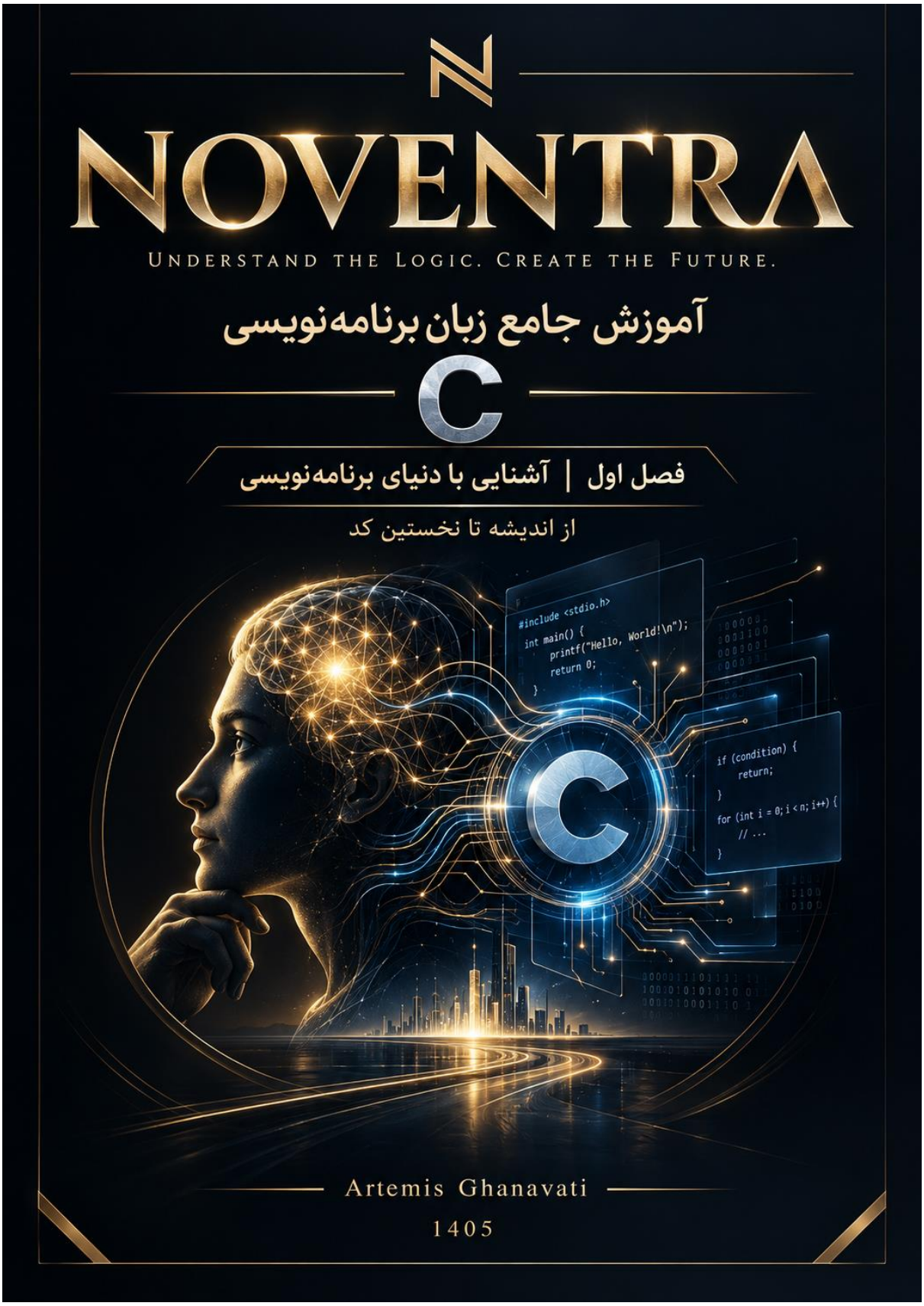

NOVENTRA
UNDERSTAND THE LOGIC. CREATE THE FUTURE.
آموزش جامع زبان برنامه‌نویسی
C
فصل اول | آشنایی با دنیای برنامه‌نویسی
از اندیشه تا نخستین کد
#include <stdio.h>
int main() {
printf("Hello, World!\n");
return 0;
}
if (condition) {
return;
}
for (int i = 0; i < n; i++) {
// ...
}
Artemis Ghanavati
1405

فصل اول

آغاز یک اندیشه؛ از تفکر تا آفرینش

به دنیای برنامه‌نویسی خوش آمدید.

پیش از آنکه نخستین خط کد را بنویسید، می‌خواهم یک حقیقت را با شما در میان بگذارم:

جهان با یک فکر تغییر می‌کند؛ فکری که شاید در ابتدا کوچک، ساده و حتی غیرممکن به نظر برسد، اما روزی می‌تواند به چیزی تبدیل شود که زندگی انسان‌ها را دگرگون کند.

برنامه‌نویسی، فقط نوشتن دستورهایی برای یک رایانه نیست. برنامه‌نویسی هنر اندیشیدن، قدرت حل مسئله و توانایی تبدیل یک ایده به واقعیت است.

هر نرم‌افزاری که امروز از آن استفاده می‌کنید، روزی تنها یک تصور در ذهن انسانی بوده است. هر وب‌سایتی که می‌بینید، هر سامانه‌ای که کار می‌کند و هر فناوری‌ای که زندگی را آسان‌تر کرده، از یک نقطه آغاز شده است :**یک سؤال، یک ایده و اراده‌ای برای ساختن.**

در این مسیر، قرار نیست فقط کدها را حفظ کنید. می‌خواهم یاد بگیرید چرا یک دستور نوشته می‌شود، چگونه یک مسئله به بخش‌های کوچک‌تر تقسیم می‌شود و چطور می‌توان با منطق، خلاقیت و دانش، برای آن راه‌حل ساخت.

ممکن است در آغاز، بعضی مفاهیم دشوار باشند. ممکن است کدی بنویسید که اجرا نشود، نتیجه‌ای بگیرید که اشتباه باشد یا بارها مجبور شوید به عقب برگردید و دوباره تلاش کنید. اما فراموش نکنید: خطا، پایان مسیر نیست؛ بخشی از فرایند یادگیری و کشف است.

من می‌خواهم در پایان این آموزش، شما تنها کسی نباشید که می‌تواند یک برنامه را اجرا کند؛ می‌خواهم به نقطه‌ای برسید که بتوانید مسئله‌ای را تحلیل کنید، راه‌حل طراحی کنید، کد بنویسید، خطاهای خود را پیدا کنید و با اعتمادبه‌نفس پروژه‌ای مستقل بسازید.

هدف من تربیت افرادی نیست که فقط از فناوری استفاده می‌کنند؛ هدفم پرورش ذهن‌هایی است که توانایی خلق فناوری را دارند.

پس از همین لحظه، با کنجکاوی سؤال بپرسید، با دقت مشاهده کنید، با منطق بیندیشید و از اشتباه کردن نترسید.

شاید امروز فقط اولین خط کدتان را بنویسید؛ اما هیچ‌کس نمی‌داند همین خط اول، شما را تا کجا خواهد برد.

به NOVENTRA خوش آمدید؛ جایی که اندیشه به منطق تبدیل می‌شود، منطق به خلاقیت می‌رسد و خلاقیت، آینده را می‌سازد.

Understand the Logic. Create the Future.

—نویسنده و مدرس: آرتیمیس قنواتی

درس اول: برنامه‌نویسی چیست؟

۱.۱ تعریف برنامه‌نویسی(Programming)

برنامه‌نویسی فرایند طراحی، تدوین و نوشتن مجموعه‌ای از دستورالعمل‌های منطقی و ساختاریافته است که به رایانه ارائه می‌شوند تا وظیفه‌ای مشخص را انجام دهد یا مسئله‌ای را حل کند.

