



محاسبات

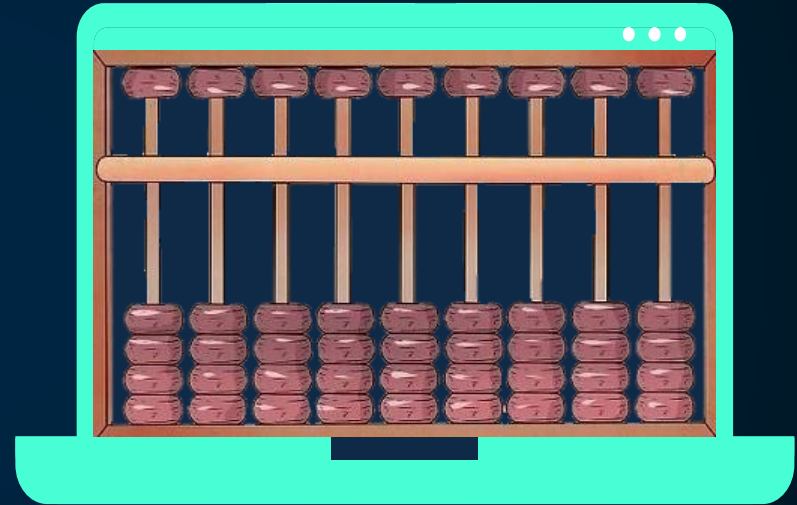
جلسه پنجم

بسم الله الرحمن الرحيم

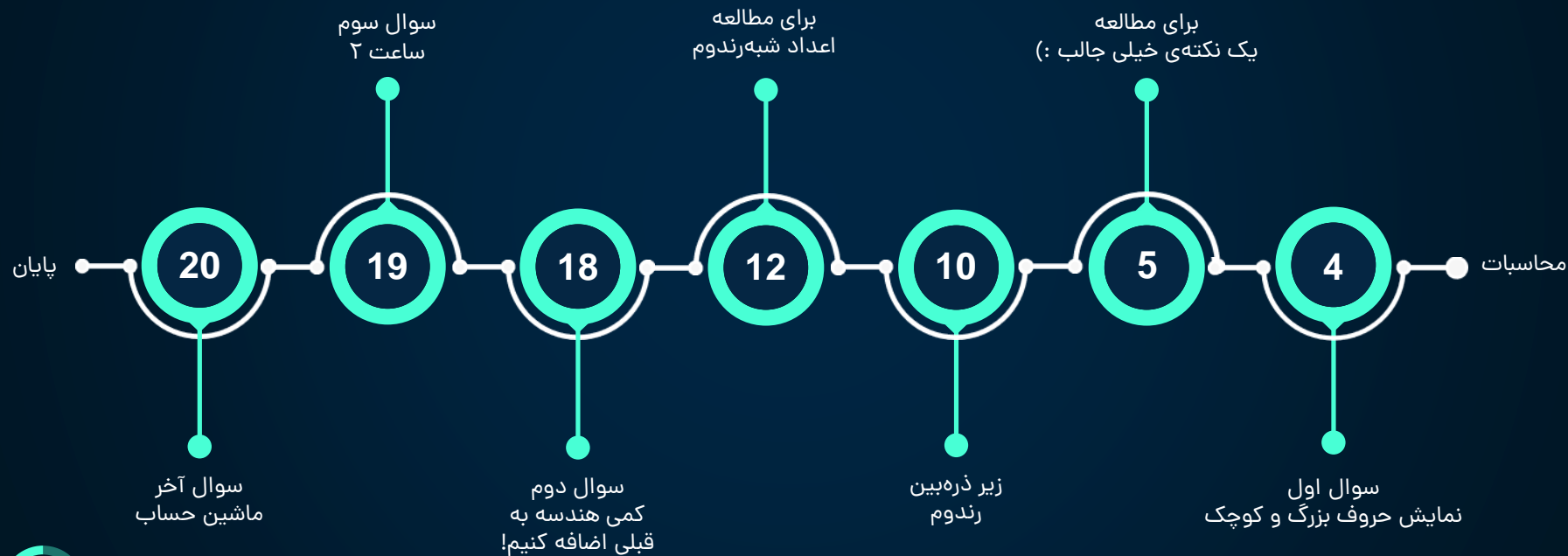


کارگاه مبانی برنامه نویسی - دانشکده مهندسی کامپیوتر دانشگاه هوایی شهید ستاری

در دنیای کامپیوتر، تکنولوژی و برنامه‌نویسی، اهمیت محاسبات و برنامه‌هایی که در این زمینه به کمک انسان‌ها می‌آیند کاملاً مشهود است. اگر کامپیوترها نبودند، انسان هرگز قادر به انجام برخی از این محاسبات نبود. بخش بزرگی از پیشرفتی که امروزه در تمام زمینه‌ها به دست آمده، مدیون همین محاسبات پیچیده‌ایست که با برنامه‌های کامپیوتری انجام می‌شود. به همین دلیل، این جلسه از کارگاه را به انجام محاسبات و گرفتن ورودی از کاربر و نشان دادن خروجی توسط برنامه اختصاص داده‌ایم.



