সহজ ভাষায় প্রোগ্রামিং সি/সি++

আহমদ ওয়াহিদ মাসুদ

গত পর্বে স্ট্রিং নিয়ে আলোচনা করা হয়েছে। যেকোনো প্রোগ্রামিং ল্যাঙ্গুয়েজে স্ট্রিংয়ের গুরুত্ব অপরিসীম। আধুনিক ল্যাঙ্গুয়েজে যেকোনো ইনপুট সাধারণত স্ট্রিং হিসেবেই নেয়া হয়। এরপর একে নির্দিষ্ট ডাটা টাইপে কনভার্ট করা হয়। যেকোনো কিছু প্রিন্ট করতে হলে তা স্ট্রিং হিসেবে প্রিন্ট হয়। তাই দক্ষ প্রোগ্রামিংয়ের জন্য স্ট্রিংয়ের ওপর ধারণা থাকা আবশ্যক। এ লেখায় স্ট্রিংয়ের বিভিন্ন ব্যবহার যেমন ভেরিয়েবল, অ্যারে বা ফাংশনের প্যারামিটার ইত্যাদি নিয়ে আলোচনা করা হয়েছে।

char* ও string ভেরিয়েবল

প্রোগ্রামে প্রয়োজনে char*-কে স্ট্রিং ভেরিয়েবল হিসেবে ব্যবহার করা সম্ভব। এর আগে char* কী, তা জেনে নেয়া যাক। char মানে ক্যারেক্টার টাইপের ডাটা তা সবাই জানে। আর অ্যাস্টেরিক সাইন (*) পয়েন্টারের জন্য ব্যবহার করা হয়, তাও সবার জানা। আর পয়েন্টার হলো একটি বিশেষ ভেরিয়েবল, কিন্তু এটি কোনো সাধারণ মান ধারণ করতে পারে না। এটি শুধু অপর ভেরিয়েবলের অ্যাড্রেস ধারণ করতে পারে। একটি প্রোগ্রামের জন্য অ্যাড্রেসটিই মূল বিষয়। প্রতিটি ভেরিয়েবলেরই একটি করে অ্যাড্রেস থাকে। প্রোগ্রাম ওই ভেরিয়েবলগুলোকে তাদের নামে নয় বরং তাদের অ্যাড্রেস দিয়ে চেনে। ওই অ্যাড্রেসে কোনো কিছু পরিবর্তন করলে সংশ্লিষ্ট ভেরিয়েবলেও সেই পরিবর্তন দেখা যাবে। অর্থাৎ কোনো ভেরিয়েবলের যে অ্যাড্রেস আছে, সে অ্যাড্রেসের মানকে মুছে দেয়া হলে ভেরিয়েবলের মানও ডিলিট হয়ে যাবে। আবার কোনো অ্যাড্রেসে নতুন কোনো মান অ্যাসাইন করা হলে অ্যাড্রেসের যে ভেরিয়েবল আছে, তার মানও পরিবর্তন হয়ে যাবে।

এখন কোনো ডাটা টাইপের সাথে যখন অ্যাস্টেরিক সাইন ব্যবহার করা হয়, তখন সেটি কোনো ভেরিয়েবল হিসেবে ডিক্লেয়ার করা হয় না, আবার কোনো পয়েন্টার হিসেবেও ডিক্লেয়ার করা হয় না। আসলে এভাবে প্রোগ্রামকে বলে দেয়া হয় যে ডিক্লারেশনের পর যে ডাটা থাকবে তাকে ওই ডাটা টাইপের পয়েন্টার দিয়ে পয়েন্ট করতে হবে। একটি ছোট উদাহরণ :

```
char* ch = ‘No smoking’;
```