به بیان ساده‌تر، برنامه‌نویسی یعنی مشخص کنیم رایانه چه کاری را، با چه ترتیبی و تحت چه شرایطی انجام دهد.

رایانه برخلاف انسان، منظور ما را از طریق زبان طبیعی به‌طور مستقیم درک نمی‌کند. بنابراین باید خواسته‌های خود را با استفاده از زبان‌های برنامه‌نویسی و بر اساس قواعد مشخص بیان کنیم تا رایانه بتواند آن‌ها را پردازش و اجرا کند.

۱.۲ چرا به برنامه‌نویسی نیاز داریم؟

امروزه برنامه‌نویسی در بخش‌های مختلف زندگی، علم و صنعت کاربرد دارد. برخی از مهم‌ترین حوزه‌های استفاده از برنامه‌نویسی عبارت‌اند از:

۱ .توسعه نرم‌افزار

طراحی و تولید نرم‌افزارهای کامپیوتری، برنامه‌های موبایل و ابزارهای کاربردی.

۲. صنعت و اتوماسیون

کنترل دستگاه‌ها، ماشین‌آلات و فرایندهای صنعتی با استفاده از برنامه‌های کامپیوتری.

۳. هوش مصنوعی

پردازش داده‌ها، یادگیری ماشین و توسعه سامانه‌های هوشمند.

۴. طراحی وب

ساخت وب‌سایت‌ها، فروشگاه‌های اینترنتی و سامانه‌های تحت وب.

۱.۳ یک مثال ساده از منطق برنامه‌نویسی

فرض کنید می‌خواهیم برنامه‌ای طراحی کنیم که مجموع دو عدد را محاسبه کند.

برای انجام این کار، ابتدا باید مراحل حل مسئله را مشخص کنیم:

۱. دریافت عدد اول
۲. دریافت عدد دوم
۳. محاسبه مجموع دو عدد
۴. نمایش نتیجه

در این مثال هنوز از زبان برنامه‌نویسی C استفاده نکرده‌ایم. ابتدا منطق حل مسئله را مشخص کرده‌ایم و سپس می‌توانیم همین مراحل را با استفاده از یک زبان برنامه‌نویسی به کد تبدیل کنیم.

این موضوع یکی از مهم‌ترین اصول برنامه‌نویسی است: پیش از نوشتن کد، باید مسئله را بشناسیم و روش حل آن را مشخص کنیم.

تمرین پایان درس اول

فرض کنید می‌خواهیم برنامه‌ای طراحی کنیم که میانگین سه عدد را محاسبه کند.

به نظر شما برای انجام این کار، چه مراحلی باید طی شود؟

راهنما: فعلاً کدنویسی نکنید. فقط مراحل حل مسئله را به ترتیب بنویسید.

درس دوم: آشنایی با زبان برنامه‌نویسی C

۲.۱ زبان برنامه‌نویسی چیست؟

زبان برنامه‌نویسی مجموعه‌ای از قواعد، نمادها و دستورالعمل‌های مشخص است که برنامه‌نویس با استفاده از آن‌ها می‌تواند خواسته‌ها و روش حل مسائل را به شکلی قابل پردازش برای رایانه بیان کند.

هر زبان برنامه‌نویسی دارای قواعد دستوری (Syntax) و ساختار مشخصی است. همان‌طور که در زبان‌های انسانی برای ساخت جمله از قواعد خاصی استفاده می‌کنیم، در زبان‌های برنامه‌نویسی نیز باید دستورها را مطابق قواعد همان زبان بنویسیم.

اگر این قواعد رعایت نشوند، ممکن است برنامه با خطای دستوری (Syntax Error) روبه‌رو شود.

۲.۲ معرفی زبان C

زبان C یکی از زبان‌های مهم و تأثیرگذار در تاریخ برنامه‌نویسی است. این زبان در اوایل دهه ۱۹۷۰ میلادی توسط دنیس ریچی (Dennis Ritchie) در آزمایشگاه‌های بل (Bell Labs)توسعه داده شد.

زبان C به دلیل سرعت اجرای بالا، انعطاف‌پذیری و امکان کنترل دقیق‌تر منابع سخت‌افزاری و حافظه، در توسعه بسیاری از نرم‌افزارها و سیستم‌های کامپیوتری مورد استفاده قرار گرفته است.

این زبان همچنان یکی از پایه‌های مهم آموزش برنامه‌نویسی و توسعه نرم‌افزارهای سیستمی محسوب می‌شود.

۲.۳ ویژگی‌های زبان C

مهم‌ترین ویژگی‌های زبان C عبارت‌اند از:

۱. **ساختار دستوری منظم:** دارای قواعد مشخص و ساختاری نسبتاً ساده برای یادگیری اصول برنامه‌نویسی است.

۲. **سرعت اجرای بالا:** برنامه‌های C پس از کامپایل، معمولاً با سرعت مناسبی اجرا می‌شوند.

۳. **قابلیت انتقال:** با استفاده از کامپایلر مناسب، می‌توان برنامه‌های C را برای سیستم‌عامل‌ها و معماری‌های مختلف آماده کرد؛ البته ممکن است برای انتقال، تغییراتی در کد لازم باشد.

۴. **پشتیبانی از برنامه‌نویسی ساخت‌یافته:** امکان تقسیم برنامه به بخش‌ها و توابع کوچک‌تر را فراهم می‌کند.

۵. **امکان کار با حافظه و آدرس‌ها:** برنامه‌نویس می‌تواند با استفاده از اشاره‌گرها (Pointers) به آدرس‌های حافظه دسترسی داشته باشد و آن‌ها را مدیریت کند.

۶. **کاربرد در برنامه‌نویسی سیستمی و نهفته:** در توسعه سیستم‌عامل‌ها، میکروکنترلرها و تجهیزات الکترونیکی کاربرد گسترده‌ای دارد.

۲.۴ کاربردهای زبان C

زبان C در حوزه‌های مختلفی استفاده می‌شود. برخی از کاربردهای مهم آن عبارت‌اند از:

۱. سیستم‌عامل‌ها

برای توسعه بخش‌هایی از سیستم‌عامل‌ها، هسته (Kernel) و ابزارهای سیستمی.

۲. سیستم‌های نهفته (Embedded Systems)

برای برنامه‌نویسی میکروکنترلرها و تجهیزاتی که درون دستگاه‌های الکترونیکی و صنعتی قرار دارند.

۳. صنعت و تجهیزات هوشمند

برای کنترل تجهیزات، ماشین‌آلات و سامانه‌های صنعتی.

۴. نرم‌افزارهای سطح پایین

برای توسعه برنامه‌هایی که به ارتباط نزدیک‌تری با سخت‌افزار و منابع سیستم نیاز دارند.

۲.۵ تفاوت زبان C با زبان‌های دیگر

هر زبان برنامه‌نویسی با توجه به اهداف و کاربردهای خاصی طراحی شده است. بنابراین نمی‌توان یک زبان را برای تمام پروژه‌ها بهترین گزینه دانست.

در جدول زیر، چند زبان پرکاربرد و حوزه اصلی استفاده از آن‌ها را مشاهده می‌کنید:

زبان	کاربرد اصلی
C	برنامه‌نویسی سیستمی و نزدیک به سخت‌افزار
C++	نرم‌افزارهای پیچیده و برنامه‌نویسی شیءگرا
Java	نرم‌افزارهای سازمانی و چندسکویی
Python	هوش مصنوعی، تحلیل داده و اتوماسیون
HTML	ساختاردهی صفحات وب

نکته مهم HTML :یک زبان نشانه‌گذاری (Markup Language) است، نه یک زبان برنامه‌نویسی عمومی. از HTML برای مشخص‌کردن ساختار و محتوای صفحات وب استفاده می‌شود.