فهرست



سوال اول: نمایش حروف بزرگ و



کوچک

این جمله را بارها و بارها شنیده‌ایم که کامپیوترها تنها با اعداد ۰ و ۱ کار می‌کنند. ۰ به معنای خاموش و ۱ به معنای روشن. در درس فهمیدیم که کامپیوتر برای تشخیص اعداد صحیح، اعشاری و غیره، باید هرکدام را به صورت ۰ و ۱ در بیاورد و شیوه‌های کد کردن انواع مختلف اعداد نیز با یکدیگر متفاوت است.




اما حروف را چطور باید کدگذاری کرد؟ آیا شیوه‌ی فهماندن حروف به کامپیوتر را به یاد دارید؟




یک نکته‌ی خیلی جالب (:

برای مطالعه

همان‌طور که می‌دانید حروف زبان انگلیسی ۲۶ تا هستند؛ پس ما به حداقل ۵ بیت برای نشان دادن این حروف نیاز داریم. در مثال‌ها هم اگر دقت کنید، ۵ بیت کم ارزش برای مشخص کردن این حروف به کار می‌رود:


 $A = (10\ 00001), B = (10\ 00010), C = (10\ 00011), \dots, Z = (10\ 11010)$
و دو بیت پر ارزش، کوچک یا بزرگ بودن حروف را مشخص خواهد کرد.
پس در کل ما ۷ بیت داریم.


 $A = (10\ 00001), a = (11\ 00001), B = (10\ 00010), b = (11\ 00010), \dots,$
 $Z = (10\ 11010), z = (11\ 11010)$

حرف A در سیستم ASCII برابر با چه برابر است؟ عدد را در مبنای ۲ به دست آورید.

$$65 = 64 + 1 = 2^6 + 1$$

$$= (???????)_2$$

حالا همین کار را برای حرف a انجام دهید...


$$97 = 64 + 32 + 1 =$$

$$2^6 + 2^5 + 1 = (???????)_2$$

یک نکته‌ی خیلی جالب (:

برای مطالعه (ادامه)

راه حل اولی که به ذهن همه می‌رسد چیست؟
افزایش تعداد بیت‌ها. در این صورت

$$A = (10 \dots 000 \dots 00001)$$


همان‌طور که می‌بینید تعداد بیت زیادی بی‌استفاده می‌ماند تا فقط یک کاراکتر نمایش داده شود.

از طرفی کاراکتری به نام null وجود دارد (که به زودی با آن آشنا خواهید شد) که پترن آن تماماً صفر است. با وجود این تعداد صفری که در کاراکترها می‌تواند به وجود بیاید، ممکن است بخشی از آن به اشتباه به عنوان null برداشت شود و مشکلات زیادی به وجود بیاید. برای رفع این مشکل راه دیگری پیشنهاد شد...

اما بقیه‌ی کاراکترها چه؟
کاراکترهای زبان فارسی،
زبان چینی، عربی، حتی
کاراکترهای ریاضی و
موسیقی و هزاران کاراکتر
دیگر. آیا برای تعریف این
کاراکترها، ۷ بیت که
حتی بخشی از آن هم
برای کاراکترهای انگلیسی
مصرف شده کافی است؟

یک نکته‌ی خیلی جالب: سیستم UTF-8

برای مطالعه (ادامه)

در صورتی که به تعداد کمی بیت نیاز داشته باشیم -یعنی حداکثر ۷ بیت (برابر تعداد x ها)- از این مدل کد استفاده می‌کنیم.

0xxxxxxx: 1 byte code

اگر به تعداد بیت بیش‌تری نیاز داشتیم -تا حداکثر ۱۱ بیت- از مدل دیگری استفاده می‌کنیم. در این مدل بایت اول (هشت بیت اول) با 110 شروع شده است. دو تا 1 یعنی کد ما دو بایت است. سپس دو بایت دوم با 10 شروع شده که معنای ادامه‌ی کاراکتر است. بنابراین ۳ بیت اول و ۲ بیت نهم و دهم برای در کد یک کاراکتر نقشی ندارند و فقط برای طولانی‌تر کردن فضای کد حضور دارند.

110xxxxx 10xxxxxx: 2 byte code

1110xxxx 10xxxxxx
10xxxxxx : 3 byte code,
16bits

11110xxx 10xxxxxx
10xxxxxx 10xxxxxx : 4
byte code, 21bits

...

✓ برای حل مسئله‌ی پیش‌رو، در صورت نیاز می‌توانید به جدول تبدیل کد اسکی^۱ به کاراکتر که در لینک زیر آمده‌است مراجعه کنید.