এখানে char* দিয়ে বুঝানো হচ্ছে, এরপরে যে ডাটা থাকবে তাকে একটি ক্যারেক্টার টাইপের পয়েন্টার পয়েন্ট করবে। আর সেই পয়েন্টারের নাম হবে ch। আর এটি মেমরিতে কীভাবে থাকবে সেটিও সহজে বোঝা যায়। মেমরিতে ‘No smoking’ একটি স্ট্রিং হিসেবে থাকবে, যার কোনো নাম থাকবে না। আর ch নামের একটি পয়েন্টার ওই স্ট্রিংটির প্রথম ক্যারেকটার N-কে পয়েন্ট করে থাকবে। তাই প্রোগ্রাম যখন ওই পয়েন্টার দিয়ে স্ট্রিংটিকে প্রিন্ট করতে চাইবে, তখন একটি একটি করে ক্যারেক্টার প্রিন্ট হবে, আর ch পরের ক্যারেক্টারকে পয়েন্ট করবে। তবে লক্ষণীয়, এ ক্ষেত্রে অবশ্যই একটি নাল ক্যারেক্টারের প্রয়োজন। তা না হলে প্রিন্ট করা শেষ হবে না এবং স্ট্রিং প্রিন্ট করার পর গারবেজ ভ্যালু প্রিন্ট করা শুরু হবে। তাই স্ট্রিংয়ের শেষে একটি নাল ক্যারেক্টার বসানো প্রয়োজন। যদি ওপরের নিয়মে স্ট্রিং ডিক্লেয়ার করা হয়, তাহলে নাল বসানোর এ কাজটি প্রোগ্রাম নিজে থেকেই করে নেয়। আরেকটি বিষয়, ‘No smoking’ স্ট্রিংটির জন্য ব্যবহার হওয়া জায়গাকে আননেমড স্পেস বলে, কারণ মেমরিতে এ ব্যবহার হওয়া জায়গার জন্য কোনো নাম ঠিক করা হয়নি। বরং যেই পয়েন্টার দিয়ে স্ট্রিংটিকে পয়েন্ট করা হচ্ছে তার নাম ch হিসেবে ঠিক করা হয়েছে। এ ধরনের আরও দুটি উদাহরণ :

```
char* house_name=‘Basundhara’;
char* road_no=‘10, high level road’;
```

এবার char*-কে কীভাবে স্ট্রিং ভেরিয়েবল হিসেবে ব্যবহার করা যায়, তা ব্যাখ্যা করা হবে। আগেই বলা হয়েছে, এখানে ch দিয়ে ওই স্ট্রিংটিকে পয়েন্ট করা হচ্ছে না, বরং ওই স্ট্রিংয়ের প্রথমেই যে ক্যারেক্টারটি আছে, তাকে পয়েন্ট করা হচ্ছে। তাহলে ইউজার যদি ওই স্ট্রিং প্রিন্ট করার জন্য এভাবে স্টেটমেন্ট লেখেন :

```
printf(“%s”,*ch);
```

তাহলে এখানে গারবেজ ভ্যালু দেখাবে। কারণ, ফাংশনের ভেতরে ভেরিয়েবল হিসেবে ব্যবহার হয়েছে একটি ক্যারেক্টার টাইপের পয়েন্টার, কিন্তু তা প্রিন্ট করার জন্য স্ট্রিংয়ের ফরম্যাট স্পেসিফায়ার ব্যবহার করা হয়েছে। তাই কীভাবে ch ব্যবহার করে স্ট্রিংটি প্রিন্ট করা যায়, তা জানার জন্য আগে জানতে হবে কীভাবে ch কাজ করছে। আগে বলা হয়েছে, ch শুধু ওই স্ট্রিংয়ের প্রথম ক্যারেক্টার N-কে পয়েন্ট করছে। তাই এখানে ch-কে ক্যারেক্টার হিসেবে প্রিন্ট করলে গারবেজ ভ্যালু প্রিন্ট না হয়ে N প্রিন্ট হবে। আবার ch-কে ইনক্রিমেন্ট করলে তা পরের ক্যারেক্টারকে পয়েন্ট করবে। সুতরাং এখানে একটি একটি করে ক্যারেক্টার প্রিন্ট করতে হবে। যেমন :

```
printf(“%c”,*ch);
*ch++;
```

