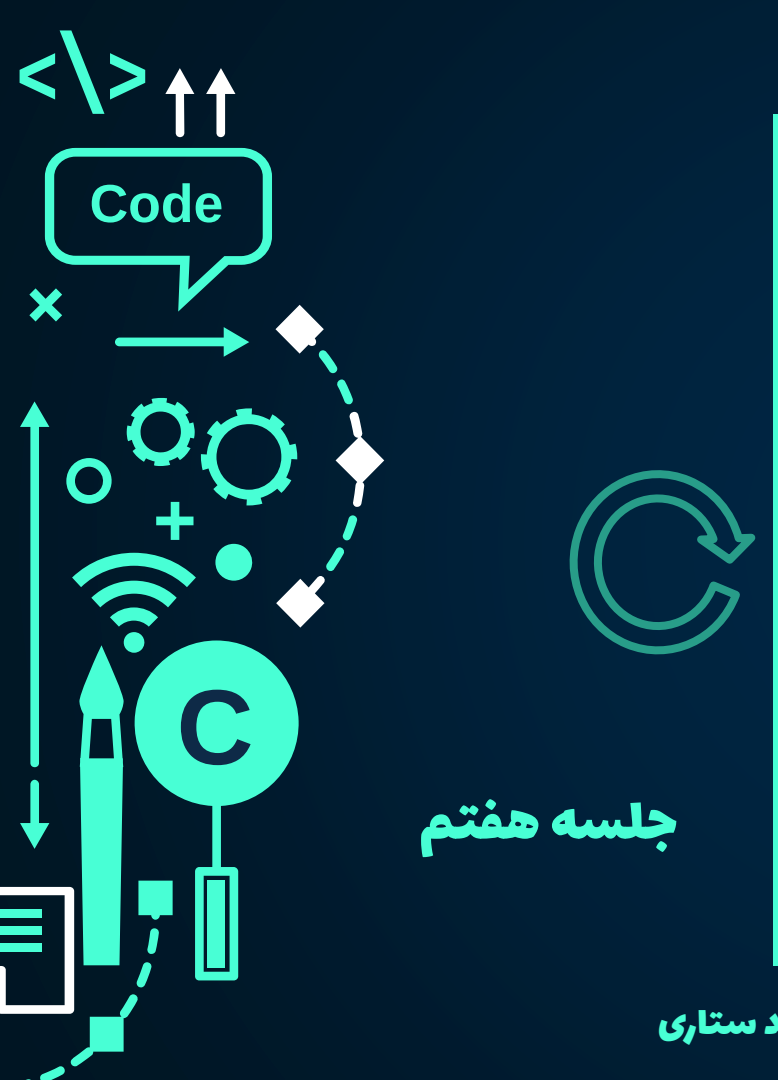


بسم الله الرحمن الرحیم

حلقه‌ها

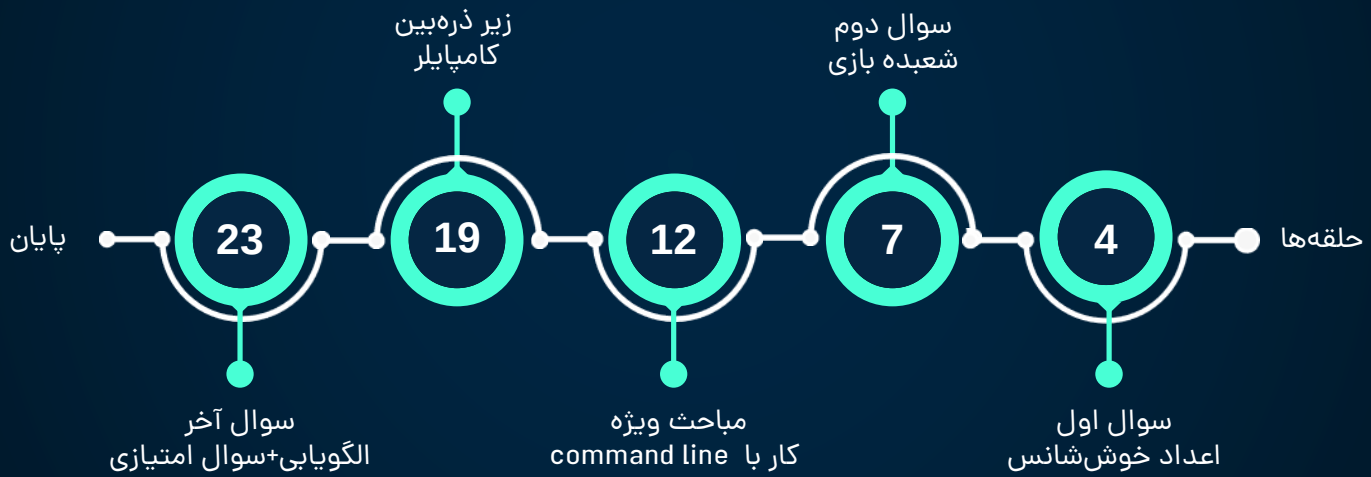
جلسه هفتم

آیا شما از انجام دادن یک کار تکراری به تعداد زیاااد لذت می‌برید؟

احتمالا تعداد زیادی از ما از کارهای تکراری خوشمان نمی‌آید، اما می‌دانیم که می‌توانیم از این کارهای تکرارشونده در حل مشکلات زیادی استفاده کنیم. برخی از پدیده‌هایی که اطرافمان می‌بینیم، همیشه در حال تکرار شدن هستند. علاوه بر آن‌ها، تکرارها را در الگوریتم‌های زیادی نیز می‌توانیم ببینیم.

مثلا برای به دست آوردن ب.م.م دو عدد یا مرتب کردن تعدادی از اعداد، باید کارهای تکرار شونده‌ای انجام دهیم که می‌توانیم برای انجام‌شان از قالب حلقه‌ها استفاده کنیم. حلقه‌ها می‌توانند قالبی برای چند خط کد باشند که قرار است به تعداد مشخصی، تکرار شوند. در این صورت، یک کار را چند بار انجام می‌دهیم، اما دستورالعمل مربوط به آن را فقط یک بار می‌نویسیم. به همین دلیل، کارگاه این هفته به مبحث مهم حلقه‌ها اختصاص یافته‌است.

فهرست


حلقه‌ها
4
سوال اول
اعداد خوش‌شانس
7
سوال دوم
شعبده بازی
12
مباحث ویژه
کار با command line
19
زیر ذره‌بین
کامپایلر
23
سوال آخر
الگویابی+سوال امتیازی
پایان

سوال اول: اعداد خوش‌شانس

بیاید اول یک مثال حل شده را با هم ببینیم (به لینک صفحه‌ی بعدی مراجعه کنید).

اگر اعدادی که تنها از عوامل اول 2، 3 و 5 تشکیل شده‌اند را اعداد زشت بدانیم، می‌خواهیم برنامه‌ای بنویسیم که در تشخیص زشت بودن یا نبودن یک عدد به ما کمک کند.