تمرین پایان درس دوم

۱. زبان **C** توسط چه کسی و در چه دوره‌ای توسعه داده شد؟

۲. دو مورد از ویژگی‌های زبان **C** را نام ببرید.

۳. زبان **C** در چه حوزه‌هایی کاربرد دارد؟

۴. منظور از قابلیت انتقال در زبان **C** چیست؟

۵. آیا **HTML** یک زبان برنامه‌نویسی عمومی است؟ دلیل خود را توضیح دهید.

درس سوم: کامپایلر چیست؟

۳.۱ تعریف کامپایلر(Compiler)

کامپایلر نرم‌افزاری است که کد نوشته‌شده توسط برنامه‌نویس را ترجمه می‌کند تا سیستم بتواند آن را اجرا کند یا در مراحل بعدی ساخت برنامه مورد استفاده قرار دهد.

برای مثال، برنامه‌نویس هنگام نوشتن یک برنامه به زبان C از دستورهایی مانند printf استفاده می‌کند؛ اما پردازنده نمی‌تواند دستورهای زبان C را مستقیماً اجرا کند.

پردازنده در نهایت دستورهای ماشین را اجرا می‌کند. بنابراین کد نوشته‌شده باید با کمک ابزارهای ترجمه و ساخت برنامه به قالب قابل اجرا تبدیل شود.

۳.۲ مراحل کلی تبدیل برنامه C به فایل اجرایی

فرایند ساخت یک برنامه C معمولاً شامل چهار مرحله اصلی است:

مرحله اول: پیش‌پردازش(Preprocessing)

در این مرحله، دستورهای پیش‌پردازنده پردازش می‌شوند.

برای مثال:

- دستور #include برای واردکردن محتوای فایل‌های سرآیند استفاده می‌شود.
- دستور #define برای تعریف ماکروها به کار می‌رود.

خروجی این مرحله، کدی است که برای مراحل بعدی آماده شده است.

مرحله دوم: کامپایل(Compilation)

در این مرحله، کد منبع بررسی و ترجمه می‌شود. کامپایلر ابتدا ساختار و قواعد دستوری برنامه را بررسی می‌کند و سپس کد را به زبان اسمبلی یا یک نمایش میانی تبدیل می‌کند.

اگر خطاهای دستوری یا خطاهای قابل تشخیص دیگری وجود داشته باشند، ممکن است در این مرحله گزارش شوند.

مرحله سوم: اسمبل(Assembly)

در این مرحله، کد اسمبلی به دستورهای ماشین تبدیل می‌شود و معمولاً در قالب فایل شیء (Object File) ذخیره می‌شود.

فایل شیء به‌تنهایی لزوماً یک برنامه کامل و قابل اجرا نیست؛ زیرا ممکن است هنوز به بخش‌ها یا کتابخانه‌های دیگری نیاز داشته باشد.

مرحله چهارم: پیونددهی(Linking)

در این مرحله، فایل‌های شیء و کتابخانه‌های موردنیاز به یکدیگر متصل می‌شوند.

هدف از پیونددهی، تکمیل بخش‌های موردنیاز برنامه و ایجاد فایل اجرایی است.

۳.۳ تفاوت کامپایلر و مفسر(Interpreter)

کامپایلر و مفسر هر دو ابزارهایی برای پردازش و اجرای برنامه‌ها هستند، اما روش کار آن‌ها متفاوت است.

مفسر(Interpreter)	کامپایلر(Compiler)
معمولاً دستورها را هنگام اجرا پردازش می‌کند.	معمولاً برنامه را پیش از اجرا ترجمه می‌کند.
لزوماً فایل اجرایی مستقل تولید نمی‌کند.	اغلب خروجی قابل اجرا یا فایل‌های میانی تولید می‌کند.
نمونه: مفسر CPython برای Python	نمونه GCC :برای زبان C

نکته :این جدول تفاوت را به‌صورت آموزشی و ساده بیان می‌کند. در عمل، روش اجرای زبان‌ها می‌تواند پیچیده‌تر باشد و بعضی زبان‌ها از ترکیبی از روش‌های ترجمه و اجرا استفاده می‌کنند.

۳.۴ چند نمونه از کامپایلرهای زبانC

برخی از ابزارهای پرکاربرد برای کامپایل برنامه‌های C عبارت‌اند از:

۱. GCC

مجموعه کامپایلرهای GNU است که از زبان C و چند زبان برنامه‌نویسی دیگر پشتیبانی می‌کند.

۲. Clang

یک کامپایلر و ابزار زنجیره‌ای مرتبط با پروژه LLVM است که از زبان C پشتیبانی می‌کند.

۳. Microsoft Visual C Compiler (MSVC)

کامپایلر شرکت مایکروسافت برای توسعه برنامه‌ها در محیط ویندوز است.

این ابزارها در محیط‌ها و سیستم‌های مختلف برای ترجمه و ساخت برنامه‌های C مورد استفاده قرار می‌گیرند.

تمرین پایان درس سوم

۱. چرا کد نوشته‌شده به زبان **C** باید ترجمه شود؟

۲. پیش‌پردازش چه وظیفه‌ای دارد؟

۳. تفاوت مرحله کامپایل و اسمبل چیست؟

۴. وظیفه پیونددهنده **(Linker)** چیست؟

۵. دو نمونه از کامپایلرهای زبان **C** را نام ببرید.

۶. تفاوت کلی کامپایلر و مفسر را توضیح دهید.

درس چهارم: الگوریتم و تبدیل آن به برنامه(Algorithm)

۴.۱ الگوریتم چیست؟

الگوریتم مجموعه‌ای از دستورالعمل‌های دقیق، مرتب و محدود است که برای حل یک مسئله یا انجام یک کار طراحی می‌شود.

هر الگوریتم باید ویژگی‌های زیر را داشته باشد:

۱. در صورت نیاز، ورودی‌های مشخصی داشته باشد.

۲. مراحل آن روشن و قابل اجرا باشند.

۳. در نهایت به خروجی یا نتیجه مشخصی برسد.

۴. پس از تعداد محدودی مرحله پایان پیدا کند.

۴.۲ مثال: محاسبه مجموع دو عدد

فرض کنیم می‌خواهیم دو عدد را از کاربر دریافت کنیم و مجموع آن‌ها را نمایش دهیم.

مراحل الگوریتم:

۱. شروع

۲. دریافت عدد اول و عدد دوم

۳. محاسبه مجموع دو عدد

۴. نمایش نتیجه

۵. پایان

در این مثال، ابتدا اطلاعات موردنیاز را دریافت می‌کنیم، سپس عملیات جمع را انجام می‌دهیم و در پایان نتیجه را نمایش می‌دهیم.

۴.۳ شبه‌کد(Pseudocode)

شبه‌کد روشی برای بیان مراحل الگوریتم با زبانی ساده و نزدیک به زبان برنامه‌نویسی است؛ بدون اینکه به قواعد دستوری یک زبان خاص وابسته باشد.

شبه‌کد مثال محاسبه مجموع دو عدد:

```
شروع
    دریافت عدد اول
    دریافت عدد دوم
    مجموع = عدد اول + عدد دوم
    نمایش مجموع
پایان
```

شبه‌کد به برنامه‌نویس کمک می‌کند پیش از نوشتن کد، منطق حل مسئله را مشخص کند.

۴.۴ تفاوت الگوریتم و برنامه(Program)

الگوریتم :مجموعه‌ای از مراحل منطقی و مرتب برای حل یک مسئله است.

برنامه :پیاده‌سازی الگوریتم با استفاده از قواعد و دستورهای یک زبان برنامه‌نویسی است.

