সহজ ভাষায় প্রোগ্রামিং সি/সি++

আহমদ ওয়াহিদ মাসুদ

বিভিন্ন ল্যাঙ্গুয়েজে যেকোনো ইনপুট সাধারণত স্ট্রিং হিসেবেই নেয়া হয়। তারপর তাকে নির্দিষ্ট ডাটা টাইপে কনভার্ট করা হয়। যেকোনো কিছু প্রিন্ট করতে হলে তা স্ট্রিং হিসেবে প্রিন্ট হয়। অর্থাৎ ইউজার প্রোগ্রামিংয়ে যা-ই করুক না কেনো, তা কোনো না কোনো সময় স্ট্রিংয়ে কনভার্ট হয়। এ লেখায় স্ট্রিংয়ের বিভিন্ন ব্যবহার যেমন ফাংশনের রিটার্ন টাইপ, ইনপুট/আউটপুট, ফাংশন ইত্যাদি নিয়ে আলোচনা করা হয়েছে।

ফাংশনের রিটার্ন টাইপ হিসেবে স্ট্রিং

যেকোনো ফাংশনের রিটার্ন ডাটা টাইপ হিসেবেও স্ট্রিংকে ব্যবহার করা সম্ভব। তবে এক্ষেত্রে একটি বিষয় লক্ষণীয়, কলার ফাংশন যাতে গারবেজ ভ্যালু পয়েন্ট না করে। যেমন :

```
char* funcX()
{
        char str[]=”A sample string”;
        return str;
}
```

এ ফাংশনটি কম্পাইল করার সময় কম্পাইলার কোনো এরর না দেখালেও কলার ফাংশন ঠিকমতো ডাটা পাবে না। কারণ ফাংশনটির কাজ করার সময় এর যে ডাটা স্ট্যাকে রাখা হবে, ফাংশনটির কাজ শেষে তা আর থাকবে না। তাই কলার ফাংশন এমন একটি অ্যাড্রেস পাবে, যা গারবেজ ডাটা পয়েন্ট করবে। ফাংশন থেকে স্ট্রিং রিটার্ন করার সময় এ বিষয়টি খেয়াল রাখতে হবে। এক্ষেত্রে কলার ফাংশনে প্রয়োজনীয় জায়গা নির্ধারণ করে তা কল্ড ফাংশনে পাঠানো যেতে পারে। যেমন :

```
funcX(char* str)
{
        strcpy(str,”String from funcX”)
        return st;
}
```

এভাবে ফাংশনটি ব্যবহার করা যায়। এতে ফাংশনটি যেহেতু মেইনের str-এর অ্যাড্রেস রিটার্ন করছে, তাই ptr কোনো গারবেজ ডাটাকে পয়েন্ট করবে না। অথবা কল্ড ফাংশনে ম্যালক ফাংশন ব্যবহার করেও কাজটি করা সম্ভব। যেমন :

```
char* funcX()
{
        char *ptr;
        ptr=malloc(32*sizeof(char));
        strcpy(ptr,”String from funcX”);
        return ptr;
}
```

এখানে ফাংশনে ম্যালক ফাংশন কল করার মাধ্যমে হিপ থেকে যে জায়গা নির্ধারণ করা হবে তা ptr পয়েন্ট করবে। ফাংশনটির কাজ শেষ হওয়ার পর মেইনের ptrও সেই একই জায়গা পয়েন্ট করবে।

এভাবে কলার ফাংশনে প্রয়োজনীয় জায়গা নির্ধারণ করে অথবা কল্ড ফাংশনে ম্যালক ফাংশন ব্যবহার করে হিপ থেকে জায়গা নিয়ে কল্ড ফাংশন থেকে স্ট্রিং রিটার্ন করলে কলার ফাংশন গারবেজ ডাটা পয়েন্ট করবে না। তাই ফাংশন থেকে স্ট্রিং রিটার্ন করার সময় গারবেজ ডাটার কথা সবসময় খেয়াল রাখতে হবে।

স্ট্রিংয়ে ডাবল কোটেশন বা ব্যাকস্ল্যাশ ব্যবহার

কোনো স্ট্রিংয়ে সরাসরি ডাবল কোটেশন (“”) বা ব্যাকস্ল্যাশ (\) ব্যবহার করা যায় না। স্ট্রিংয়ের মাধ্যমে ডাবল কোটেশন বা ব্যাকস্ল্যাশ ব্যবহার করতে হলে প্রতিটির আগে একটি বাড়তি ব্যাকস্ল্যাশ ব্যবহার করতে হবে। একে ক্যারেক্টার স্কিপ করা বলে। যেমন :

```
printf(“\”software\” which make the computer usable.”)
```

এখানে আউটপুট হবে : “software” which make the computer usable.

