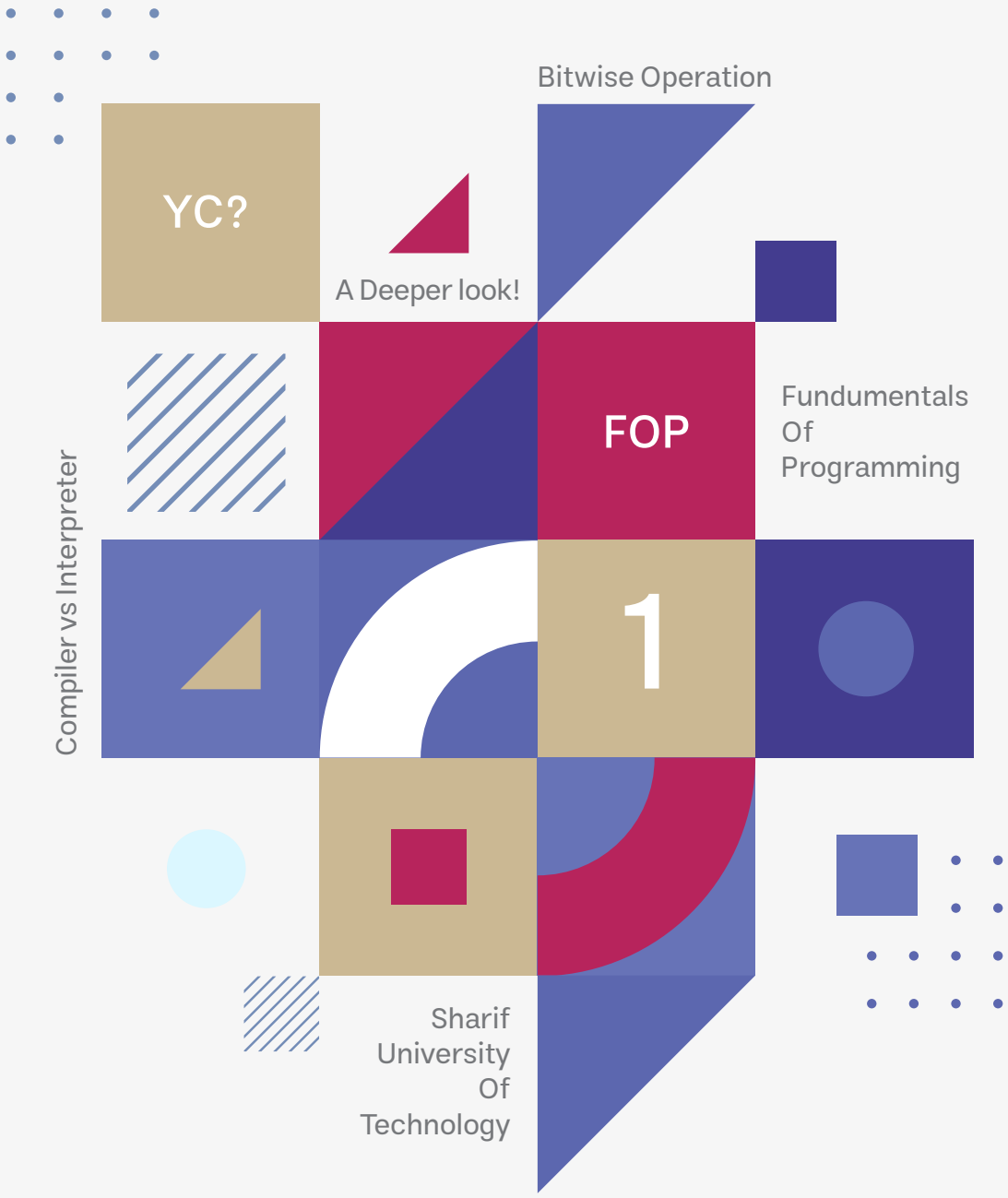
کدنامه

مبانی برنامه‌نویسی مهندسی کامپیوتر دانشگاه صنعتی شریف



YC?