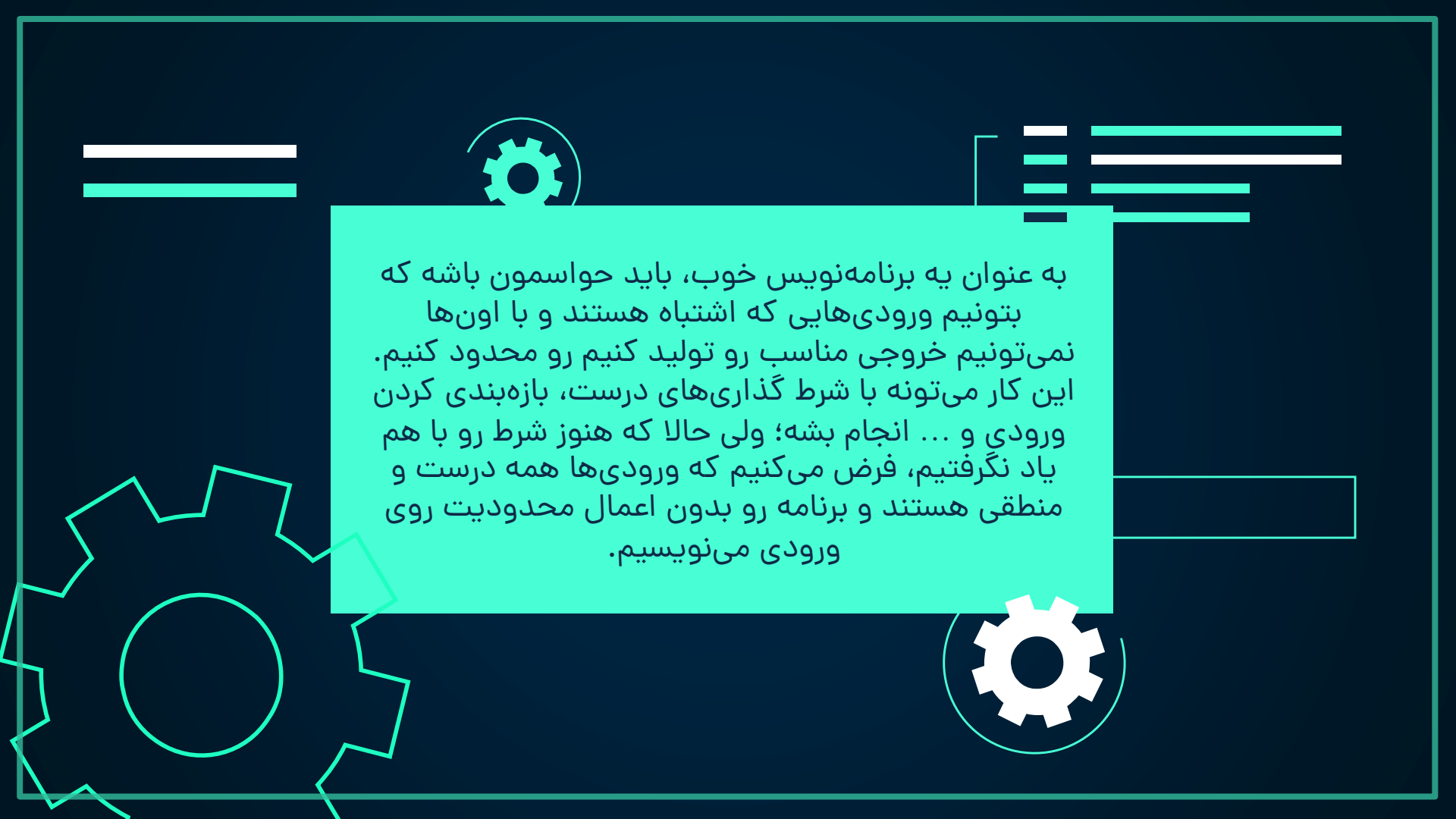


ASCII lookup table

با هم‌گروهی خود برنامه‌ای بنویسید که ابتدا در ورودی یک عدد و یک کاراکتر الفبایی دریافت کند.

سپس کاراکتر الفبایی را به اندازه‌ی عدد شیفت دهد. خروجی برنامه هر دو صورت حرف بزرگ و حرف کوچک آن کاراکتر خواهد بود.

✓ مثال: اگر در ورودی کاراکتر B و عدد +3 باشد، در خروجی E و e چاپ شود.



به عنوان یه برنامه‌نویس خوب، باید حواسمون باشه که بتونیم ورودی‌هایی که اشتباه هستند و با اون‌ها نمی‌تونیم خروجی مناسب رو تولید کنیم رو محدود کنیم. این کار می‌تونه با شرط گذاری‌های درست، بازه‌بندی کردن ورودی و ... انجام بشه؛ ولی حالا که هنوز شرط رو با هم یاد نگرفتیم، فرض می‌کنیم که ورودی‌ها همه درست و منطقی هستند و برنامه رو بدون اعمال محدودیت روی ورودی می‌نویسیم.

زیر ذره بین: رندوم

چرا اعداد رندوم مهم هستند؟ به نظر شما این اعداد چه کاربردی در دنیای کامپیوتر دارند؟ در برنامه نویسی، خیلی وقت ها ما می خواهیم مسائل دنیای واقعی را شبیه سازی کنیم که بسیاری از آن ها حداقل از دید ما تصادفی هستند.