برای پیاده‌سازی چنین سوالی، باید تمام مقسوم‌علیه‌های یک عدد بررسی شوند تا ببینیم آیا برابر با یکی از اعداد 2، 3 یا 5 هست؟ هر گاه یکی از این مقسوم‌علیه‌ها چنین شرطی نداشت، متوجه می‌شویم که عدد ما زشت نیست.

پس باید کاری را به صورت تکراری انجام دهیم که شرط پایان آن، رسیدن به یک مقسوم‌علیه دیگر است یا بررسی شدن تمام مقسوم علیه‌ها.

برای انجام این کار، راه‌های مختلفی پیش‌رو داریم. برای مثال، در کد زیر ابتدا عدد مورد نظر تا جایی که بر 2 بخش‌پذیر است بر 2 تقسیم شده. سپس همین روند برای 3 و 5 هم اتفاق افتاده. حالا که دیگر عدد ما بر هیچ کدام از این ارقام بخش‌پذیر نیست، اگر حاصل 1 باشد یعنی عدد ما زشت بوده و اگر 1 نباشد یعنی مقسوم‌علیه‌های اول دیگری هم داشته و نمی‌تواند یک عدد زشت باشد.

```
while (n % 2 == 0)
    n /= 2;
while (n % 3 == 0)
    n /= 3;
while (n % 5 == 0)
    n /= 5;

if (n == 1)
    printf("The number is ugly");
else
    printf("The number is not ugly");
```

اگر خواستید می‌توانید کد روش روبه‌رو و همچنین روش بازگشتی این سوال را در ریپازیتوری زیر و در پوشه‌ی مربوط به کارگاه هفتم بیابید.

CE102-C-Lab

در ادامه، سوالی تقریبا مشابه مثال قبل را می‌بینید که Numfather آن را برای کارگاه طرح کرده.

Numfather به دلیل علاقه‌ی شدیدی که به اعداد 4 و 7 دارد، تمام عددهایی که ارقامشان فقط از این دو رقم تشکیل شده باشد را خوش‌شانس می‌نامد. مثلا عدد 47، 744 یا 4 به نظر او اعداد خوش‌شانسی بوده‌اند که تمام رقم‌هایشان 4 و 7 است اما عدد 17، 467 یا 6 این شانس را نداشته‌اند.