بررسی علل انتخاب زبان C برای آموزش در درس مبانی برنامه‌سازی

یکی از دغدغه‌های دانشجوهای ورودی اینه که چرا از بین این همه زبان‌های مختلف، دانشکده زبان C رو برای تدریس توی درس مبانی انتخاب کرده. شاید حتی بعضیا از اینکه یه زبان قدیمی انتخاب شده عصبانی باشن! توی این بخش میخوایم این انتخاب رو بررسی کنیم و یه مقدار شما رو با دلیل این انتخاب آشنا کنیم.

محمدپارسا بشری

سی زنده‌ست ...

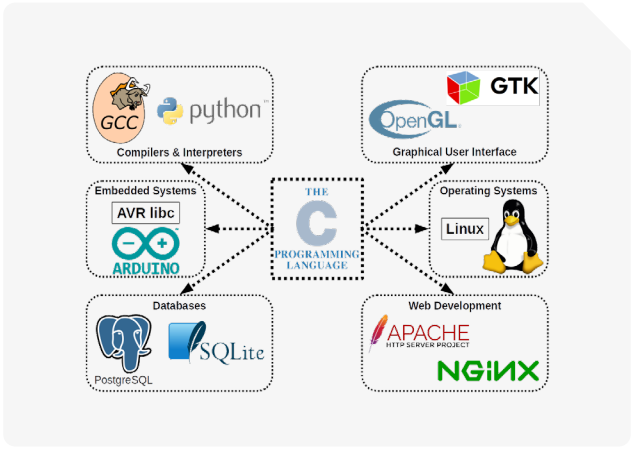
فرض کنین امروز از خواب بلند میشین و ساعت دیجیتالتون که داره زنگ میزنه رو خاموش می‌کنین. عملکرد دکمه‌های این ساعت به احتمال زیاد با زبان C نوشته شده! بعدش چون حوصله نون خریدن ندارین، یه تیکه نون یخزده رو از توی فریزر درمیارین و با مایکروویو گرمش میکنین. توی مایکروویو هم احتمالا کدی به زبان C داره اجرا میشه. بعدش سوار آسانسور میشین و درسته! اونم با C کار میکنه.

همه این دیوایس‌هایی که گفته شد (که بهشون Embedded Systems میگن) فقط یکی از جاهایی هست که زبان C توشون کاربرد داره. خب به ادامه روزمون برسیم ...

وقتی از آسانسور پیاده شدین و از خونه بیرون رفتین، سوار تاکسی میشین و احتمالا گوشی‌تون رو درمیارین. و بعدش وقتی که به دانشگاه رسیدین، لپتاپ رو کف لابی باز میکنین و شروع به کار میکنین. و خب همونطور که شاید تا الان حدس زده باشین، C توی اینا هم کاربرد داره. درواقع هسته (کرنل) اکثر سیستم عامل‌ها یا کاملا با C نوشته شدن یا اینکه بخش بزرگیشون به این زبانه. حتی همین الان میتونین کد سیستم‌عامل‌های متن باز رو ببینین (مثلا کد کرنل لینوکس رو اینجا میتونین ببینین)

همونطوری که توی لابی نشستین، تصمیم میگیرین که با بچه‌ها یه دست فیفا بزنین. به دلیل اینکه C و C++ به شدت سریع هستن، توی بازی‌سازی و شبیه‌سازی‌های سه‌بعدی (مثل سیستم خودران ماشین‌ها) خیلی کاربرد دارن به طوریکه اکثر بازی‌های بزرگ (که با یه سرچ ساده لیستشون درمیاد) با C++ نوشته شدن (اگه دوست داشتین بیشتر در مورد بازی‌سازی با C و C++ بدونین، عبارت Unreal Engine 5 رو توی گوگل سرچ کنین!).

سی مفیده ...