مثلا شکل ابرها یا بُر زدن دسته ای کارت و نحوه ی قرار گرفتن آن ها. ✓

به علاوه، اعداد رندوم کاربرد زیادی در ساخت و پیاده سازی بازی ها دارند. طبیعتا شما دوست ندارید بازی ای را انجام دهید که در آن همیشه یک اتفاق می افتد و می خواهید هر بار در یک حالت مشابه اتفاقات متفاوتی رخ دهد، مثلا در پرتاب تاس در بازی مار و پله. بازی ریاضی جلسه ی قبل را به یاد دارید؟ اگر همیشه ترتیب کارت و شماره های بازی یکسان باشد، آیا ادامه ی بازی معنایی دارد؟

یکی دیگر از کاربردهای خیلی مهم اعداد رندوم، در آمار و احتمالات و حل و پیاده سازی مسائل مربوط به آن است که با آن در ترم های آینده به طور کامل آشنا خواهید شد. ✓



حال اعداد رندوم چگونه تولید می‌شوند؟ آیا واقعا کامپیوتر قدرت این‌که اعدادی **کاملا**

تصادفی تولید کند را دارد؟



جواب این پرسش خیر است!



اعداد تصادفی‌ای که کامپیوتر تولید می‌کند، به هیچ‌وجه کاملاً تصادفی نیستند و کامپیوتر تنها بر اساس الگوریتم‌های تولید عدد رندومی که دارای قطعیت هستند این اعداد را تولید می‌کند.



یعنی این اعداد با کمک برخی توابع ریاضی از پیش تعریف شده تولید می‌شوند و نه به شکل تصادفی. همین موضوع باعث می‌شود که اگر ما برخی قواعد تولید عدد رندوم را رعایت نکنیم و یا به نفع خودمان تغییر دهیم، اعداد تولید شده حتی در ظاهر هم تصادفی نباشند و از یک الگوی مشخص پیروی کنند.

اعداد شبه‌رندوم یا pseudo random number

برای مطالعه

در ساخت کامپیوترهای امروزی، سعی شده تا هیچ نویزی از بیرون کامپیوتر اعداد ۰ و ۱ درون کامپیوتر را تغییر ندهد، پس در حالت ایده‌آل، هیچ عددی که توسط یک کامپیوتر تولید شده دچار عدم قطعیت نمی‌شود (یعنی مثلاً جای برخی ۰ و ۱ ها عوض نمی‌شوند).

به همین علت، برای تولید اعداد رندوم در کامپیوتر از توابعی به نام **توابع شبه‌رندوم** استفاده می‌شود. یک تابع شبه‌رندوم خیلی ساده، می‌تواند تابعی باشد که در خود دو المان نگه می‌دارد.

- ✓ یکی تعداد بارهایی که اجرا شده (در این‌جا، n) است.
- ✓ دیگری یک سری نامتناهی از اعداد از پیش تعیین‌شده (در این‌جا، S).

از دبیرستان **سری‌های** حسابی و هندسی را به خاطر دارید. یک سری اعداد شبه‌رندوم، یعنی **سری اعدادی** که در نگاه اول، کاملاً رندوم و بدون هیچ الگوی خاصی به نظر می‌آیند، اما کاملاً قطعی هستند و پروسه‌ی تولید آن‌ها کاملاً تکرارپذیر است.


اعداد شبه‌رندوم یا pseudo random number

برای مطالعه (ادامه)

این تابع با هربار اجرا شدن، یک واحد به عدد n خود اضافه می‌کند و سپس n امین عدد موجود در S را به عنوان خروجی برمی‌گرداند. این تابع شبه‌رندوم است، چراکه با وجود بی‌ارتباط بودن ظاهری خروجی‌های آن نسبت به هم، اگر بتوانیم بعد از چندبار اجرای آن، n را به حالت اولیه‌ی خود برگردانیم، باز با همان سری قبلی مواجه خواهیم شد (که در اعدادی که واقعا رندوم باشند چنین اتفاقی نخواهد افتاد).

لازم به ذکر است که با استفاده از برخی سایت‌ها در اینترنت، می‌توان اعداد رندوم واقعی تولید کرد که این اعداد از وقایع طبیعی استخراج می‌شوند (مثلا سایت [random.org](https://www.random.org) این اعداد را با استفاده از میزان نویزهای موجود از اتمسفر زمین استخراج می‌کند).

برای دیدن لیستی از توابع شبه‌رندوم به لینک زیر مراجعه کنید.

 <https://b2n.ir/739146>

 <https://www.random.org/>

تا الان متوجه شدیم که اعداد تولید شده به هیچ عنوان تصادفی نیستند و به کمک برخی توابع و فرمول‌های ریاضی تولید می‌شوند. حال لازم است تا با مفهومی به نام **seed** آشنا شوید. می‌توان گفت **seed** نقطه‌ی شروع تابع تولید عدد رندوم است. بدیهی است در صورتی که این مقدار تغییر نکند، تابع همواره رشته‌ای از اعداد ثابت تولید می‌کند و دیگر تصادفی نیست. برای مثال، تکه کد زیر را اجرا کنید.

```
for(int i = 0; i < 5; i++)  
    printf("%d", rand());  
return 0;
```

می‌بینید که در هر بار اجرای این شبه کد، ۵ عدد ثابت تولید می‌شوند. دلیل آن هم واضح است، ما هیچ‌گاه **seed** را تغییر ندادیم. برای تغییر **seed** چه باید کرد؟



در این مرحله تابع `srand` به کمک ما می‌آید. این تابع خروجی ندارد و کاربرد آن تنها این است که مقدار `seed` را برای تابع `rand` مشخص می‌کند. به این شکل که ورودی آن یک عدد صحیح است که به عنوان `seed` تابع `rand` انتخاب می‌شود. (در حالت قبلی که از این تابع استفاده نکردیم مقدار `seed` به شکل پیش‌فرض ۱ قرار گرفت).



