যবহারের ক্ষেত্রে খুব সতর্ক থাকতে হবে এবং প্রতিবার রান করার আগে অবশ্যই একবার করে সেভ করে নিতে হবে। শুধু মেমরি কম থাকলেই এমন সমস্যা হতে পারে তা নয়, মেমরি বেশি থাকলেও হওয়ার সম্ভাবনা থাকে। রিকার্সনের কোড অথবা অসংখ্য ডাটার কোড রান করার সময় যদি অন্যান্য হাই রিসোর্সের প্রোগ্রাম চালানো থাকে তাহলেও স্ট্যাক ওভারফ্লো হওয়ার সম্ভাবনা থাকে। তাই রিকার্সিভ প্রোগ্রাম চালানোর সময় এ বিষয়গুলো ইউজারের খেয়াল রাখা উচিত।

আমরা জানি, স্ট্যাকের মধ্যে বিভিন্ন তথ্য রাখা হয় প্রোগ্রামের সুবিধার্থে। বিভিন্ন ফাংশনের প্যারামিটার, ডাটা, ইনস্ট্রাকশন ইত্যাদিকে কল ফ্রেম বলা হয়। ভিন্ন ভিন্ন ফাংশনের কল ফ্রেম ভিন্ন হয়ে থাকে এবং প্রোগ্রামে ব্যবহার সব ফাংশনের জন্যই কল ফ্রেম তৈরি হয়। সবশেষে যে ফাংশনকে কল করা হয়, তার কল ফ্রেম স্ট্যাকে সবার উপরে থাকে। আর যখন যে ফাংশনের কাজ শেষ হয়, তখন তার কল ফ্রেমও ডিলিট করা হয়। ফাংশনের যত প্রোপার্টি থাকে সবকিছু ডিলিট হয়ে যায়। এ কারণেই একই নামে একাধিক লোকাল ভেরিয়েবল ডিক্লেয়ার করা সম্ভব যদি তাদের স্কোপ ভিন্ন হয়ে থাকে। আর কোনো কল ফ্রেমের জন্য কোনো ফাংশনের কাজ শেষ হলে স্ট্যাকের জায়গাও খালি হয়ে যায়।

স্ট্যাটিক ভেরিয়েবল

স্কোপ অনুযায়ী ভেরিয়েবল কত ধরনের হতে পারে এবং তাদের মধ্যে পার্থক্য ও কাজ কী সে সম্পর্কে ধারণা দেয়া হলো। কিন্তু বিশেষ ধরনের একটি ভেরিয়েবল আছে যাকে স্ট্যাটিক ভেরিয়েবল বলা হয়। প্রোগ্রামে যত ধরনের ভেরিয়েবল ব্যবহার করা হয়, তাদের আরেক নাম অটোমেটিক ভেরিয়েবল। কেননা এই ভেরিয়েবলগুলোকে যখন ডিক্লেয়ার করা হয়, তখন তারা স্বয়ংক্রিয়ভাবে মেমরিতে জায়গা করে নেয় এবং ফাংশনের কলের সাথে স্ট্যাকেও জায়গা করে নেয়। আবার যখন ফাংশনের কাজ শেষ হয় তখন এই ভেরিয়েবলগুলো স্বয়ংক্রিয়ভাবে ডিঅ্যালোকেট হয়ে যায় বা ডিলিট হয়ে যায়। এদের জায়গা দখল বা ডিলিট হওয়ার জন্য ইউজারের বাড়তি কোনো ইনস্ট্রাকশন দেয়ার প্রয়োজন হয় না। আর অটোমেটিক ভেরিয়েবলের ফাংশনকে যতবার কল করা হবে, ততবার ভেরিয়েবলগুলো নতুন করে নিজেদের কপি করে নেবে। কিন্তু প্রোগ্রামে অনেক সময় এমন ভেরিয়েবলের প্রয়োজন হতে পারে যে, প্রথমবার কল করার সময় ভেরিয়েবলটির যে মান নির্ধারণ করে দেয়া হবে, পরের প্রত্যেকবার ফাংশনটিকে কল করার সময় ভেরিয়েবলটি আগের মানটি ব্যবহার করতে পারবে। অর্থাৎ ফাংশনকে নতুন করে কল করা হলেও ভেরিয়েবল আগের মতোই থাকবে এবং আগের মান অপরিবর্তিত থাকবে। এ ধরনের কাজ করার জন্য স্ট্যাটিক ভেরিয়েবল ব্যবহার করা হয়। নিচে স্ট্যাটিক ভেরিয়েবলের জন্য ছোট একটি প্রোগ্রাম উদাহরণ হিসেবে দেয়া হলো :

```
    void static_function()
    {
        static int count=1;
        printf("static variable is called %d
times",count);
    }
    void main()
    {
        static_function();
    static_function();
    static_function();
    getch();
    }
```

এখানে দেখা যাচ্ছে ইউজার ডিফাইন্ড ফাংশনে একটি স্ট্যাটিক ভেরিয়েবল ব্যবহার হয়েছে। এটি যদি অটোমেটিক ভেরিয়েবল হতো আউটপুটে ভেরিয়েবলটির মান তিনবারই ১ আসত। কারণ, মেইন ফাংশনে প্রতিবার ফাংশনটিকে কল করার সাথে সাথে স্ট্যাকে নতুন করে ভেরিয়েবল তৈরি হবে এবং তার জন্য নতুন করে মান নির্ধারণ করা হবে এবং প্রতিবার ফাংশন ক্লোজ করার সাথে সাথে ভেরিয়েবলটিও ডিলিট হয়ে যাবে। কিন্তু যেহেতু এটি স্ট্যাটিক ভেরিয়েবল, তাই প্রথমবার ভেরিয়েবল ডিক্লেয়ার করার পর ফাংশন ক্লোজ করা হলেও ভেরিয়েবলটি ডিলিট হবে না। তাই এই প্রোগ্রামে আউটপুট হিসেবে count-এর মান যথাক্রমে ১, ২, ৩ আসবে। যদিও ইউজার ডিফাইন্ড ফাংশনটিকে তিনবার কল করা হচ্ছে এবং প্রতিবারই count-এর মান ১ নির্ধারণ করে দেয়া হচ্ছে, তবুও ভেরিয়েবলটি শুধু একবারই ডিক্লেয়ার করা হবে এবং কখনো ডিলিট করা হবে না।

প্রোগ্রামে সব ধরনের ভেরিয়েবলেরই প্রয়োজন আছে। একেক সময় একেক ভেরিয়েবলের দরকার হয়। সাধারণত বড় বড় প্রোগ্রামের ক্ষেত্রে গ্লোবাল এবং স্ট্যাটিক ভেরিয়েবল ব্যবহার করার দরকার হয়। আর বিভিন্ন ভেরিয়েবলের ব্যবহার না জানলে, শুধু লোকাল ভেরিয়েবল ব্যবহার করলে প্রোগ্রাম করা অনেক কষ্টসাধ্য হয় এবং কখনো কখনো অসম্ভব হয়ে ওঠে। তাই সব ধরনের ভেরিয়েবলের ব্যবহার জানা অত্যন্ত জরুরি।

ফিডব্যাক : *wahid_cseaust@yahoo.com*