به عبارت دیگر، الگوریتم روش حل مسئله را مشخص می‌کند و برنامه آن روش را به زبان قابل پردازش برای رایانه تبدیل می‌کند.

برای مثال، الگوریتم محاسبه میانگین سه عدد را می‌توان با زبان C به یک برنامه تبدیل کرد.

۴.۵ پیاده‌سازی الگوریتم در زبان C

در این مثال، برنامه‌ای می‌نویسیم که سه عدد را از کاربر دریافت کند، میانگین آن‌ها را محاسبه کند و نتیجه را نمایش دهد.

```
#include <stdio.h>

int main()
{
    float a, b, c, average;

    printf("Enter three numbers: ");
    scanf("%f %f %f", &a, &b, &c);

    average = (a + b + c) / 3.0;

    printf("Average = %.2f", average);

    return 0;
}
```

۴.۶ تحلیل دستورات برنامه

- `#include <stdio.h>` برای استفاده از توابع ورودی و خروجی مانند `printf` و `scanf` است.
- `int main()` تابع اصلی برنامه را تعریف می‌کند.
- `float a, b, c, average;` چهار متغیر از نوع اعشاری تعریف می‌کند.
- `printf()` پیام دریافت اطلاعات را نمایش می‌دهد.
- `scanf()` سه عدد را از ورودی دریافت می‌کند.
- `average = (a + b + c) / 3.0;` میانگین سه عدد را محاسبه می‌کند.

- `printf("Average = %.2f", average);` نتیجه را با دو رقم اعشار نمایش می‌دهد.
- `return 0;` پایان موفقیت‌آمیز اجرای تابع اصلی را نشان می‌دهد.

۴.۷ آشنایی با مفهوم ورودی، پردازش و خروجی

یکی از مدل‌های پایه‌ای در طراحی برنامه‌ها، مدل ورودی، پردازش و خروجی است.

۱. ورودی(Input)

داده‌هایی که برنامه دریافت می‌کند.

مثال: ۱۰، ۲۰ و ۳۰

۲. پردازش(Processing)

عملیاتی که روی داده‌های ورودی انجام می‌شود.

مثال:

(۱۰ + ۲۰ + ۳۰ ÷ ۳)

۳. خروجی(Output)

نتیجه‌ای که پس از پردازش به دست می‌آید.

مثال: ۲۰٫۰۰

این مدل یکی از پایه‌ای‌ترین مفاهیم علوم کامپیوتر است و در طراحی بسیاری از برنامه‌ها کاربرد دارد.

تمرین هنرجویان

برنامه‌ای طراحی کنید که دو عدد را از کاربر دریافت کند و حاصل‌ضرب آن‌ها را نمایش دهد.

هدف تمرین :آشنایی عملی با تعریف متغیر، دریافت ورودی، انجام پردازش و نمایش خروجی.

درس پنجم: انواع داده‌ها(Data Types)

۵.۱ نوع داده چیست؟

هر متغیر در برنامه‌نویسی برای نگهداری یک مقدار استفاده می‌شود؛ اما همه مقادیر از یک جنس نیستند.

برای مثال:

- سن یک فرد: عدد صحیح
- قد یک فرد: عدد اعشاری
- حرف اول نام: یک کاراکتر
- وضعیت روشن یا خاموش بودن دستگاه: مقدار منطقی

در زبان C برای مشخص‌کردن نوع اطلاعاتی که یک متغیر می‌تواند نگهداری کند، از مفهومی به نام نوع داده (Data Type) استفاده می‌کنیم.

نوع داده مشخص می‌کند چه نوع مقداری در متغیر ذخیره شود و چه عملیاتی را می‌توان روی آن انجام داد.

۵.۲ انواع داده‌های پایه در زبانC

در این بخش با چهار نوع داده پرکاربرد در زبان C آشنا می‌شویم:

۱ .نوع دادهint

برای نگهداری اعداد صحیح استفاده می‌شود.

مثال:

```
int age = 25;
int temperature = -8;
```

۲ .نوع دادهfloat

برای نگهداری اعداد اعشاری با دقت معمولی استفاده می‌شود.

مثال:

```
float height = 172.5;
float average = 18.75;
```

۳. نوع داده double

برای نگهداری اعداد اعشاری با دقت بیشتر نسبت به `float` استفاده می‌شود.

مثال:

```
double number = 3.1415926535;
```

۴. نوع داده char

برای نگهداری یک کاراکتر استفاده می‌شود.

مثال:

```
char grade = 'A';
char letter = 'B';
char digit = '7';
```

نکته :برای نوشتن مقدار یک کاراکتر در زبان C از علامت نقل‌قول تکی استفاده می‌کنیم؛ مانند 'A'.

۵.۳ آشنایی با نوع داده منطقی

در زبان C استاندارد C99 و نسخه‌های بعدی، نوع داده _Bool برای نگهداری مقادیر منطقی وجود دارد.

همچنین با اضافه‌کردن فایل سرآیند زیر می‌توان از نام bool و مقادیر true و false استفاده کرد:

```
#include <stdbool.h>
```

مثال:

```
#include <stdio.h>
#include <stdbool.h>


int main()
{
bool isOn = true;


printf("%d", isOn);


return 0;
}
```

در این مثال، مقدار منطقی trueهنگام نمایش با %dبه‌صورت عدد ۱ نشان داده می‌شود.

۵.۴ نحوه تعریف متغیرها

برای تعریف متغیر در زبان C از ساختار زیر استفاده می‌کنیم:

```
data_type variable_name;
```

در این ساختار:

- data_typeنوع داده را مشخص می‌کند.
- variable_nameنام متغیر است.

مثال:

```
int age;
float height;
double price;
char grade;
```

در این مثال، چهار متغیر با چهار نوع داده متفاوت تعریف شده‌اند.

۵.۵ تفاوت floatو double

هر دو نوع داده floatو doubleبرای نگهداری اعداد اعشاری استفاده می‌شوند؛ اما تفاوت‌هایی دارند.

- floatمعمولاً دقت کمتری دارد و فضای حافظه کمتری مصرف می‌کند.
- doubleمعمولاً دقت بیشتری دارد و فضای حافظه بیشتری مصرف می‌کند.

بنابراین انتخاب نوع داده به مقدار موردنیاز و میزان دقت محاسبات بستگی دارد.

۵.۶ نمایش متغیرها باprintf

برای نمایش مقدار متغیرها با استفاده از تابع printf، باید از مشخص‌کننده قالب (Format Specifier)مناسب استفاده کنیم.

نوع داده	مشخص‌کننده درprintf
int	%d
float	%f
double	%f

%c	char

مثال:

```
#include <stdio.h>

int main()
{
    int age = 25;
    float height = 172.5;
    double price = 1250.75;
    char grade = 'A';

    printf("Age: %d\n", age);
    printf("Height: %f\n", height);
    printf("Price: %f\n", price);
    printf("Grade: %c\n", grade);

    return 0;
}
```

نکته مهم :در تابع printfبرای نمایش مقدار doubleنیز از %fاستفاده می‌کنیم؛ اما در تابع scanfبرای دریافت مقدار doubleباید از %lfاستفاده شود.

تمرین پایان درس پنجم

۱ .نوع داده مناسب برای نگهداری سن یک فرد چیست؟

۲ .برای ذخیره‌سازی یک عدد اعشاری با دقت بیشتر، از کدام نوع داده استفاده می‌کنیم؟

۳ .تفاوت کلی `float` و `double` چیست؟

۴ .برای نگهداری یک کاراکتر از کدام نوع داده استفاده می‌شود؟

۵ .برای نمایش مقدار یک متغیر از نوع `char` در تابع `printf` از چه مشخص‌کننده‌ای استفاده می‌کنیم؟

۶ .چرا انتخاب نوع داده مناسب در برنامه‌نویسی اهمیت دارد؟

تمرین عملی:

برنامه‌ای بنویسید که سن، قد، قیمت یک کالا و حرف اول نام یک فرد را در متغیرهای مناسب ذخیره کند و سپس مقادیر آن‌ها را با استفاده از printf نمایش دهد.