حالا که کاربردهای C رو دیدین، شاید بگین: «خب باشه قبول کردیم که C خیلی پرکاربرده، ولی اصلا چرا باید با زبان C شروع کرد؟». اول از همه باید بدونین که زبان C توی خیلی از دانشگاه‌های دنیا به عنوان اولین زبان به دانشجوهای مهندسی و علوم کامپیوتر آموزش داده میشه و برای این کار دلایلی وجود داره که در ادامه بهشون می‌پردازیم:

زبان C شما رو به یک Problem Solver تبدیل میکنه!

برنامه‌نویسی به زبان C قدرت حل مسئله شما رو زیاد میکنه. حالا چرا؟ توی اکثر زبان‌های برنامه‌نویسی یه سری رابط به صورت پیشفرض (توی کتابخونه‌های استاندارد اون زبان) وجود دارن که کار رو برای برنامه نویس راحت میکنن ولی توی C تعداد این رابط‌ها خیلی کمتر از بقیه زبان‌هاست و پیاده‌سازی منطق اونها از صفر به عهده خود برنامه نویسه. بنابراین زبان C پره از این جور مسئله‌های کوچیک که با حل کردنشون طرز فکر کردن حل مسئله‌ای رو یاد میگیرین و حسابی قوی میشین!

زبان C شما رو با واقعیت رو به رو میکنه!

به طور کلی زبان C یکی از زبان‌های نزدیک به سخت‌افزار بعد از اسمبلی (Assembly) محسوب میشه به طوری که بعضیا میگن «C is a portable assembly». این باعث میشه که حین یاد گرفتن زبان C با مفاهیم اصلی سخت‌افزار هم آشنا بشین و یه جورایی یه درکی از اتفاقی که در پشت صحنه در حال رخ دادنه پیدا کنین. درحالیکه توی زبان‌های سطح بالاتر، این مفاهیم از برنامه‌نویس پنهان میشن و به عهده خود زبان گذاشته میشن (تا کار برنامه‌نویس راحت‌تر باشه) و همین باعث میشه که به عنوان برنامه‌نویس C، وقتی بعدا با زبان‌های سطح بالاتر کار می‌کنیم، درک بهتری از اتفاقاتی که بقیه برنامه‌نویسا ازشون خبر ندارن داشته باشیم!

زبان C شما رو با Memory Management آشنا میکنه!

مدیریت حافظه یعنی اشغال کردن فضایی در حافظه، استفاده از اون فضا و آزاد کردنش وقتی کارمون باهاش تموم شد. در واقعیت، موقع کد زدن به زبان C باید همیشه حواسمون به حافظه‌ای که باهاش کار میکنیم باشه چون C برخلاف بقیه زبان‌های سطح بالا، آزاد کردن فضاهایی که مورد استفاده نیستن رو به عهده برنامه نویس گذاشته و خودش کاری نمیکنه (کاری که در زبان‌های دیگه به عهده garbage collector هست). بنابراین بعد از مدتی کار کردن با C، برنامه‌هایی که می‌نویسیم از لحاظ میزان حافظه مصرفی و نحوه memory management بهینه میشن.

زبان C راه رو برای یادگیری زبان‌های جدید باز میکنه!

بعضیا به زبان C میگن The mother of all languages و این به خاطر اینه که تعداد خوبی از زبان‌هایی که بعد از C ساخته شدن، یه جورایی از C گرفته شدن. به بیان دقیق‌تر، اکثر کامپایلرها و مفسرهای زبان‌های مختلف، با زبان C نوشته شدن. به طور مثال مفسر پایتون (به نام «CPython») و کامپایلر جاوا به زبان C نوشته شدن (البته کامپایلر جاوا در ابتدا با C بوده ولی الان با خود جاوا نوشته شده درحالیکه JRE (که ترم بعد باهاش آشنا میشین) همچنان با C نوشته شده). همین باعث میشه که با درک کردن برنامه‌نویسی با زبان C، یادگیری زبان‌های بعدی (که اکثرا مبتنی بر C هستن) راحت‌تر بشه.