او می‌خواهد تخفیفی برای اعداد قائل شود و آن‌ها را به عنوان همسایه‌ی اعداد خوش‌شانس حساب کند و تشخیص این اعداد را به عهده‌ی ما گذاشته است. طبق گفته‌ی او، ما می‌توانیم اعدادی را همسایه‌ی اعداد خوش‌شانس بنامیم که "جمع تعداد 4 و 7های آن‌ها یک عدد خوش‌شانس باشد". مثلا عدد 44647 همسایه‌ی اعداد خوش‌شانس است، زیرا سه رقم 4 و یک رقم 7 دارد که روی هم می‌شود چهار رقم و عدد 4 هم یک عدد خوش‌شانس است.

بنابراین، باید برنامه‌ای نوشته شود که با دریافت ورودی n تشخیص دهد که آیا این عدد یک همسایه برای اعداد خوش‌شانس هست یا خیر.

سوال دوم: شعبده بازی

در حیطه‌ی کامپیوتر و زیرشاخه‌های آن، قضیه‌ای به نام No Free Lunch می‌گوید که نمی‌توان بدون از دست دادن قابلیتی، قابلیت دیگری را به دست آورد. این قضیه فراتر از مسائلی مانند مصرف انرژی برای انجام محاسبات بر روی یک کامپیوتر است.

برای فهم بهتر آن، برنامه‌ای بنویسید که با دریافت عدد n از کاربر، مقدار جمع زیر را یک‌بار از راست به چپ و یک‌بار از چپ به راست محاسبه کند:

$$F(n) = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{2^n}$$

این برنامه را به ازای nهای مختلف اجرا کنید. آیا دو مقدار محاسبه‌شده یکسان هستند؟ در مورد علت آن با مدرس کارگاه خود گفت‌وگو نمایید. کوچک‌ترین مقدار nای که به ازای آن این دو جمع با هم مساوی نیستند را بیابید.

در ادامه می‌توانید علت کامل آن را بخوانید. اما در آینده، در درس معماری کامپیوتر به طور کامل‌تری علت این اتفاق را درک خواهید کرد.

مساوی نبودن برخی از این جمع‌ها می‌تواند دو علت داشته‌باشد، چراکه ممکن است کامپیوترها از دو روش متفاوت برای ذخیره‌سازی اعداد در کامپیوتر استفاده کنند.

روش اول همانند ذخیره‌سازی اعداد ده‌دهی است، اما در مبنای دو. برای اعداد صحیح، این محاسبه را به خاطر داریم:

	Col 8	Col 7	Col 6	Col 5	Col 4	Col 3	Col 2	Col 1
Base^{exp}	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Weight	128	64	32	16	8	4	2	1
	2*2*2*2*2*2*2*2	2*2*2*2*2*2*2	2*2*2*2*2*2	2*2*2*2*2	2*2*2*2	2*2*2	2*2	2

حال فرض کنید می‌خواهیم همین محاسبات را برای اعداد اعشاری حساب کنیم. چون قسمت اعشاری اعداد کمتر از ۱ هستند و توان کمتر از ۰ دارند، بدین صورت آن‌ها را ذخیره می‌کنیم:

	Col 1	Col 2	Col 3	Col 4	…
$Base^{exp}$	2^{-1}	2^{-2}	2^{-3}	2^{-4}	…
Weight	0.5	0.25	0.125	0.0625	…

پس مثلا عدد 2.25 در مبنای ده، تبدیل می‌شود به عدد 10.01 در مبنای دو.

حالا، چون مکان ذخیره سازی ما برای هر عدد محدود است (مثلا به هر عدد تنها هشت بیت اختصاص دهیم)، پس نمی‌توان این اعداد را با دقت بی‌نهایت ذخیره کرد.

(به عنوان یک مثال ملموس‌تر، می‌توانید به عدد π فکر کنید؛ چون اعشار این عدد بی‌پایان هستند، ما در هنگام کار با آن، اعشارش را به تعداد محدودی رقم تقریب می‌زنیم.)

فرض کنید عدد 10.0000564 را می‌خواهیم به مبنای دو ببریم، اما صرفا 6 بیت برای ذخیره‌سازی این عدد داریم. چون قسمت صحیح آن ارزش بیش‌تری دارد، سعی می‌کنیم تمامی آن را ذخیره کنیم. این باعث می‌شود تعداد بیت کافی برای ذخیره‌کردن قسمت اعشاری عددمان نداشته‌باشیم، پس تنها می‌توانیم آن را به صورت 1010.00 (باینری) تقریب بزنیم که قسمت اعشاری را کاملا از دست می‌دهیم. روش‌های تقریب‌زدن متفاوتی برای ذخیره‌کردن این نوع اعداد اعشاری وجود دارد و هنگامی که کامپیوتر در حال محاسبه‌ی این جمع از چپ و از راست است، این تقریب را در مراحل متفاوتی انجام می‌دهد که منجر به نتیجه‌های متفاوتی می‌شود.