درس ششم: عملگرها در زبان C (Operators)

۶.۱ عملگر چیست؟

عملگرها نمادهایی هستند که برای انجام عملیات مختلف روی داده‌ها و متغیرها استفاده می‌شوند.

برای مثال، در عبارت زیر:

```
result = a + b;
```

علامت + یک عملگر حسابی برای جمع دو مقدار است و علامت = مقدار حاصل را در متغیر result قرار می‌دهد.

۶.۲ انواع عملگرها در زبان C

الف) عملگرهای حسابی (Arithmetic Operators)

این عملگرها برای انجام محاسبات ریاضی استفاده می‌شوند.

مثال	عملکرد	عملگر
a + b	جمع	+
a - b	تفریق	-
a * b	ضرب	*
a / b	تقسیم	/
a % b	باقیمانده تقسیم	%

مثال:

```
#include <stdio.h>

int main()
{
    int a = 10;
    int b = 3;

    printf("%d", a + b);

    return 0;
}
```

خروجی:

13

نکته مهم درباره تقسیم صحیح:

وقتی هر دو عملوند از نوع صحیح باشند، نتیجه تقسیم نیز از نوع صحیح خواهد بود و بخش اعشاری حذف می‌شود.

مثال:

```
int a = 15;
int b = 4;

printf("%d", a / b);
```

خروجی:

3

اما اگر بخواهیم نتیجه اعشاری داشته باشیم، باید از نوع داده اعشاری استفاده کنیم.

```
float a = 15;
float b = 4;


printf("%f", a / b);
```

خروجی:

3.750000

نکته :عملگر %برای محاسبه باقیمانده تقسیم اعداد صحیح استفاده می‌شود.

ب) عملگرهای مقایسه‌ای(Relational Operators)

این عملگرها برای مقایسه دو مقدار استفاده می‌شوند.

مفهوم	عملگر
مساوی بودن	==
نامساوی بودن	!=
بزرگ‌تر بودن	>
کوچک‌تر بودن	<
بزرگ‌تر یا مساوی	>=
کوچک‌تر یا مساوی	<=

نتیجه مقایسه در زبان C از نوع عدد صحیح است:

- عدد 1به معنی درست(True)
- عدد 0به معنی نادرست(False)

مثال:

```
int a = 10;
int b = 5;
```

```
printf("%d", a > b);
```

خروجی:

1

زیرا عدد ۱۰ از عدد ۵ بزرگ‌تر است.

ج) عملگرهای منطقی(Logical Operators)

این عملگرها برای ترکیب شرط‌ها یا معکوس‌کردن نتیجه یک شرط استفاده می‌شوند.

عملگر	مفهوم	توضیح
&&	AND یا و	هر دو شرط باید درست باشند.
`	OR یا یا	`
		درست‌بودن حداقل یکی از شرط‌ها کافی است.
!	NOT یا نقیض	نتیجه شرط را معکوس می‌کند.

مثال:

```
int age = 25;
```

```
printf("%d", age >= 18 && age <= 60);
```

خروجی:

1

زیرا هر دو شرط درست هستند.

د) عملگرهای انتساب(Assignment Operators)

عملگر انتساب برای قرار دادن یک مقدار در متغیر استفاده می‌شود.

ساده‌ترین عملگر انتساب =است.

```
int a;
a = 10;
```

یعنی مقدار ۱۰ در متغیر aذخیره می‌شود.

زبان C علاوه بر انتساب ساده، عملگرهای انتساب ترکیبی نیز دارد.

مفهوم	معادل عبارت	عملگر
جمع و انتساب	a = a + 5	+=
تفریق و انتساب	a = a - 5	-=
ضرب و انتساب	a = a * 5	*=
تقسیم و انتساب	a = a / 5	/=
باقیمانده و انتساب	a = a % 5	%=

مثال:

```
int a = 10;
```

a += 5;

```
printf("%d", a);
```

خروجی:

15

هـ) عملگرهای افزایش و کاهش(Increment and Decrement)

در زبان C دو عملگر مهم برای تغییر مقدار یک متغیر به اندازه یک واحد وجود دارد:

- ++افزایش مقدار متغیر به اندازه یک واحد
- --کاهش مقدار متغیر به اندازه یک واحد

مثال افزایش:

```
int count = 5;
```

++count;

```
printf("%d", count);
```

خروجی:

6

مثال کاهش:

```
int count = 5;
```

--count;

```
printf("%d", count);
```

خروجی:

4

۶.۳ تفاوت افزایش پیشوندی و پسوندی

این بخش اهمیت زیادی دارد؛ زیرا محل قرارگرفتن عملگر می‌تواند ترتیب تغییر مقدار و استفاده از آن را مشخص کند.

۱ .افزایش پیشوندی(Prefix Increment)

در حالت پیشوندی، ابتدا مقدار متغیر افزایش پیدا می‌کند و سپس مقدار جدید در عبارت استفاده می‌شود.

مثال:

```
int a = 5;
int b = ++a;
```

ابتدا مقدار aبه ۶ افزایش می‌یابد و سپس مقدار جدید در bقرار می‌گیرد.

نتیجه:

a = 6

b = 6

۲ .افزایش پسوندی(Postfix Increment)

در حالت پسوندی، ابتدا مقدار فعلی متغیر در عبارت استفاده می‌شود و سپس مقدار متغیر افزایش پیدا می‌کند.

مثال:

```
int a = 5;
```

```
int b = a++;
```

ابتدا مقدار ۵ در bقرار می‌گیرد و سپس مقدار aبه ۶ افزایش می‌یابد.

نتیجه:

a = 6

b = 5

همین مفهوم برای عملگرهای کاهش پیشوندی و پسوندی نیز برقرار است.

۶.۴ تقدم عملگرها(Operator Precedence)

وقتی چند عملگر در یک عبارت وجود دارند، ترتیب انجام آن‌ها اهمیت دارد.

برای مثال:

```
int result = 2 + 3 * 4;
```

ابتدا ضرب انجام می‌شود و سپس جمع:

2 + 12 = 14

بنابراین مقدار resultبرابر ۱۴ خواهد بود.

اما با استفاده از پرانتز:

```
int result = (2 + 3) * 4;
```

ابتدا عبارت داخل پرانتز محاسبه می‌شود:

5 * 4 = 20

بنابراین مقدار resultبرابر ۲۰ خواهد بود.

نکته:پرانتز به برنامه‌نویس کمک می‌کند ترتیب موردنظر خود را به‌صورت روشن مشخص کند.

تمرین تحلیل کد

کد زیر را بررسی کنید و خروجی آن را مشخص کنید:

```
int count = 5;
int result;
```

result = count++;

```
printf("%d\n", result);
```

```
printf("%d\n", count);
```

گزینه صحیح را انتخاب کنید:

۱. ابتدا ۵ و سپس ۵

۲. ابتدا ۵ و سپس ۶

۳. ابتدا ۶ و سپس ۶

۴. ابتدا ۶ و سپس ۵

تمرین پایان درس ششم

۱. تفاوت عملگرهای =و ==چیست؟

---

۲. حاصل عبارت `15 / 4`در صورتی که هر دو مقدار از نوع `int`باشند، چقدر است؟

۳. عملگر **%**چه کاربردی دارد؟

۴. تفاوت عملگرهای **&&**و **||**چیست؟

۵. تفاوت افزایش پیشوندی و پسوندی را با مثال توضیح دهید.