برای استفاده از این تابع هم چالش‌هایی وجود دارد، زیرا ما نیاز داریم برای این که عددمان هر بار تصادفی باشد، مقدار `seed` در هر بار اجرا تفاوت کند و ثابت نباشد. به نظر شما چگونه می‌توان این مشکل را حل کرد؟



چه چیزی را می‌توان به عنوان هسته یا `seed` رندوم انتخاب کرد که مدام در حال تغییر باشد و خروجی ثابت ایجاد نکند؟

یکی از راه‌حل‌های اصلی این است که از تابع **time** استفاده کنیم، زیرا این تابع هربار بر اساس زمان سیستم عدد مختلفی را به عنوان خروجی برمی‌گرداند.

برای مثال، کد زیر را اجرا کنید.

```
srand(time(0));  
for(int i = 0; i < 5; i++)  
    printf("%d", rand());
```

این بار، برخلاف دفعه‌ی قبل بعد از هر بار اجرای برنامه، خروجی متفاوتی را مشاهده می‌کنیم که دلیل آن تغییر مقدار **seed** است.

توجه: به خاطر داشته باشید که برای درست کار کردن تابع رندوم، باید تنها یک بار به آن **seed** اختصاص دهید. در غیر این صورت نمی‌توان تضمین کرد که برای عدد رندوم تولید شده، احتمال آمدن عدد x با عدد y برابر باشد.



اگر دوست دارید در مورد تابع `time` و خروجی‌ش تحقیق کنید. به نتایج جالبی می‌رسین :) این تابع مقدار ثانیه‌های سپری شده از تاریخ ۱ ژانویه ۱۹۷۰ رو برمی‌گردونه (که به این تاریخ می‌گن Unix epoch). پس با هر بار اجرای تابع `rand`، مقدار متفاوتی نسبت به دفعه‌ی قبل به دست میاریم که قابل پیش‌بینی نیست.

برای غیرقابل پیش‌بینی‌تر کردن این مقدار، می‌تونیم از کارهای جالب و خلاقانه‌ی دیگه‌ای هم استفاده کنیم؛ مثلاً خروجی تابع تایم رو به جای ثانیه به میلی‌ثانیه تبدیل کنیم. یه مثال دیگه، برنامه‌هایی هستن که بر اساس حرکت موس کاربر یا حرکت دادن موبایل و... می‌تونن `seed` بسازن و با کمک اون عدد رندوم تولید کنن.



سوال دوم: کمی هندسه به قبلی اضافه کنیم!



فرض کنید مثلثی دارید که دو ضلع و زاویه‌ی بین آن مشخص است، حال می‌خواهید با استفاده از اطلاعات فعلی، اندازه‌ی ضلع دیگر مثلث را حساب کنید. برای محاسبه‌ی ضلع سوم، احتمالا از دبیرستان به‌خاطر دارید که می‌توان از قانون کسینوس‌ها استفاده نمود. (اگر یادتان نیست با گوگل احساس راحتی کنید!)




شرط مسئله این است که برنامه قرار نیست اندازه‌ی اضلاع و زاویه‌ی بین را دریافت کند و باید هر ۳ را با کمک عدد رندوم تولید کنید! به این شکل که زاویه عددی بین ۰ تا ۹۰ و اضلاع هم هر کدام بین ۵ تا ۲۵ باشند. یعنی کد شما هم صورت سوال را طراحی می‌کند و هم پاسخ آن را می‌یابد!




امتیازی: پس از محاسبه‌ی ضلع سوم، به کمک قانون سینوس‌ها اندازه‌ی زاویه‌ی کوچک‌تر را به دست بیاورید (فکر می‌کنید چرا زاویه‌ی کوچک‌تر؟ چرا هر ۲ زاویه یا حتی زاویه‌ی بزرگ‌تر نه؟! و در نهایت اندازه‌ی زاویه‌ی آخر را به دست آورید.

سوال سوم: ساعت ۲

مشابه جلسه‌ی قبل، سوالی مربوط به ساعت اما با رویکردی متفاوت را می‌خواهیم بررسی کنیم.