কারণ, প্রিন্ট ফাংশনের আর্গুমেন্ট (যে স্ট্রিংটি প্রিন্ট হবে) কতটুকু তা নির্ধারণ করা হয় দুটি ডাবল কোটেশনের মাধ্যমে। তাই ইউজার যদি শুধু ডাবল কোটেশন প্রিন্ট করতে চান তাহলে প্রোগ্রাম মনে করবে এই ডাবল কোটেশনটি প্রিন্ট করার জন্য নয় বরং এটি প্রিন্ট ফাংশনের আর্গুমেন্ট শেষ করার জন্য। তাই প্রিন্ট ফাংশনের আর্গুমেন্টে ডাবল কোটেশনের আগে ব্যাকস্ল্যাশ ব্যবহার করার মানে হলো প্রোগ্রামকে বলে দেয়া ওই ডাবল কোটেশনটি প্রিন্ট হবে।

ওপরের উদাহরণ থেকে বোঝা যায়, কোনো ক্যারেক্টার স্কিপ করতে হলে (অর্থাৎ ডাবল কোটেশনের মতো স্পেশাল ক্যারেক্টার প্রিন্ট করা) ব্যাকস্ল্যাশ ব্যবহার করতে হয়। কিন্তু ইউজার যদি একটি ব্যাকস্ল্যাশকেই প্রিন্ট করতে চান তাহলেও তা সাধারণ উপায়ে হবে না। অর্থাৎ এক্ষেত্রেও ক্যারেক্টার স্কিপ করতে হবে। যেমন :

```
printf (“a backslash \\ is printed”);
```

এখানে একটি ব্যাকস্ল্যাশের জায়গায় দুটি লেখা হয়েছে। প্রথমটি দিয়ে বোঝানো হচ্ছে পরের ক্যারেক্টারটি স্কিপ করতে হবে অর্থাৎ পরের ক্যারেক্টারের একমাত্র কাজ শুধু স্ক্রিনে প্রিন্ট হওয়া, অন্য কিছু নয়। আর পরের ক্যারেক্টারটি একটি স্কিপ করা স্পেশাল ক্যারেক্টার। এখন ওপরে বলা হলো ক্যারেক্টার স্কিপ করা মানে পরবর্তী ক্যারেক্টারটি শুধু প্রিন্ট করার কাজে ব্যবহার হবে, অন্য কাজে নয়। সুতরাং ইউজার ভাবতে পারেন যদি \a লেখা হয়, তাহলে শুধু a প্রিন্ট হবে। কারণ এখানে ব্যাকস্ল্যাশের পরবর্তী ক্যারেক্টারটি স্কিপ হচ্ছে অর্থাৎ শুধু প্রিন্ট হবে। কিন্তু এক্ষেত্রে আবার প্রত্যাশিত ফলাফল পাওয়া যাবে না। কারণ ক্যারেক্টার স্কিপিং শুধু স্পেশাল ক্যারেক্টারের জন্য প্রযোজ্য, সাধারণ ক্যারেক্টারের জন্য নয়।

স্ট্রিং ইনপুট/আউটপুট

নিচে স্ট্রিং ইনপুট ও আউটপুট পদ্ধতি নিয়ে আলোচনা করা হয়েছে। সি ল্যাঙ্গুয়েজে স্ট্রিং নিয়ে কাজ করার বিভিন্ন উপায় আছে। যেমন শুধু যদি স্ট্রিং ইনপুটের কথাই বলা হয়, ইউজার চাইলে অনেকভাবে স্ট্রিং ইনপুট নিতে পারেন। একেকটি নিয়মের কাজ করার পদ্ধতি একেক ধরনের। এদের কিছু সুবিধা ও অসুবিধা আছে। আবার ইউজারের কাজের সুবিধার্থে ইনপুট/আউটপুটের জন্য সি-তে কিছু বিল্টইন ফাংশনও আছে।