پس همان‌طور که دیدیم، There is no free lunch! یا مجبوریم تمامی حافظه‌ی کامپیوتر را برای ذخیره کردن اعداد به صورت کاملا دقیق اختصاص دهیم، یا می‌توانیم از دقت محاسبات‌مان کم کرده و حافظه‌ی کامپیوتر را برای موارد دیگری مصرف کنیم.

- روش اول بیش‌تر جنبه‌ی آموزشی داشت. در روش دوم که اکثرا توسط کامپیوترهای امروزی استفاده می‌شود، اعداد اعشاری به صورت نماد علمی، یعنی $a * 10^b$ ذخیره می‌شوند. مثلا عدد 0.000002 را می‌توان به صورت 2×10^{-6} ذخیره کرد و کامپیوترها نیز همین کار را (اما با مبنای دو) انجام می‌دهند.

- با وارد شدن اعداد بزرگ‌تر به جمع در هنگام جمع از سمت راست، کامپیوتر در برخی زمان‌ها تصمیم می‌گیرد که توان این نماد علمی را یک واحد افزایش دهد که باعث از دست دادن دقت می‌شود و در هنگام جمع از سمت چپ، چون از همان ابتدا داریم با توان بزرگ اعداد را محاسبه می‌کنیم، ممکن است بعضی اعداد اصلا در جمع حساب نشوند (تفاوت این دو این است که هنگام جمع از سمت راست، جمع دو عدد ممکن است نتیجه‌ی بزرگ‌تری بدهد که در هنگام بالا رفتن عدد توان، از دست نرود ولی در حالت دوم از همان ابتدا جمع‌شان حساب نخواهد شد).

مباحث ویژه: کار با command line

- در این جلسه قصد داریم مروری برای مباحثی ویژه خارج از آنچه در کلاس آموخته‌اید داشته باشیم. در اولین قسمت قصد داریم از gcc برای کامپایل کردن برنامه به صورت دستی استفاده کنیم. همانطور که در درس آموخته‌اید، عملیات کامپایل کردن از مراحل زیر تشکیل شده است:
  ۱. پیش‌پردازش
  ۲. کامپایل کردن
  ۳. اسمبلر
  ۴. لینکر
- این عملیات‌ها غالبا به صورت یک جا توسط یک برنامه (کامپایلر) صورت می‌گیرند. یکی از کامپایلرهای موجود gcc می‌باشد که ما از آن استفاده خواهیم کرد. برای شروع با دستور زیر وارد command prompt شوید:

```
windows + R
cmd
```

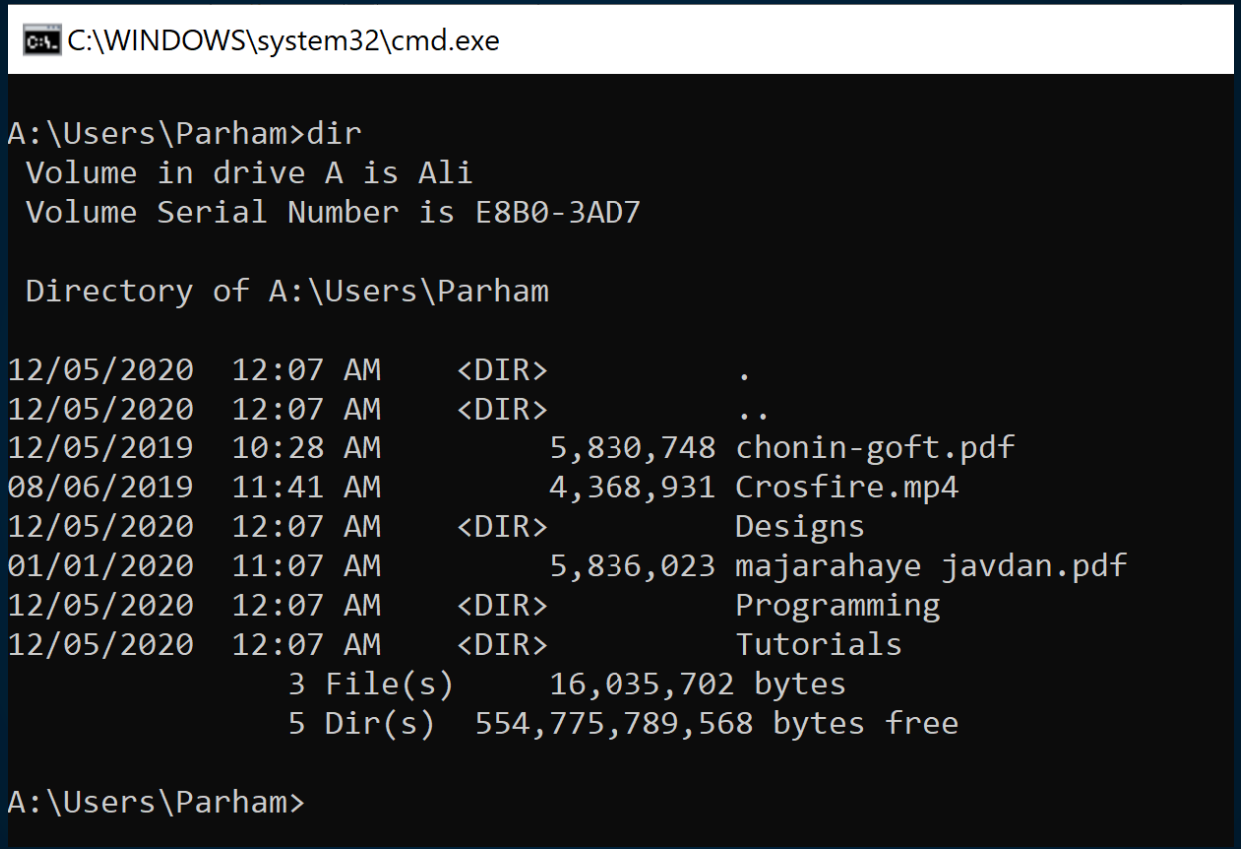
در محیط command prompt شما می‌توانید تمام آنچه را که در محیط گرافیکی انجام می‌دهید، انجام دهید. یکی از ویژگی‌های این محیط، قابلیت‌های اجرایی آن است که از محیط گرافیکی بسیار بیش‌تر است. در ابتدا اندکی با این محیط بیش‌تر آشنا می‌شویم.
همانند محیط گرافیکی، command prompt هم در یک پوشه از سیستم شما اجرا می‌شود.