এবং এ দু'টি স্টেটমেন্টকে অবশ্যই একটি লুপের মাধ্যমে চালাতে হবে। আর লুপের কন্ডিশনটি স্ট্রিংটির শেষ ক্যারেক্টার ব্যবহার করে দিতে হবে। এখানে স্ট্রিংটির সর্বশেষ ক্যারেক্টার হলো নাল ক্যারেক্টার। তাই কন্ডিশনটি এমন হবে যে যতক্ষণ না পর্যন্ত ch-এর মান নাল না হচ্ছে ততক্ষণ লুপ চলবে। সুতরাং সম্পূর্ণ কোডটি হবে :

```
for(;*ch!=null;)
{
        printf(“%c”,*ch);
*ch++;
}
```

স্ট্রিং হিসেবে char* ও char[]-এর পার্থক্য

স্ট্রিংটিকে অ্যারে নোটেশনে রাখলে যতটুকু জায়গা নিত, পয়েন্টার নোটেশনে রাখলে তার চেয়ে আরও কমপক্ষে ২ বাইট বেশি জায়গা নেবে। তাই জায়গার কথা চিন্তা করলে পয়েন্টার নোটেশন ব্যবহার করা উচিত। কিন্তু স্ট্রিং ব্যবহারের ক্ষেত্রে জায়গাই সব নয়। আরও অনেক ব্যাপার থাকে। তাই কোনো স্ট্রিংয়ের জন্য পয়েন্টার নোটেশন না অ্যারে নোটেশন ভালো, তা শুধু জায়গার ওপর নির্ভর করে না, বরং পরে সেই স্ট্রিংটি কীভাবে ব্যবহার করা হবে তার ওপরও নির্ভর করে।

পয়েন্টারের জন্য কোনো জায়গা নির্ধারিত হয় না

প্রোগ্রামে char ch[80] লেখার অর্থ হলো প্রোগ্রাম চলার সময় ch-এর জন্য ৮০ বাইট জায়গা নির্ধারিত হবে। কিন্তু char *ch লেখার মানে হলো শুধু ch-এর জন্য ২ বাইট জায়গা নির্ধারিত হবে, তবে এটি সিস্টেম ও কম্পাইলারের ওপর নির্ভর করে। তাই অনেক সময় এটি ২ বাইটের জায়গায় ৪ বাইটও হতে পারে। তাই প্রোগ্রামে নিচের মতো স্টেটমেন্ট লিখলে কোনো এরর দেখাবে না, কিন্তু অপ্রত্যাশিত ফল দেখাবে।

```
char* str;
strcpy(str, “hello”);
printf(“%s”,str);
```

এখানে strcpy() ফাংশনের কাজ হলো এক স্ট্রিংয়ের ডাটা আরেক স্ট্রিংয়ে কপি করা। ফাংশনটির দুটি প্যারামিটার থাকে। প্রথমে ডেস্টিনেশন ও পরে সোর্স লিখতে হয়। এখানে ইউজার চাইলে স্ট্রিংয়ের নাম না দিয়ে সরাসরি স্ট্রিংটি দিতে পারেন। ওপরের উদাহরণে তাই করা হয়েছে। দ্বিতীয় প্যারামিটারে সরাসরি একটি স্ট্রিং দিয়ে দেয়া হয়েছে।

এখানে strcpy() ফাংশনের মাঝে hello স্ট্রিংকে str-এ কপি করতে বলা হয়েছে। এ ক্ষেত্রে কম্পাইলার কোনো এরর না দেখালেও ফল অপ্রত্যাশিত দেখাবে। কারণ str শুধু পয়েন্টার ডাটা অর্থাৎ অ্যাড্রেস রাখবে, কোনো স্ট্রিং নয়। তাই স্ট্রিং ভেরিয়েবল হিসেবে *char ▸