۶. چرا در عبارت **2 + 3 * 4**ابتدا ضرب انجام می‌شود؟

تمرین عملی:

برنامه‌ای بنویسید که دو عدد صحیح دریافت کند و حاصل جمع، تفریق، ضرب و باقیمانده تقسیم آن‌ها را نمایش دهد.

درس هفتم: دستورات ورودی و خروجی در زبانC

۷.۱ مفهوم ورودی، پردازش و خروجی

در برنامه‌نویسی، برنامه معمولاً با داده‌هایی کار می‌کند که از منابع مختلف دریافت می‌شوند. این داده‌ها پس از پردازش، نتیجه‌ای تولید می‌کنند.

سه مفهوم اصلی در این فرایند عبارت‌اند از:

۱. **ورودی (Input):** اطلاعاتی که وارد برنامه می‌شوند.

۲. **پردازش (Processing):** عملیاتی که روی اطلاعات ورودی انجام می‌شود.

۳. **خروجی (Output):** نتیجه‌ای که برنامه پس از پردازش تولید یا نمایش می‌دهد.

برای مثال، در برنامه‌ای که مجموع دو عدد را محاسبه می‌کند:

- ورودی: دو عدد دریافت‌شده از کاربر
- پردازش: جمع دو عدد
- خروجی: نمایش حاصل جمع

۷.۲ دستور خروجی printf

در زبان c از تابع `printf()` برای نمایش اطلاعات استفاده می‌کنیم.

برای استفاده از این تابع باید فایل سرآیند زیر را به برنامه اضافه کنیم:

```
#include <stdio.h>
```

مثال:

```
#include <stdio.h>

int main()
{
printf("Hello NOVENTRA!");

return 0;
}
```

خروجی:

Hello NOVENTRA!

نمایش مقدار متغیرها

با استفاده از `printf()` می‌توانیم مقدار متغیرها را نیز نمایش دهیم.

مثال:

```
#include <stdio.h>

int main()
```

```
{
    int age = 25;

    printf("Age = %d", age);

    return 0;
}
```

خروجی:

```
Age = 25
```

در این مثال، %dمشخص‌کننده قالب نمایش عدد صحیح است و مقدار متغیر ageدر محل آن نمایش داده می‌شود.

۷.۳ مشخص‌کننده‌های قالب(Format Specifiers)

مشخص‌کننده‌های قالب تعیین می‌کنند که مقدار داده چگونه نمایش داده شود یا از ورودی دریافت شود.

کاربرد	مشخص‌کننده
عدد صحیح از نوع int	%d
عدد اعشاری از نوع floatدر printf	%f
دریافت مقدار doubleدر scanf	%lf
یک کاراکتر	%c
رشته کاراکتری	%s
عدد صحیح بدون علامت	%u
نمایش عدد صحیح در مبنای ۱۶	%x

نکته مهم :برای نمایش مقدار doubleدر تابع printf()نیز از %fاستفاده می‌کنیم؛ اما برای دریافت مقدار doubleبا تابع scanf()باید از %lfاستفاده شود.

۷.۴ دستور ورودی scanf

تابع scanf() برای دریافت داده از ورودی استاندارد، که معمولاً صفحه‌کلید است، استفاده می‌شود.

مثال:

```
#include <stdio.h>

int main()
{
    int age;

    printf("Enter your age: ");
    scanf("%d", &age);

    printf("Your age is %d", age);

    return 0;
}
```

در این برنامه ابتدا از کاربر خواسته می‌شود سن خود را وارد کند. سپس مقدار واردشده در متغیر age ذخیره و نمایش داده می‌شود.

چرا از علامت & استفاده می‌کنیم؟

در این کاربرد، علامت & آدرس متغیر را به تابع scanf() می‌دهد تا بتواند مقدار ورودی را در محل حافظه مربوط به آن متغیر ذخیره کند.

برای مثال:

```
scanf("%d", &age);
```

یعنی آدرس متغیر age را در اختیار تابع scanf() قرار بده تا مقدار ورودی را در آن ذخیره کند.

نکته تکمیلی: تابع scanf() تعداد ورودی‌هایی را که با موفقیت خوانده است برمی‌گرداند. در برنامه‌های واقعی بهتر است مقدار بازگشتی آن بررسی شود تا در صورت ورود داده نامعتبر، برنامه از مقداردهی ناموفق استفاده نکند.

V.۵ تفاوت scanf و printf

ویژگی	printf()	scanf()
کاربرد	نمایش اطلاعات	دریافت اطلاعات
جهت داده	از برنامه به خروجی	از ورودی به متغیر
نمونه	printf("%d", age);	scanf("%d", &age);

V.۶ دریافت رشته با scanf و fgets

برای دریافت یک واژه با آرایه کاراکتری می‌توان از مشخص‌کننده %s استفاده کرد.

مثال:

```
#include <stdio.h>

int main()
{
    char name[30];

    printf("Enter your name: ");
    scanf("%29s", name);

    printf("Your name is %s", name);

    return 0;
}
```

در این مثال، آرایه name ظرفیت نگهداری ۳۰ کاراکتر را دارد. استفاده از %29s باعث می‌شود حداکثر ۲۹ کاراکتر دریافت شود و یک خانه نیز برای کاراکتر پایان رشته باقی بماند.

نکته: تابع scanf() با %s دریافت رشته را در اولین فاصله متوقف می‌کند. برای دریافت نام‌ها یا عبارت‌های دارای فاصله، استفاده از تابع fgets() مناسب‌تر است.

۷.۷ برنامه نمونه: دریافت دو عدد و محاسبه مجموع

در این مثال برنامه‌ای می‌نویسیم که دو عدد صحیح را از کاربر دریافت کند و مجموع آن‌ها را نمایش دهد.

```
#include <stdio.h>

int main()
{
    int a, b, sum;

    printf("Enter two numbers: ");

    scanf("%d %d", &a, &b);

    sum = a + b;

    printf("Sum = %d", sum);

    return 0;
}
```

اگر کاربر اعداد ۱۲ و ۸ را وارد کند، خروجی چنین خواهد بود:

```
Sum = 20
```

در این برنامه:

۱. متغیرهای **a** و **b** برای نگهداری دو عدد تعریف شده‌اند.

۲. با استفاده از **scanf()** دو عدد دریافت می‌شوند.

۳. حاصل جمع در متغیر **sum** ذخیره می‌شود.

۴. با استفاده از **printf()** نتیجه نمایش داده می‌شود.

تمرین هنرجویان

برنامه‌ای بنویسید که نام و سن کاربر را دریافت کند و سپس آن‌ها را نمایش دهد.

قالب اولیه برنامه:

```
#include <stdio.h>

int main()
{
    char name[30];
    int age;

    // دریافت نام

    // دریافت سن

    // نمایش اطلاعات

    return 0;
}
```

راهنما:

- برای دریافت نام بدون فاصله از `scanf("%29s", name)` استفاده کنید.
- برای دریافت سن از `scanf("%d", &age)` استفاده کنید.
- برای نمایش اطلاعات از `printf()` استفاده کنید.

اگر نام دارای فاصله باشد، باید از `fgets()` استفاده شود.

فصل اول ـ درس هشتم: متغیرها، ثابت‌ها و قواعد نام‌گذاری در C

۱. متغیر چیست؟ (Variable)

متغیر فضایی در حافظه است که برای نگهداری یک مقدار استفاده می‌شود. مقدار ذخیره‌شده در متغیر می‌تواند در طول اجرای برنامه تغییر کند.

مثال:

```
int age = 20;
```

```
age = 25;
```

در ابتدا مقدار متغیر age برابر با ۲۰ است. سپس با اجرای دستور دوم، مقدار آن به ۲۵ تغییر می‌کند.