پوشه‌ای که در آن قرار دارید را می‌توانید در خط فرمان ببینید:

```
C:\Users\parham>
```

با دستور زیر می‌توانید محتوای این پوشه را ببینید:

```
dir
```

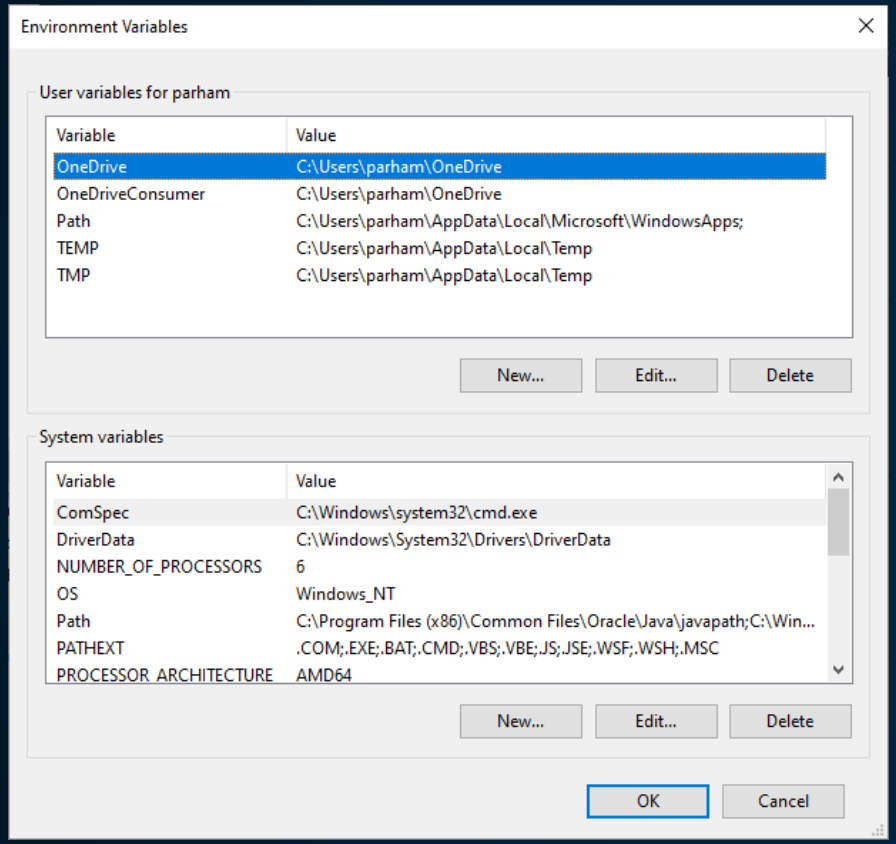

C:\WINDOWS\system32\cmd.exe
A:\Users\Parham>dir
Volume in drive A is Ali
Volume Serial Number is E8B0-3AD7
Directory of A:\Users\Parham
12/05/2020 12:07 AM <DIR> .
12/05/2020 12:07 AM <DIR> ..
12/05/2019 10:28 AM 5,830,748 chonin-goft.pdf
08/06/2019 11:41 AM 4,368,931 Crosfire.mp4
12/05/2020 12:07 AM <DIR> Designs
01/01/2020 11:07 AM 5,836,023 majarahaye javdan.pdf
12/05/2020 12:07 AM <DIR> Programming
12/05/2020 12:07 AM <DIR> Tutorials
3 File(s) 16,035,702 bytes
5 Dir(s) 554,775,789,568 bytes free
A:\Users\Parham>