স্ট্রিং ইনপুট ও আউটপুট নেয়ার জন্য সাধারণত দুই ধরনের পদ্ধতি ব্যবহার করা হয়। একটি ফরম্যাট স্পেসিফায়ার ব্যবহার করে ইনপুট নেয়া, আরেকটি ব্যবহার না করে। প্রথমে কীভাবে ফরম্যাট স্পেসিফায়ার ব্যবহার করে ইনপুট নেয়া হয় তা দেখানো হয়েছে।

```
char ch[50]=“”;
scanf(“%s”,&ch);
```

ইউজার ইচ্ছে করলে ফরম্যাট স্পেসিফায়ার ছাড়াও ইনপুট নিতে পারেন। যেমন :

```
char ch[50]=“”;
gets(ch);
```

এখানে gets() নামে যে ফাংশনটি ব্যবহার করা হয়েছে, তা শুধু স্ট্রিং ইনপুটের জন্যই ব্যবহার করা হয়। সাধারণত দুটি পদ্ধতিই ঠিকমতো কাজ করে। কিন্তু প্রোগ্রাম যদি অনেক বড় হয় এবং বিশেষত একটি লুপের মাঝে যদি স্ট্রিং ইনপুট নিয়ে লুপটিকে বারবার চালানো হয় তাহলে gets() ফাংশন ব্যবহার করলে অনেক সময় ভুল ইনপুট যেতে পারে। এর কারণ, একটি স্ট্রিংয়ের ইনপুট নেয়া তখনই শেষ হয়, যখন এন্টার বাটনটি প্রেস করা হয়। gets() ফাংশন এভাবেই বানানো হয়েছে যে যখন এন্টার প্রেস করা হবে তখন তা ইনপুট নেয়া বন্ধ করে দেবে এবং সবশেষে একটি নাল ক্যারেক্টার বসিয়ে দেবে। আসলে স্ট্রিং ইনপুট নেয়ার পদ্ধতি হলো, ইউজার যখন একটি স্ট্রিং টাইপ করেন তখন প্রতিটি বাটন প্রেস করার সাথে সাথে সেই সিম্বলটি মেমরির স্ট্যাকে বসে যায়। পরে তা স্ট্রিংয়ের জন্য ডিক্লেয়ার করা অ্যারেতে পাঠানো হয়। এখন ইউজার একটি লাইন টাইপ করে এন্টার চাপলেন, ফলে যা যা হবে তা হলো, প্রথমে ইনপুট নেয়া বন্ধ হবে, তারপর এন্টারের সিম্বলের (এন্টার বাটনের সিম্বল \n এবং আস্কি মান ১০) আগ পর্যন্ত অ্যারেতে বসবে। এরপর লুপের অন্যান্য কাজ সম্পন্ন হয়ে লুপ শেষ হবে। পরে লুপে যখন প্রোগ্রাম আসবে তখন আবার gets() ফাংশন ইনপুট নেয়ার জন্য কাজ করবে। কিন্তু এর আগের \n এখনও স্ট্যাকে রয়ে গেছে, তাই প্রোগ্রাম মনে করবে আরেকবার এন্টার প্রেস করা হয়েছে। এভাবে লুপের ভেতরে স্ট্রিং ইনপুট ▸

নেয়ার সময় সমস্যা তৈরি হতে পারে। এর জন্য ফরম্যাট scanf() ফাংশন ব্যবহার করা হলে এ ধরনের সমস্যা হওয়ার সম্ভাবনা থাকে না।

সবার জানা আছে, টাইপ করার সময় এন্টার চাপলে নতুন লাইনে কার্সর চলে যায়। এটি একেবারেই সাধারণ একটি অপারেশন এবং এর জন্য কোনো ধরনের জটিলতার মুখোমুখি হতে হয় না। কিন্তু স্ট্রিং নিয়ে কাজ করার সময় প্রোগ্রামারকে এ বিষয়ে সতর্ক থাকতে হয়। কারণ ইউজার যখন এন্টার বাটন চাপেন, তখন আসলে একই সাথে দুটি কাজ সম্পন্ন হয়। প্রথমে টাইপিংয়ের কার্সরটি যেখানে আছে ঠিক তার নিচের স্থানে চলে যায়, যার নাম নিউ লাইন এবং এর সিম্বল \n, আর পরের কাজ হলো কার্সর যেখানে যে লাইনে আছে সেখান থাকে সে লাইনের শুরুতে চলে যাওয়া, যার নাম ক্যারিয়েজ রিটার্ন এবং এর সিম্বল হলো \r। তাহলে দেখা যাচ্ছে এন্টার চাপলে আসলে \rn সিম্বল কাজ করে।