این بار از شما می‌خواهیم برنامه‌ای بنویسید که با دریافت تعداد ثانیه‌ها به صورت یک عدد طبیعی، محاسبه کند عدد داده شده معادل چند روز، چند ساعت، چند دقیقه و چند ثانیه است. عدد 9876543210 را وارد کنید و ببینید آیا برنامه به درستی کار می‌کند؟ 

این برنامه را با شبه‌کدی که در جلسه‌ی قبل نوشتید مقایسه کنید. آیا محاسبات، کار را در مقایسه با دقیقه به دقیقه پیش‌رفتن در زمان ساده‌تر نکرد؟ 

و اما سوال آخر: ماشین حساب

سوال آخر مثل همیشه اختصاص دارد به گُدخدا و Botfather.

این دو کدر^۱ بزرگ در یک دوره‌ی آموزشی از شرکت‌کننده‌ها خواستند تا یک ماشین حساب با قابلیت محاسبه‌ی چهار عمل اصلی و برخی توابع مثلثاتی را پیاده‌سازی کنند. آن‌ها نمونه کدهای جالبی دریافت کرده‌اند و از آنجایی که معتقدند مطالعه‌ی کدهای دیگران و خطایابی یکی از تمرین‌های موثر برای یادگیری برنامه‌نویسی است، یک نمونه از کدهای دارای ایراد را برای تمرین در این دستورکار فرستاده‌اند و از شما می‌خواهند خطاهای موجود در کد را رفع کنید تا برنامه به درستی کار کند. آیا می‌توانید نوع هر یک از خطاها را مشخص کنید؟



برای رفع کردن اشکالات من سعی می‌کنم راهنمایی‌هایی بکنم تا راحت‌تر بتوانید آن‌ها را پیدا کنید. از Botfather هم دعوت می‌کنم که در بخش توضیحات هر کدام از ایرادات در کنار من باشد. بیا بید با هم قدم به قدم سعی کنیم تا ایرادات موجود در کد رفع شوند. قبل از هر کاری باید بفهمیم که این ماشین حساب چه قابلیت‌هایی قرار است داشته باشد و بعد شروع کنیم به تست کردن برنامه. ممکن است در ابتدا اصلاً کد کامپایل نشود.

آیا این اتفاق برای کد شما هم افتاد؟ پس باید سعی کنیم تا مشکل به وجود آمده را رفع کنیم. به نظر شما چه چیزی این مشکل را به وجود آورده؟

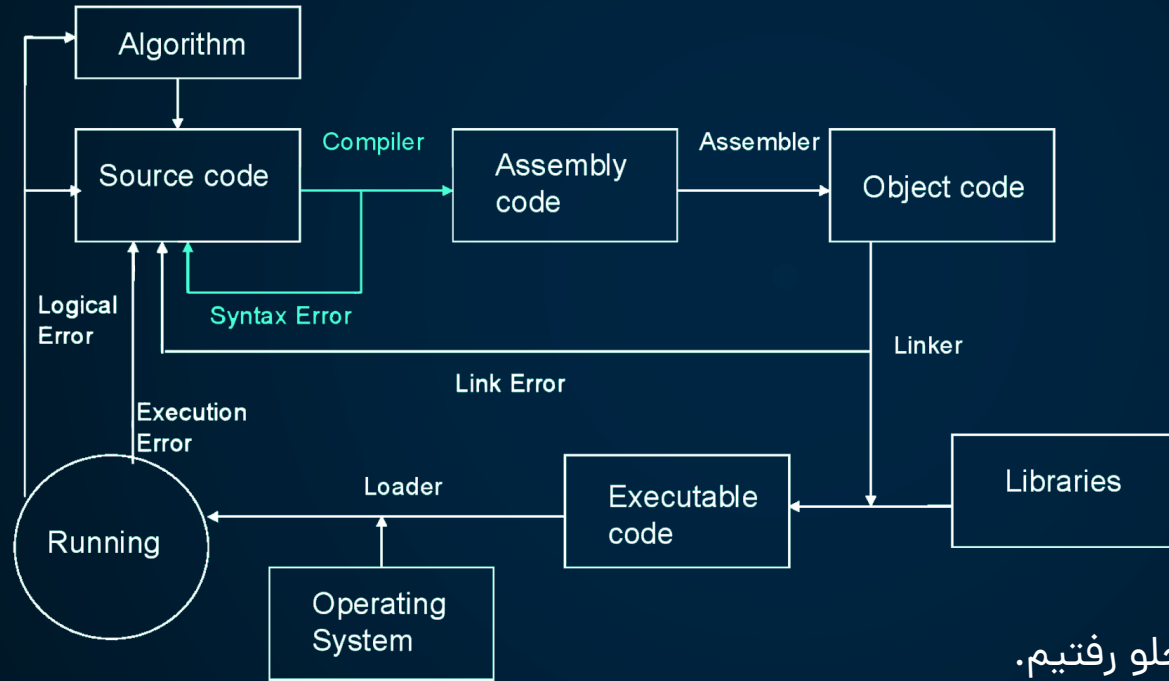


این خطا قبل از اجرا شدن برنامه رخ داده است، به همین دلیل به آن **خطای زمان کامپایل**^۱ یا **خطای نوشتاری**^۲ می‌گویند.

هر گونه خطا در نوشتن برنامه مثل نبودن " ; " یا دقت نکردن به کوچک و بزرگ نوشتن حروف در زبان C می‌تواند منجر به این خطا شود. چون کامپایلر برای تبدیل کد نوشته شده به یک برنامه‌ی اجرایی، ابتدا خط به خط کد را چک می‌کند و فایل اجرایی را می‌سازد.