- با دستور زیر می‌توانید به پوشه‌ی دیگری منتقل شوید:

```
cd Downloads
```

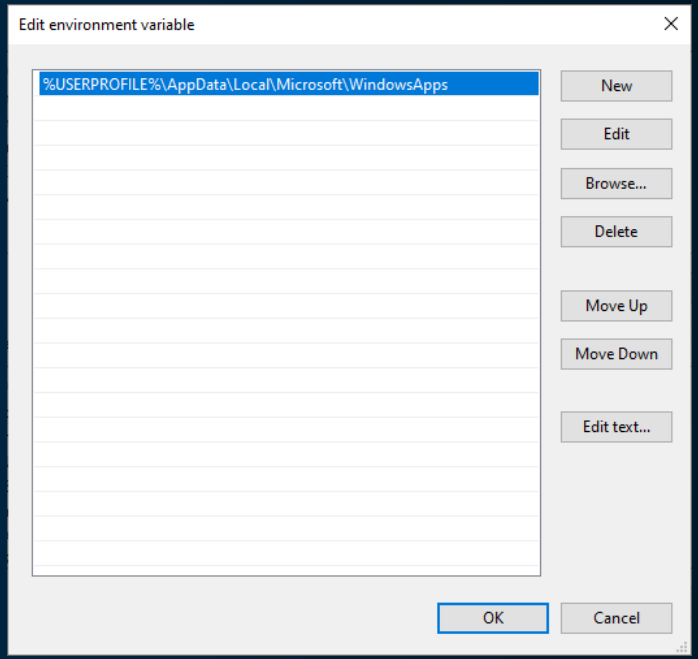
```
C:\Users\parham\Downloads>
```

- در خط فرمان، دستوراتی که فراخوانی می‌شوند در واقع برنامه‌هایی‌اند که روی سیستم شما قابل دسترسی هستند. برای هر دستور، مسیرهای مشخصی در سیستم شما جست‌وجو می‌شود تا برنامه‌ی مورد نظر شما پیدا شود.



این مسیرها در پنجره‌ی روبه‌رو قابل تنظیم هستند:

در صورتی که می‌خواهیداز gcc در محیط خط فرمان استفاده کنید، می‌بایست آدرس محل نصب آن را به این متغیر (Path) اضافه کنید.



یک برنامه ساده را در فایل hello.c می‌نویسیم:

```c
#include <stdio.h>
    int main() {

    int n;

    scanf("%d", &n);
    printf("Hello World %d\n", n);
}
```

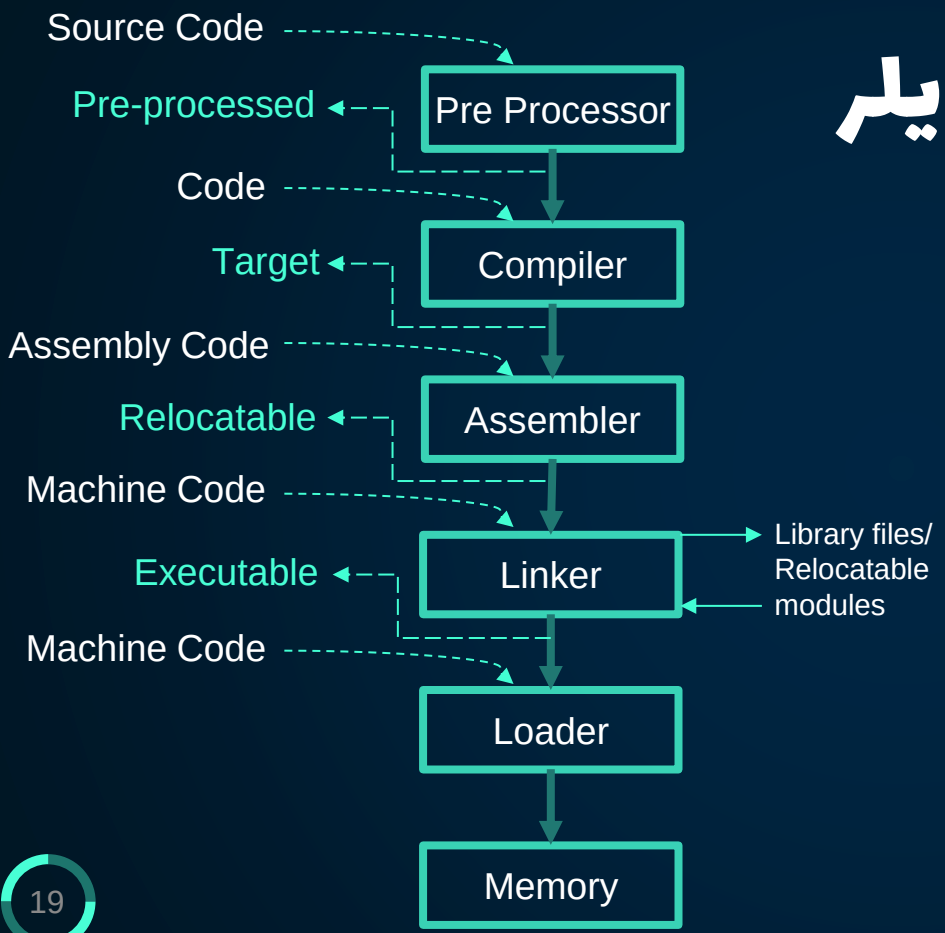
حال فرض کنید که می‌خواهیم فایل hello.c که در پوشه فعلی قرار دارد را کامپایل کنیم:

```
gcc -o hello.exe hello.c
```