এখন ধরা যাক প্রোগ্রামার একটি স্ট্রিংয়ের ওপর কন্ডিশন নিয়ে প্রোগ্রাম লিখছেন। কন্ডিশনে যদি এরকম থাকে যে, ইউজার টাইপ করতে করতে এন্টার দিলে কোনো বিশেষ কাজ হবে, তাহলে এ ক্যারিয়েজ রিটার্ন সিম্বলের দিকে খেয়াল রাখতে হবে। কারণ সেখানে কন্ডিশন হিসেবে শুধু \n চেক করলেই হবে না, \r-কেও চেক করতে হবে। তবে কম্পাইলার ভিত্তিতে নিউ লাইন ও ক্যারিয়েজ রিটার্নকে একসাথে বা আলাদাভাবে লেখা যায়।

স্ট্রিং ইনপুট/আউটপুটের কয়েকটি উদাহরণ :

```
char *ch1=”using pointer”;
char ch2[]=”using array “;
printf(“%s \n %s”,ch1,ch2);
```

ইউজার যদি ওপরের দুটি উপায়ে ইনপুট নেন, তাহলে উভয় ক্ষেত্রেই আউটপুট এরকম আসবে :

```
using pointer
using array
```

অর্থাৎ দুটি পদ্ধতিই সঠিক এবং প্রোগ্রাম কোনো এরর দেখাবে না। এখানে আসলে প্রথমে পয়েন্টার এবং পরে সাধারণ অ্যারে ব্যবহার করে স্ট্রিং ইনপুট নেয়া হয়েছে। পরে দুটিকেই ফরম্যাট স্পেসিফায়ার দিয়ে প্রিন্ট করা হয়েছে। অর্থাৎ প্রিন্ট করার উপায় একই হলেও ইনপুট নেয়ার উপায় ভিন্ন। আর দ্বিতীয় ইনপুটে ক্যারেক্টার অ্যারে ডিক্লেয়ার করা হলেও অ্যারের এলিমেন্ট সংখ্যা হয়নি। এক্ষেত্রে প্রোগ্রাম নিজে থেকেই এলিমেন্ট ডিক্লেয়ার করে নেবে। যতগুলো ক্যারেক্টার নিয়ে ইনপুট স্ট্রিংটি গঠিত হবে, অ্যারের এলিমেন্ট সংখ্যাও ততগুলো হবে।

একইভাবে কোনো স্ট্রিং অ্যারের বিভিন্ন এলিমেন্টের ডাটা লুপের মাধ্যমেও প্রিন্ট করা সম্ভব। যেমন :

```
char *ch[6]={“a”,”b”,”c”,”d”,”e”,”f”};
for(int n=0;n<6;n++)
        printf(“%s\n”,ch[n]);
```

এক্ষেত্রে সবগুলো স্ট্রিং পরপর প্রিন্ট হবে। যদিও এখানে ইনপুটে একটি মাত্র ক্যারেক্টার নেয়া হয়েছে, কিন্তু লক্ষ করলে দেখা যাবে, ক্যারেক্টারগুলো সব ডাবল কোটেশনের মাঝে রাখা হয়েছে। অর্থাৎ প্রোগ্রাম সেগুলোকে স্ট্রিং হিসেবে বিবেচনা করবে, তাতে একটি করে ক্যারেক্টারই থাকুক আর কিছু নাই থাকুক।

স্ট্রিং প্রিন্ট করার আরেকটি পদ্ধতি :

```
char ctrl_char[]=”%s\n”;
char func_argu[]=”this is a string”;
printf(ctrl_argu,func_argu);
printf(“%s\n”,func_argu);
printf(func_argu);
```

এখানে প্রিন্ট করার জন্য তিনটি লাইন লেখা হয়েছে এবং তিনটি লাইনের জন্যই this is a string এ স্ট্রিংটি প্রিন্ট হবে। এটি আবার printf() ফাংশনের ভিন্ন ধরনের ব্যবহার। মোট কথা যেভাবেই প্রিন্ট করা হোক না কেনো, প্রোগ্রামে যদি গারবেজ ভ্যালু নিয়ে সমস্যা না হয় আর নিয়ম যদি ঠিক থাকে, তাহলে অনেকভাবেই স্ট্রিং প্রিন্ট করা সম্ভব। এখান থেকে ইউজার তার পছন্দমতো যেকোনো পদ্ধতিতে স্ট্রিং প্রিন্ট করতে পারবেন। তবে মাঝেমধ্যে কোনো কোনো বিশেষ ক্ষেত্রে পদ্ধতি ব্যবহারে সমস্যা দেখা দিতে পারে। সেক্ষেত্রে অন্য পদ্ধতি ব্যবহার করাই হবে সমস্যা সমাধানের সহজ উপায়।