برای رسیدن به برنامه‌ای که هدف نهایی ماست، لازم است برای رفع کردن تمامی خطاها روندی داشته باشیم تا ایرادی باقی نماند. برای مثال می‌توانیم از چنین مسیری استفاده کنیم.



پس تا اینجا یک مرحله جلو رفتیم.





احتمالا برخی از شما Main را هم درست کردید و به این موضوع که C تفاوت بین حروف بزرگ و کوچک را متوجه می‌شود دقت کردید. اما نوشتن Main به جای main یک خطای کامپایل نیست و ما در این شرایط با **Linker error** مواجه می‌شویم. علت رخ دادن این ارور را در آینده در مبحثی به نام تابع متوجه خواهید شد.



حال با درست شدن برخی از ایرادها برنامه کامپایل می‌شود اما با این اخطار مواجه می‌شویم:

warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]

دلیل این اتفاق چیست؟



به عنوان راهنمایی می‌پرسم. آیا می‌دانید stdio مخفف چه چیزی است؟



Header files "stdio.h" and "stdlib.h" in C

در صورت اضافه نکردن کتابخانه‌های مورد نیاز، با ارور یا خطا مواجه می‌شویم تا زمانی که آن را برطرف کنیم.

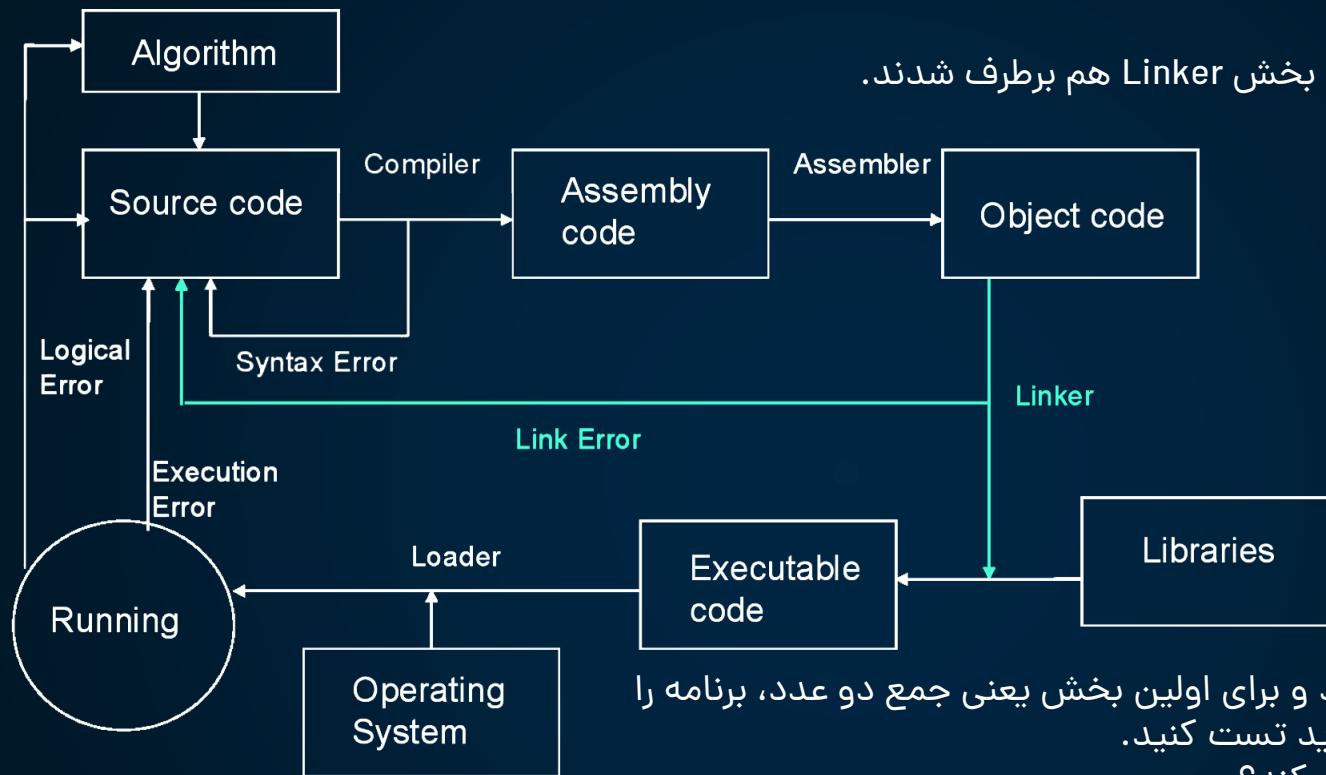
اگر کامپایلر C بتواند تابع را به صورت **ضمنی** یا **implicit** تعریف کند در این صورت با ارور مواجه نمی‌شویم.