ব্যবহার করার সময় খেয়াল রাখতে হবে যে *char-এ স্ট্রিং রাখার জন্য কোনো জায়গা নির্ধারণ করা হয় না। তবে এ ক্ষেত্রে malloc() ব্যবহার করে প্রয়োজনীয় জায়গা নির্ধারণ করা যায়।

```
int main ()
{
        char *str;
        str=malloc(20*sizeof(char));
        strcpy(str,”hello”);
        printf(“%s”,str);
        free(str);
        getch();
        return 0;
}
```

এখানে malloc() ফাংশন হিপ থেকে যে ৮০ বাইট জায়গা অ্যালোকেট করবে তাকে str পয়েন্ট করবে। এরপর str-এর জন্য যে স্ট্রিং ডাটা নির্ধারণ করা হবে, তা str-এর পয়েন্টেড মেমরিতে রাখা হবে। malloc() ফাংশন নিয়ে পরে আলোচনা করা হবে।

স্ট্রিং ভেরিয়েবল ও কন্ডিশনাল এক্সপ্রেশন

if, for, while ইত্যাদি কন্ডিশনাল স্টেটমেন্টে ব্যবহৃত এক্সপ্রেশনের অপারেন্ড হিসেবেও স্ট্রিং ব্যবহার করা যায়। যেমন :

if স্টেটমেন্ট : আমরা জানি সি-তে if-এর সাথে ব্যবহার হওয়া এক্সপ্রেশনের মান 0 না হলে if সংশ্লিষ্ট স্টেটমেন্টে কাজ করে। আবার ওপরে বলা হয়েছে প্রতিটি স্ট্রিংয়ের শেষে একটি নাল ক্যারেক্টার থাকে, যার আস্কি ভ্যালু 0। তাই কোনো *ch-কেও if-এর কন্ডিশনাল এক্সপ্রেশন হিসেবে ব্যবহার করা যায়। যেমন :

```
char* ch= “hello”;
if(*ch==’\0’)
        return 0;
```

এখানে if-এর কন্ডিশন হিসেবে বলা হয়েছে, যখন পয়েন্টার কোনো নালকে পয়েন্ট করবে, তখন প্রোগ্রাম রিটার্ন করবে অর্থাৎ বন্ধ হয়ে যাবে।

for স্টেটমেন্ট : ওপরের মতো করে for লুপেও কন্ডিশন হিসেবে স্ট্রিং পয়েন্টার ব্যবহার করা যায়। যেমন :

```
for(;*ch!=\0;*ch++)
        printf(“%c”,*ch);
```