এবার স্ট্রিং আউটপুটের জন্য puts() Ges putchar() ফাংশনের ব্যবহার দেখানো হয়েছে। puts() হলো একটি লাইব্রেরি ফাংশন, যার প্রটোটাইপ stdio.h ফাইলে লেখা আছে। প্রোগ্রামে puts() ফাংশনের মাঝে যে স্ট্রিং রাখা হবে তা প্রোগ্রাম প্রিন্ট করে দেখাবে। এ স্ট্রিংটি সরাসরি কোনো ভ্যালুও হতে পারে, আবার কোনো ভেরিয়েবলও হতে পারে। শুধু একটি বিষয়ই এখানে লক্ষণীয়, স্ট্রিংটি যেনো একটি নাল ক্যারেক্টার দিয়ে টারমিনেটেড হয়। নাল ক্যারেক্টারের কথা এজন্যই বলা হলো যাতে puts() ফাংশনটি বুঝতে পারে, যে স্ট্রিংটি তাকে দেয়া হলো তা কোথায় গিয়ে শেষ হবে। তা না হলে মূল স্ট্রিং প্রিন্ট করার পর গারবেজ ভ্যালু প্রিন্ট হতে থাকবে। আরেকটি বিষয় জেনে রাখা ভালো, এ ফাংশনটি যদি ঠিকভাবে কাজ করে তাহলে যেকোনো ধনাত্মক সংখ্যা রিটার্ন করবে, অন্যথায় EOF (End Of File) রিটার্ন করবে। এবং এটিও মূলত নাল ক্যারেক্টার না থাকার জন্যই হয়ে থাকে।

প্রোগ্রামে প্রয়োজনে putchar()-কেও স্ট্রিংয়ের আউটপুটের জন্য ব্যবহার করা যেতে পারে। এটিও একটি বিল্টইন ফাংশন এবং এর প্রটোটাইপ stdio.h হেডার ফাইলে লেখা আছে। আগের ফাংশনের সাথে এর মূল পার্থক্য হলো, আগেরটি সরাসরি স্ট্রিং প্রিন্ট করবে আর এটি একটি একটি করে ক্যারেক্টার প্রিন্ট করবে। যেমন :

```
int ch=’a’;
putchar(ch);
```

এখানে প্রোগ্রাম a প্রিন্ট করবে। এখন আমরা জানি একটি স্ট্রিং হচ্ছে অনেকগুলো ক্যারেক্টারের সমষ্টি। সুতরাং ইউজার যদি অ্যারে দিয়ে স্ট্রিংয়ের ইনপুট নিয়ে থাকেন, তাহলে তার আউটপুট দেয়ার জন্য একটি লুপের মাধ্যমে ওই অ্যারেটি এই ফাংশন দিয়ে প্রিন্ট করতে পারবেন। কারণ অ্যারেটির এলিমেন্টগুলো যেহেতু একটি করে ক্যারেক্টার ধারণ করবে, তাই তা putchar() সহজেই প্রিন্ট করতে পারবে। তবে এক্ষেত্রে লক্ষণীয়, ফাংশনটি নাল ক্যারেক্টার পেলেও তা প্রিন্ট করার চেষ্টা করবে। সুতরাং ইউজার যদি লুপের মাধ্যমে এ ফাংশনটি ব্যবহার করেন, তাহলে লুপের কন্ডিশন এমন রাখতে হবে যেনো নাল ক্যারেক্টার আসলে লুপটি ব্রেক হয়ে যায়।

স্ট্রিং প্রোগ্রামিংয়ের খুব গুরুত্বপূর্ণ একটি অংশ। এর সুবিধাও যেমন অনেক বেশি, তেমনি এর ব্যবহার করার উপায়ও অনেক। সব পদ্ধতিরই কিছু সুবিধা-অসুবিধা আছে। তাই ইউজারের প্রোগ্রাম লেখার সময় সতর্কতার সাথে স্ট্রিংয়ের উপায়গুলো ব্যবহার করা উচিত কজ

ফিডব্যাক : wahid_cseaust@yahoo.com