برای عمیق‌تر شدن در این موضوع می‌توانید بیشتر تحقیق کنید. به عنوان مثال می‌توانید به لینک زیر مراجعه کنید و اگر دوست داشتید درباره‌ی این موضوع در کلاس تدریس‌یاری بیشتر بحث کنید.



Why printf and scanf work without stdio.h?



خب خطاهای مربوط به بخش Linker هم برطرف شدند.



حال برنامه را اجرا کنید و برای اولین بخش یعنی جمع دو عدد، برنامه را هر جور که دوست دارید تست کنید. آیا برنامه درست کار می‌کند؟

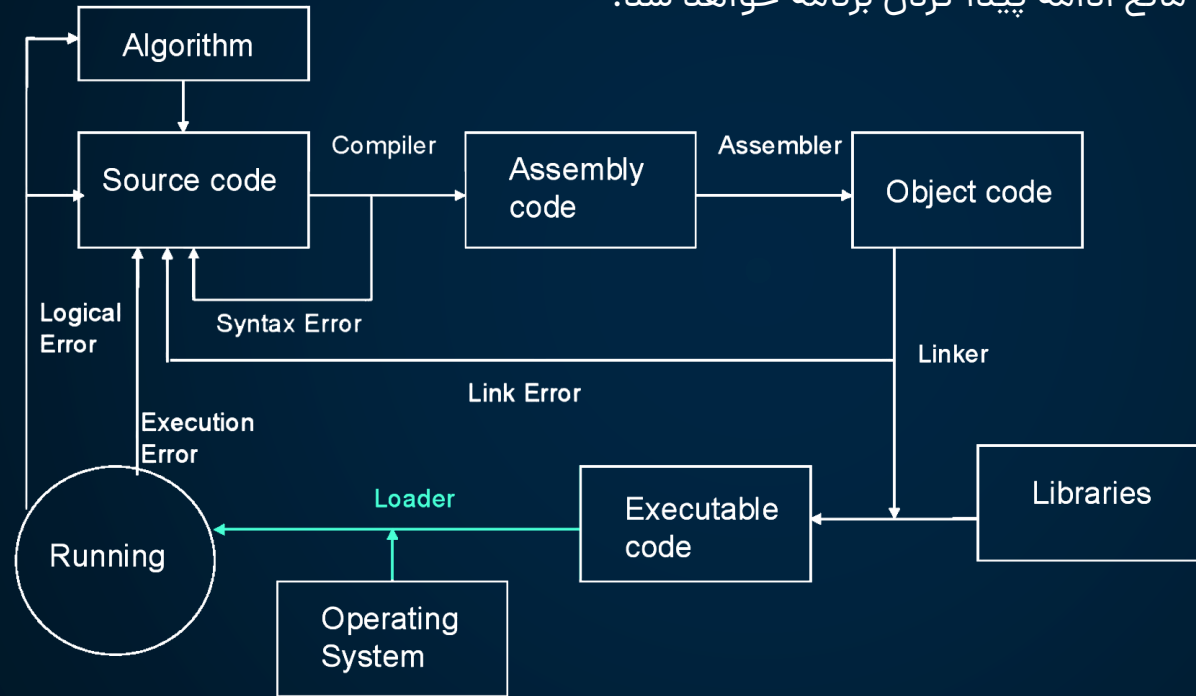


این خطا که مانع ادامه پیدا کردن برنامه شده است، **execution error** یا **runtime error** نام دارد چون در زمان اجرا شدن رخ داده است. این بار علت این خطا از کجاست؟





به یاد داشته باشید که قرار ندادن & در scanf یکی از معروفترین خطاها در زمان اجرای برنامه است. در این زمان برنامه‌ی شما به خانه‌ای از حافظه دسترسی پیدا کرده که سیستم عامل اجازه‌ی دسترسی‌اش را به برنامه نمی‌دهد و به همین دلیل مانع ادامه پیدا کردن برنامه خواهد شد.





بالاخره می‌توانیم خود برنامه را تست کنیم که آیا ماشین حساب درست کار می‌کند یا خیر؟
برای شروع خیلی راحت از اولین عملگر یعنی $+$ آغاز می‌کنیم.
آیا حاصل $2 + 3$ درست است؟
حاصل $2.5 + 6.3$ چطور؟



این خطای آخر، یک **خطای منطقی**^۱ است؛ یعنی خطایی است که تنها به دست برنامه‌نویس قابل تشخیص است و به کاربردهایی که او در نظر دارد وابسته است. تنها راه تشخیص چنین خطاهایی هم تست کردن مکرر برنامه به ازای ورودی‌های مختلف است.



می‌توانید برای تسلط بیشتر روی خطاها به هر کدام از لینک‌های زیر مراجعه کنید و اگر دوست دارید در کلاس تدریس‌یاری راجع به آن‌ها با تدریس‌یارها صحبت کنید تا بهتر بتوانید از وقوع این ارورها جلوگیری کنید و یا علت ارورها را متوجه شوید.



لینک‌های مشابه لینک‌های زیر با نشان دادن تفاوت‌های بین ارورها می‌توانند در یادگیری و عمیق‌تر شدن کمک کنند.



[Programming Errors in C](#)



[Compile errors VS Runtime errors](#)

;