এখানে for লুপের মাধ্যমে সম্পূর্ণ স্ট্রিংটি প্রিন্ট করা হয়েছে।

while স্টেটমেন্ট :

```
while(*ch!=null)
        putc(*ch++);
```

এখানেও একইভাবে কন্ডিশনাল স্টেটমেন্টে স্ট্রিং পয়েন্টার ব্যবহার হয়েছে। আর প্রতিটি ক্যারেক্টার প্রিন্ট করার জন্য putc() নামে একটি ফাংশন ব্যবহার করা হয়েছে। এর কাজ প্যারামিটার হিসেবে যে পয়েন্টার বা ভেরিয়েবল থাকবে, তার ভ্যালু বা ক্যারেক্টার প্রিন্ট করবে। এখানে প্যারামিটার হিসেবে সরাসরি ক্যারেক্টারও ব্যবহার করা যায়। আর প্রিন্ট করার পর পয়েন্টার ইনক্রিমেন্ট করা হয়েছে। ++ অপারেটরকে ইনক্রিমেন্ট বলা হয়। কোনো ভেরিয়েবলের সাথে ইনক্রিমেন্ট ব্যবহার করলে তার মান ১ বেড়ে যায়। তবে ইনক্রিমেন্ট দুই ধরনের। প্রি ইনক্রিমেন্ট ও পোস্ট ইনক্রিমেন্ট। যেমন : a=1; printf(“%d”,++a); printf(“%d”,a++); এখানে প্রথমে প্রি ইনক্রিমেন্ট ও পরে পোস্ট ইনক্রিমেন্টের উদাহরণ দেয়া হয়েছে। প্রথমে a-এর মান বেড়ে ২ হবে, তারপর তা প্রিন্ট হবে। পরেরবার a-এর ভ্যালু প্রিন্ট হবে, তারপর তার ভ্যালু বেড়ে ৩ হবে। অর্থাৎ এ তিনটি স্টেটমেন্টের আউটপুট হবে ২২ (প্রথমে ২, তারপর আবার ২ প্রিন্ট করবে)। ডিক্রিমেন্ট অপারেটরের (--) ক্ষেত্রেও একই ধরনের নিয়ম প্রযোজ্য। এখানে ++, -- হলো ইউনারি অপারেটর ও এদের অ্যাসোসিয়েটিভিটি হলো ডান থেকে বাম দিকে। আর এদের সবার প্রিসিডেন্স সমান। তাই কোনো এক্সপ্রেশনে পয়েন্টার ভেরিয়েবলের সাথে যদি দুটো অপারেটর ব্যবহার করা হয়, তাহলে এক্সপ্রেশনের মান বের করার জন্য সাধারণত অ্যাসোসিয়েটিভিটি ব্যবহার করা হয়।

স্ট্রিংয়ের অ্যারে

স্ট্রিংয়ের অ্যারে কী, তা বোঝার জন্য অ্যারে সম্পর্কে ভালো ধারণা থাকতে হবে। অ্যারে হলো অনেকগুলো ভেরিয়েবল একসাথে ডিক্লেয়ার করার একটি পদ্ধতি। ধরুন, কোনো প্রোগ্রামে একসাথে পাঁচটি ভেরিয়েবল ডিক্লেয়ার করার প্রয়োজন হলো। তাহলে ইউজার সাধারণ নিয়মে পাঁচটি ভেরিয়েবল ডিক্লেয়ার করতে পারেন। এজন্য পাঁচটি স্টেটমেন্ট লেখার প্রয়োজন হবে। কিন্তু অ্যারে ব্যবহার করে পাঁচটি ভেরিয়েবল একসাথে অর্থাৎ একটি স্টেটমেন্ট দিয়েই ডিক্লেয়ার করা সম্ভব। মাত্র পাঁচটি ভেরিয়েবলের ক্ষেত্রে হয়তো এটি তেমন গুরুত্বপূর্ণ বিষয় নয়, কিন্তু অনেক বড় বড় প্রোগ্রামে একসাথে যখন ১০০ বা ১০০০ ভেরিয়েবল ডিক্লেয়ার করার প্রয়োজন হয়, তখন অ্যারে ব্যবহার করলে কোডিং অনেক সহজ হয়ে যায়। অ্যারে ডিক্লেয়ার করার সিনটেক্স : data_type array_name[array_size]। এখানে ডাটা টাইপ হলো অ্যারের ডাটা টাইপ, অর্থাৎ কোন ধরনের ভেরিয়েবলের অ্যারে ডিক্লেয়ার করা হচ্ছে তা বলে দেয়া। অ্যারের নাম হলো যেকোনো ভেরিয়েবলের নাম। অর্থাৎ ভেরিয়েবলের নামের নিয়মানুসারে অ্যারের নাম করতে হবে। কতগুলো ভেরিয়েবল নিয়ে অ্যারে ডিক্লেয়ার করা হচ্ছে সেটা বলতে হবে, এটি হলো অ্যারে সাইজ। শুধু অ্যারের নামের ভেরিয়েবল হিসেবে ব্যবহার করা যায় না। অর্থাৎ int a[5] অ্যারে ডিক্লেয়ার করা হলে শুধু a নামে কোনো ভেরিয়েবল থাকবে না। তবে ফাংশনের প্যারামিটার হিসেবে কোনো অ্যারে পাঠাতে হলে শুধু অ্যারের নাম ব্যবহার করতে হয়। তাও আবার লোকাল প্যারামিটারের ক্ষেত্রে এটি প্রযোজ্য।