۲. تعریف و مقداردهی متغیر

برای تعریف یک متغیر در زبان C از ساختار زیر استفاده می‌کنیم:

```
data_type variable_name;
```

در این ساختار:

- data_type نوع داده‌ای را مشخص می‌کند که متغیر می‌تواند نگهداری کند.
- variable_name نامی است که برای متغیر انتخاب می‌کنیم.

مثال:

```c
int age;
float height;
char grade;
```

در این مثال، سه متغیر با انواع داده متفاوت تعریف شده‌اند:

- age از نوع عدد صحیح است.
- height از نوع عدد اعشاری است.
- grade از نوع کاراکتر است.

می‌توان هنگام تعریف متغیر، مقدار اولیه آن را نیز تعیین کرد:

```c
int age = 20;
float height = 165.5;
char grade = 'A';
```

به تعیین مقدار اولیه هنگام تعریف متغیر، **مقداردهی اولیه (Initialization)** گفته می‌شود.

۳. ثابت‌ها(Constants)

ثابت مقداری است که پس از تعریف، نباید در طول برنامه تغییر کند.

در زبان C می‌توان با استفاده از کلمه کلیدی const یک ثابت تعریف کرد.

مثال:

```c
const float PI = 3.14159;
```

در این مثال، متغیر ثابت PIبا مقدار اولیه ۳٫۱۴۱۵۹ تعریف شده است. تلاش برای تغییر مقدار آن، با محدودیت زبانی روبه‌رو می‌شود و کامپایلر باید درباره این خطا هشدار دهد یا آن را گزارش کند.

روش دیگر تعریف ثابت، استفاده از دستور پیش‌پردازنده #defineاست:

```
#define PI 3.14159
```

تفاوت مهم #define: وconst

- constیک شیء دارای نوع داده تعریف می‌کند.
- #defineیک دستور پیش‌پردازنده است که پیش از مرحله کامپایل، متن مشخص‌شده را جایگزین می‌کند.

بنابراین این دو روش از نظر عملکرد و نحوه پردازش یکسان نیستند.

۴ .قواعد نام‌گذاری متغیرها

برای انتخاب نام متغیرها باید قواعد زبان C را رعایت کنیم:

۱. نام متغیر باید با یک حرف انگلیسی یا زیرخط _شروع شود.

۲. نام متغیر نمی‌تواند با عدد شروع شود.

۳. استفاده از فاصله در نام متغیر مجاز نیست.

۴. حروف بزرگ و کوچک با یکدیگر تفاوت دارند؛ برای مثال ageو Ageدو نام متفاوت محسوب می‌شوند.

۵. کلمات رزروشده زبان C را نمی‌توان به‌عنوان نام متغیر استفاده کرد.

مثال‌هایی از نام‌های مجاز و غیرمجاز:

وضعیت	نام متغیر
مجاز	Age
مجاز	student_name
مجاز	number2
غیرمجاز	2number

غیرمجاز	student name
مجاز	Float

نکته: کلمه `float` در زبان C یک کلمه رزروشده است؛ اما `Float` به دلیل تفاوت حروف بزرگ و کوچک، نامی متفاوت محسوب می‌شود و می‌تواند به‌عنوان نام متغیر استفاده شود.

فصل اول ـ درس نهم: تبدیل نوع داده‌ها و خطاهای مقدماتی در C

بخش اول: تبدیل نوع داده (Type Conversion)

گاهی در برنامه‌نویسی لازم است مقدار یک متغیر را از یک نوع داده به نوع دیگری تبدیل کنیم؛ برای مثال، یک عدد صحیح را به عدد اعشاری تبدیل کنیم.

در زبان C دو روش اصلی برای تبدیل نوع داده وجود دارد:

۱. تبدیل ضمنی (Implicit Conversion)

در این روش، کامپایلر به‌صورت خودکار نوع داده را تبدیل می‌کند.

مثال:

```
int a = 10;
float b;

b = a;

printf("%f", b);
```

خروجی:

```
10.000000
```

در این مثال، مقدار صحیح متغیر `a` هنگام قرار گرفتن در متغیر `b` به نوع اعشاری تبدیل می‌شود.

۲. تبدیل صریح (Explicit Conversion)

در این روش، برنامه‌نویس خودش نوع داده موردنظر را مشخص می‌کند. به این روش **Casting** نیز گفته می‌شود.

ساختار کلی:

(data_type) expression

مثال:

```
int a = 15;
int b = 4;
float result;
```

result = (float)a / b;

```
printf("%f", result);
```

خروجی:

3.750000

در این مثال، با تبدیل aبه نوع float، تقسیم به‌صورت اعشاری انجام می‌شود.

اگر تبدیل صریح انجام نمی‌شد، تقسیم دو عدد صحیح انجام می‌گرفت و نتیجه آن ۳ بود؛ زیرا در تقسیم صحیح، بخش اعشاری نتیجه نگهداری نمی‌شود.

نکته مهم :تبدیل نوع داده همیشه بدون از دست رفتن اطلاعات نیست. برای مثال، هنگام تبدیل floatبه int، بخش اعشاری حذف می‌شود.

```
float x = 8.75;
int y = (int)x;
```

نتیجه:

y = 8

بخش دوم: خطاهای مقدماتی در برنامه‌نویسی

هنگام نوشتن و اجرای برنامه ممکن است خطاهای مختلفی رخ دهند. آشنایی با انواع خطا به برنامه‌نویس کمک می‌کند مشکلات برنامه را بهتر شناسایی و برطرف کند.

۱ .خطای نحوی(Syntax Error)

این خطا زمانی رخ می‌دهد که قواعد نوشتاری زبان C رعایت نشده باشند.

مثال اشتباه:

```
int a = 10
printf("%d", a);
```

در این مثال، علامت ; در پایان دستور تعریف متغیر فراموش شده است.

شکل صحیح:

```
int a = 10;
printf("%d", a);
```

۲ .خطای زمان اجرا(Runtime Error)

خطایی است که هنگام اجرای برنامه اتفاق می‌افتد.

برای مثال، دسترسی غیرمجاز به حافظه یا استفاده از اشاره‌گر نامعتبر می‌تواند باعث بروز خطای زمان اجرا شود.

۳ .خطای منطقی(Logical Error)

در این نوع خطا، برنامه ممکن است بدون خطای نحوی اجرا شود، اما نتیجه اشتباهی تولید کند.

مثال:

```
int a = 10;
 int b = 5;


int result = a - b;


printf("%d", result);
```

اگر هدف ما جمع دو عدد باشد، برنامه باید از عملگر +استفاده کند، نه -.

شکل صحیح:

```
int result = a + b;
```

بخش سوم: آشنایی با روش عیب‌یابی(Debugging)

Debuggingفرایند شناسایی، بررسی و اصلاح خطاهای برنامه است.

برنامه‌نویس برای عیب‌یابی می‌تواند از روش‌های زیر استفاده کند:

- کد را مرحله‌به‌مرحله بررسی کند.
- مقدار متغیرها را کنترل کند.
- پیام‌های خطا را بخواند.
- از ابزارهای عیب‌یابی (Debugger) استفاده کند.

تمرین هنرجویان

۱ .تبدیل صریح نوع داده با کدام روش انجام می‌شود؟

الف) Casting)

ب) Scanning)

ج) Printing)

۲ .اگر برنامه اجرا شود، اما جواب اشتباهی بدهد، با چه نوع خطایی روبه‌رو هستیم؟

الف) Syntax Error)

ب) Logical Error)

ج) Runtime Error)

۳ .هنگام تبدیل `float` به `int` معمولاً چه اتفاقی می‌افتد؟

الف) بخش اعشاری حذف می‌شود.

ب) دقت عدد افزایش می‌یابد.

ج) عدد به متن تبدیل می‌شود.