در ادامه برنامه را اجرا می‌کنیم:

```
hello.exe
```

زیر ذره‌بین: کامپایلر



در اکثر مواقع، کامپایل و اجرای برنامه با زدن یک دکمه در IDEها انجام می‌شود. اما بیاید کمی دقیق‌تر به این پروسه نگاه کنیم. به طور کلی، مراحلی که از زدن کد تا اجرای آن روی CPU طی می‌شود، این‌ها هستند:

- به طور خلاصه، وظیفه‌ی pre-processor یا پیش‌پردازنده، انجام دادن مراحل اولیه مثل حذف کامنت‌ها، یکی کردن فایل‌های کد و تبدیل خط‌هایی از کد که با # شروع می‌شود (ماکروها[1]) به رهنمودهایی برای مراحل بعد کامپایل (در زبان C) است. مثلا با دیدن خط

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

- فایل stdio.h را در کد ما include می‌کند و این هدر[2] مانند یک قسمت عادی در کنار کد ما قرار می‌گیرد تا به همراه هم کامپایل شوند.
- با اضافه کردن فلگ[3] E- هنگام فراخوانی دستور gcc می‌توان این مرحله را به صورت دقیق‌تر دید:

```
gcc -E main.c -o main.i
```


1- Macro
2- Header
3- Flag

- در مرحله‌ی بعد، compiler با دریافت یک فایل واحد از مرحله‌ی قبل، کدی به زبان assembly تولید می‌کند که می‌توانید آن را به صورت دقیق با اضافه‌کردن فلگ -S به gcc، هنگام کامپایل ببینید:

```
gcc -S main.i -o main.s
```

- در مرحله‌ی بعد، assembler کد assembly تولید شده را به machine code تبدیل می‌کند.
Machine code کدی‌ست که با توجه به معماری پردازنده‌ی کامپیوتر تولید می‌شود و در اکثر اوقات روی سایر پردازنده‌ها قابل اجرا نیست. برای مثال، در اکثر اوقات نمی‌توان کدی که توسط کامپیوتری با پردازنده Intel تولید شده را بر روی کامپیوتری با پردازنده AMD اجرا کرد.
- این کدها در فایل‌هایی با پسوند .o یا .obj به معنای object code ذخیره می‌شوند و با اضافه کردن فلگ -c هنگام فراخوانی دستور gcc می‌توان این مرحله را به صورت دقیق‌تر دید:

```
gcc -c main.c -o main.o
```

- در مرحله‌ی بعد، linker تمامی object code های تولید شده (چه از سمت کد کاربر، چه از سمت سیستم‌عامل) را به هم متصل می‌کند و یک فایل نهایی برای انتقال بر روی پردازنده تولید می‌کند. به طور مثال، تا قبل از اجرای این مرحله، دستور printf معنای خاصی برای برنامه ندارد، اما بعد از وصل شدن فایل‌های سیستم‌عامل به کد ما در این مرحله، دستور printf معنای نوشتن در خروجی تهیه شده توسط سیستم‌عامل را پیدا می‌کند.

- و در نهایت، loader که جزوی از سیستم‌عامل است، حجم برنامه‌ی ساخته شده را آنالیز می‌کند، بر روی مموری (RAM) کامپیوتر حافظه‌ای برای آن تخصیص می‌دهد و آن را در صف اجرا توسط پردازنده قرار می‌دهد.