সাধারণ ভেরিয়েবলের মতো স্ট্রিংয়েরও অ্যারে ডিক্লেয়ার করা যায়। যেমন :

স্ট্রিং ভেরিয়েবল : char *str;
স্ট্রিং ভেরিয়েবলের অ্যারে : char *str[5];

প্রথমটি একটি সাধারণ স্ট্রিং ভেরিয়েবল। কিন্তু পরেরটি একটি স্ট্রিং অ্যারে, যেখানে পাঁচটি এলিমেন্ট ডিক্লেয়ার করা হয়েছে। ইউজার ইচ্ছে করলে সাধারণ অ্যারের মতো এখানেও প্রতিটি এলিমেন্টের মান অ্যাসাইন করে দিতে পারেন। যেমন :

str[0]=”one”;
str[1]=”two”; ইত্যাদি।

তবে স্ট্রিং অ্যারের ক্ষেত্রেও প্রতিটি এলিমেন্টের মান ডিক্লেয়ার করার সময়ই অ্যাসাইন করা যায়। যেমন :

```
char* river[3]={“padma”,”meghna”,
”jamuna”};
```

এখানে লক্ষণীয়, কোনো স্ট্রিংয়ের অ্যারে যখন ডিক্লেয়ার করা হয়, তখন তা মেমরিতে পর পর সাজানো থাকে। প্রথম এলিমেন্টের পর একটি নাল ক্যারেক্টার থাকে, তারপর পরের এলিমেন্টের মান থাকে। এভাবে পরের সব এলিমেন্টের মান মেমরিতে সাজানো থাকে।

ফাংশনের প্যারামিটার হিসেবে স্ট্রিং ভেরিয়েবল

স্ট্রিং ভেরিয়েবলকে প্রয়োজনে ফাংশনের প্যারামিটার হিসেবেও ব্যবহার করা যায়। যেমন :

```
char* ch=”hello”;
fun1(ch);
fun1(char* ch)
{
        for(;ch!=null;ch++)
                putc(ch);
}
```

এখানে fun1(ch) নামে একটি ফাংশন তৈরি করা হয়েছে এবং ফাংশনের বডিতে শুধু স্ট্রিংটি প্রিন্ট করার কমান্ড লেখা হয়েছে। এখন মেইন ফাংশন থেকে এ ফাংশনটি কল করলে এবং যেকোনো স্ট্রিং পয়েন্টার তার প্যারামিটার হিসেবে দিলে ওই স্ট্রিংটি প্রিন্ট হবে।

স্ট্রিং আধুনিক প্রোগ্রামিং ল্যাঙ্গুয়েজের একটি বিশেষ বৈশিষ্ট্য। স্ট্রিং দিয়ে ইনপুট-আউটপুটের কাজ করা হলে প্রোগ্রামের লজিক অনেক সহজ হয়ে যায়। আর যদি সিঙ্গেল ক্যারেক্টার দিয়ে এসব কাজ করা হয়, তাহলে প্রোগ্রাম অনেক বড় ও জটিল হয়ে যায়। এ ছাড়া আধুনিক অনেক ল্যাঙ্গুয়েজের সব ইনপুট স্ট্রিং হিসেবে নেয়া হয়। পরে প্রয়োজনীয় ফাংশন কল করে তাকে অন্যান্য ডাটা টাইপে কনভার্ট করা হয় কজ

ফিডব্যাক : wahid_cseaust@yahoo.com