فصل اول ـ درس دهم: ساختار برنامه در C و اجرای نخستین پروژه

۱ .ساختار کلی یک برنامه در زبان C

هر برنامه در زبان C از بخش‌های مختلفی تشکیل می‌شود. برای درک بهتر، ابتدا یک برنامه ساده را بررسی می‌کنیم.

مثال: دریافت و نمایش یک عدد

```
#include <stdio.h>

int main()
{
    int number;

    printf("Enter a number: ");
```

```
    scanf("%d", &number);

    printf("Your number is: %d", number);

    return 0;
}
```

تحلیل بخش‌های برنامه

بخش	توضیح
#include <stdio.h>	اضافه کردن اعلان‌های توابع ورودی و خروجی استاندارد
int main()	تابع اصلی برنامه که اجرای برنامه از آن آغاز می‌شود
{ }	مشخص کردن محدوده دستورات تابع
int number;	تعریف یک متغیر از نوع عدد صحیح
printf()	نمایش پیام یا اطلاعات
scanf()	دریافت اطلاعات از ورودی
return 0;	پایان موفق اجرای تابع اصلی

۲. مراحل اجرای برنامه

برای تبدیل کد نوشته‌شده به یک برنامه قابل اجرا، معمولاً مراحل زیر انجام می‌شوند:

۱. نوشتن کد منبع(Source Code)

۲. پیش‌پردازش(Preprocessing)

۳. کامپایل و تولید کد شیء(Compilation)

۴. پیوند با کتابخانه‌ها(Linking)

۵. تولید فایل اجرایی(Executable File)

۶. اجرای برنامه(Execution)

این مراحل، نمای ساده‌شده‌ای از روند رایج ساخت و اجرای برنامه با ابزارهایی مانند GCC هستند.

۳. پروژه عملی: ماشین‌حساب ساده

در این پروژه، برنامه‌ای می‌نویسیم که دو عدد را از کاربر دریافت کند و چهار عمل اصلی ریاضی را روی آن‌ها انجام دهد:

- جمع
- تفریق
- ضرب
- تقسیم

کد برنامه

```
#include <stdio.h>

int main()
{
    float a, b;

    printf("Enter first number: ");
    scanf("%f", &a);

    printf("Enter second number: ");
    scanf("%f", &b);

    printf("Sum = %.2f\n", a + b);
    printf("Difference = %.2f\n", a - b);
    printf("Product = %.2f\n", a * b);

if (b != 0)
{
        printf("Division = %.2f\n", a / b);
}
else
```

```
    {
        printf("Division by zero is not allowed.\n");
    }


    return 0;
}
```

بررسی نمونه اجرای برنامه

اگر کاربر مقادیر زیر را وارد کند:

First number: 12

Second number: 4

خروجی برنامه به شکل زیر خواهد بود:

Sum = 16.00

Difference = 8.00

Product = 48.00

Division = 3.00

بررسی عملکرد برنامه

در ابتدای برنامه، دو متغیر `a` و `b` از نوع `float` تعریف شده‌اند تا بتوانند اعداد صحیح و اعشاری را نگهداری کنند.

سپس با استفاده از `printf` از کاربر درخواست می‌کنیم دو عدد وارد کند و با `scanf` مقادیر واردشده را دریافت می‌کنیم.

در ادامه، چهار عمل اصلی ریاضی انجام می‌شوند.

برای تقسیم، از دستور شرطی `if / else` استفاده کرده‌ایم تا پیش از انجام تقسیم، مقدار مخرج بررسی شود.

- اگر مقدار `b` صفر نباشد، تقسیم انجام می‌شود.
- اگر مقدار `b` صفر باشد، پیام مناسبی نمایش داده می‌شود و از تقسیم بر صفر جلوگیری می‌کنیم.

در این برنامه، عبارت `%.2f` باعث می‌شود عدد اعشاری با دو رقم بعد از ممیز نمایش داده شود و `\n` به معنی رفتن به خط بعد است.

۴ .مفاهیمی که در این پروژه استفاده کردیم

در این پروژه با مفاهیم زیر به‌صورت عملی کار کردیم:

- تعریف متغیرها
- نوع داده float
- دریافت ورودی با scanf
- نمایش خروجی با printf
- عملگرهای حسابی
- عملگر مقایسه‌ای =!
- دستور شرطی if / else
- بررسی تقسیم بر صفر

نکته آموزشی :دستور if / elseدر این پروژه برای نخستین بار به‌صورت کاربردی معرفی شده است. آموزش کامل ساختارهای شرطی در فصل بعد انجام می‌شود.

ارزیابی پایانی فصل اول

حالا ببینیم هنرجویان تا چه اندازه مفاهیم مقدماتی برنامه‌نویسی را یاد گرفته‌اند.

آزمون فصل اول

سؤال ۱ .کدام تابع برای دریافت ورودی استفاده می‌شود؟

الف) printf

ب) scanf

ج) main

سؤال ۲ .حاصل ۲۰ تقسیم بر ۴ چیست؟

الف) ۴

ب) ۵

ج) ۸

سؤال ۳ .کدام نوع داده برای نگهداری عدد اعشاری مناسب است؟

الف)int

ب)char

ج) float

سؤال ۴ .علامت =!چه مفهومی دارد؟

الف) مساوی

ب) نامساوی

ج) بزرگ‌تر

سؤال ۵ .هدف از `return 0` در تابع `main` چیست؟

الف) پایان موفق برنامه

ب) دریافت ورودی

ج) تعریف متغیر

ثبت و بررسی آزمون

پس از پاسخ‌گویی هنرجویان، پاسخ‌ها بررسی می‌شوند تا مشخص شود کدام مفاهیم به تمرین بیشتری نیاز دارند.

پایان فصل اول

این پایان نیست؛ آغاز توانستن است...

اگر تا اینجا همراه من آمده‌اید، شما دیگر همان کسی نیستید که در نخستین صفحه این فصل قدم به دنیای برنامه‌نویسی گذاشت.

اکنون با مفاهیم اولیه آشنا شده‌اید، منطق نوشتن دستورها را شناخته‌اید، نخستین الگوریتم‌ها را طراحی کرده‌اید و اولین برنامه‌های خود را نوشته‌اید.

شاید در نگاه اول، این دستاوردها کوچک به نظر برسند؛ اما هر مسیر بزرگی از قدم‌هایی آغاز می‌شود که روزی کوچک بوده‌اند.

به یاد داشته باشید، ارزش یک برنامه‌نویس به تعداد خط‌هایی که می‌نویسد نیست؛ به عمق تفکری است که پشت هر خط قرار دارد.

در این مسیر، همیشه پاسخ‌ها آماده نیستند. گاهی باید فکر کنید، آزمایش کنید، اشتباه کنید و دوباره بسازید. درست در همین لحظه‌هاست که از یک یادگیرنده به یک خالق تبدیل می‌شوید.

من باور دارم هر ذهن کنجکاوی می‌تواند چیزی فراتر از آنچه امروز می‌شناسد خلق کند؛ به شرط آنکه از پرسیدن، یادگرفتن و تلاش کردن دست نکشد.

پس به آنچه آموخته‌اید افتخار کنید، اما در همین نقطه متوقف نشوید. هنوز مسئله‌های بسیاری برای حل کردن، ایده‌های بسیاری برای جان بخشیدن و پروژه‌های بسیاری برای ساختن وجود دارند.

فصل اول به پایان رسید؛ اما داستان شما در دنیای فناوری تازه آغاز شده است.

باور داشته باشید که روزی به برنامه‌ای نگاه خواهید کرد که خودتان از ابتدا طراحی و ساخته‌اید و با لبخند به یاد خواهید آورد که همه‌چیز از همین اولین قدم شروع شد.

با احترام و امید به آینده‌ای که شما خواهید ساخت،

آرتیمیس قنواتی

NOVENTRA — Understand the Logic. Create the Future.