- توضیحات بیش‌تر درباره‌ی کامپایلر و چگونگی ارتباط آن با سیستم‌عامل را در درس‌های «اصول طراحی کامپایلر» و «سیستم‌عامل‌ها» خواهید خواند.

و اما سوال آخر: الگویابی

سلام بچه‌ها. امیدواریم حالتون خوب باشه. امروز می‌خوایم یکم راجع به الگویابی صحبت کنیم.

می‌دونیم که توی برنامه‌نویسی، نظم و ترتیب‌ها برای ما خیلی اهمیت داره. اصلا یکی از دلایل به وجود اومدن حلقه‌ها این بوده که بعد از دقت به اطراف دیدیم که خیلی از اتفاقات به صورت تکرارشونده‌ای اتفاق می‌افتن. پس برای ما خیلی اهمیت داره که با تمرین‌هایی ذهنمون رو فعال کنیم و به عنوان برنامه‌نویس دید متفاوتی به اطراف و اتفاقات و مسائل داشته باشیم. یکی از این تمرین‌ها پیدا کردن الگوریتم چاپ الگوها و پترن‌های مختلف است.

برای چاپ هر یک از این پترن‌ها، با کد باید اول هر کدوم رو طبق مختصات R^2 بررسی کنیم و الگوی تکرار شونده‌ی اجزای اون الگو رو بدست آوریم (این اجزا می‌تونن فاصله، ستاره، اعداد و یا هر کاراکتر تشکیل دهنده‌ی الگو باشن) بعد با کمک حلقه‌ها، این الگوی تکرار شونده رو پیاده سازی کنیم.

شکل‌های روبرو رو ببینین:

شکل ۱:

```
*
**
***
****
*****
```

شکل ۲:

```
A
B B
C C C
D D D D
E E E E E
```

پیاده‌سازی الگوی شماره ۱ به شکل روبه‌رو هست:

فایل کد روبه‌رو رو می‌تونید توی ریپازیتوری کارگاه ببینین.

```c
#include <stdio.h>
int main(){
    int i, j, rows;
    printf("Enter the number of rows:");
    scanf("%d", &rows);
    for(i = 1; i <= rows; ++i){
        for(j = 1; j <= i; ++j){
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```

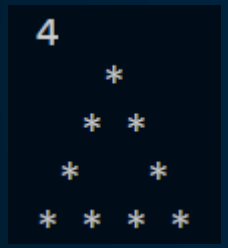
حالا بعد گرم کردن ذهن‌ها، الگوی زیر رو شما پیاده سازی کنین :)

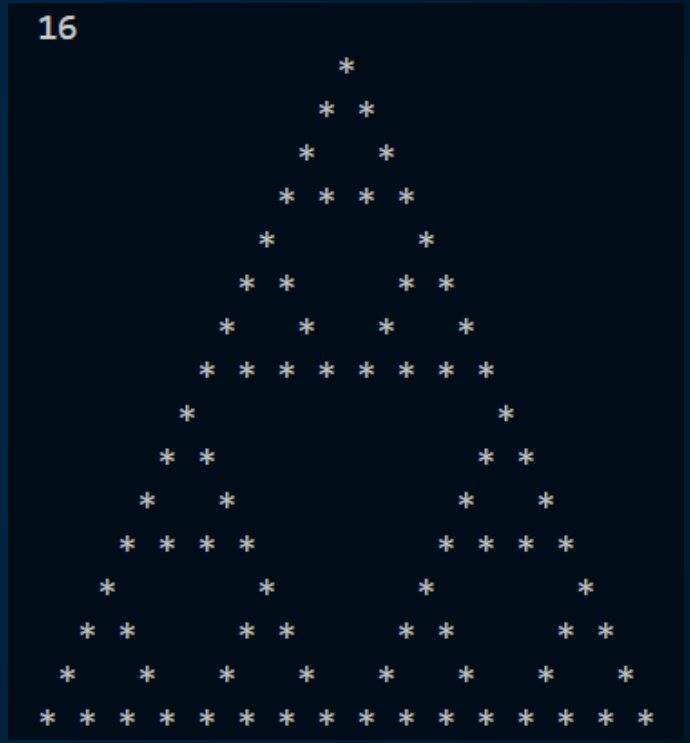
```
1          1
12        21
123      321
1234    4321
123454321
```

بعد از پیاده‌سازی شکل بالا، اگر دوست داشتید خودتون رو بیش‌تر به چالش بکشید، می‌تونید شکل بعدی که به مثلث سرپینسکی معروف است را الگویابی کنید.

(اگه به ساختن موجوداتی مثل مثلث سرپیسنکی علاقه‌مند شدید، بهتون توصیه می‌کنیم درباره‌ی مبحث فراکتال‌ها بیش‌تر مطالعه کنید.)

امتیازی: مثلث سرپینسکی





;