دلایل زیاد دیگه‌ای برای یادگیری برنامه‌نویسی با زبان C وجود داره (مثل اینکه توی درس‌های ترم‌های بعدتون بهش نیاز پیدا می‌کنین یا برای شرکت توی مسابقات برنامه‌نویسی به دلیل سرعت بالاش یکی از گزینه‌های خوبه و ...) اما فکر میکنم که به اندازه کافی صحبت کردیم و امیدوارم شما هم الان نسبت به قبل از خوندن این متن، از اینکه C میخونین خوشحال‌تر باشین :)

COMPILER VS INTERPRETER

زبان سطح بالا به زبان‌هایی گفته می‌شه که به زبان ما انسان‌ها نزدیکه و درکش برای ما آسونه و در عوض برای کامپیوتر سخته ولی زبان سطح پایین برعکس برای ما سخته؛ ولی برای کامپیوتر آسونه!

در ادامه می‌خوایم به بررسی هر کدوم بپردازیم

안녕 얘들아? 잘 지내고 있나요?

فهمیدین که چی نوشتم؟ اگه آره، چجوری فهمیدین؟ احتمالا رفتین از یه برنامه‌ای که رابط زبان کره‌ای و فارسی باشه استفاده کردین تا بتونین ترجمش کنین. توی برنامه‌نویسی هم همین‌طوره. ینی وقتی شما یه کدی رو می‌نویسین، نیازه تا یه مترجمی باشه که کد شما رو به کامپیوتری که فقط ۰ و ۱ می‌فهمه، بفهمونه! این کار وظیفه کامپایلر(Compiler) و مفسر(Interpreter) هستش. حالا به نظرتون چقدر اینا با هم فرق دارن؟ کدوم بهتره ؟

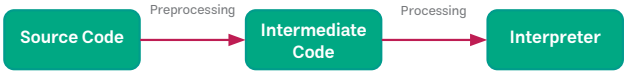
سیدعلی جعفری

Compiler

کامپایلر، به نوعی یک برنامه کامپیوتریه که کدهایی که با زبان برنامه‌نویسی سطح بالا می‌نویسیم رو به زبان ماشین تبدیل می‌کنه. (از زبونی که ما می‌فهمیم به زبونی که کامپیوتر می‌فهمه!)
به طور کلی کامپایلرها، کل سورس‌کد ما رو می‌خونن و اونو تبدیل به یک فایل اجرایی (EXE) می‌کنند که ما از اون استفاده کنیم و نخوایم هر بار دوباره کامپایل کنیم؛ البته مشکل اینجاست که اگر بخوایم کوچکترین تغییری رو توی کدمون بدیم، نیازه که از اول دوباره کل کد رو کامپایل کنیم.

Interpreter

Interpreter (مُفسّر)، یک برنامه کامپیوتریه که هر خط از دستورات یک زبان سطح بالا را به کد ماشین تبدیل می‌کند.
به عبارتی این برنامه میاد سورس‌کد ما رو، خط به خط در حین اجرای برنامه ترجمه و اجرا می کنه و مثل کامپایلر نمی‌آد اول همش رو ترجمه کنه و تبدیل به فایل اجرایی کنه تا ما بتونیم اجرا کنیم. توی مفسری فایل اجرایی تولید نمیشه.



Compiler	Interpreter
چونکه اول کل برنامه را ترجمه می‌کنه و بعد اجرا می‌کنه، آخرش همه اررورها رو با هم نشون میده و خطایابی سخت‌تره.	خط به خط ترجمه کردن، باعث میشه اررورها هم به صورت خط به خط نشون داده بشه و می‌شه راحت‌تر خطای کد رو پیدا کرد.
برای چندمین بار اجرا کردن، نیازی به سورس‌کد نداره.	برای اجراهای بعدی نیاز به سورس‌کد داره.
کد ماشین که ساخته می‌شه در حافظه دستگاه ذخیره می‌شه.	کد ماشین جایی ذخیره نمی‌شه.
کامپایلرها زمان بیشتری رو برای تحلیل و بررسی کد مصرف می‌کنند. البته از لحاظ زمان کلی اجرا، در نهایت از مفسرها سریع‌تر هستن.	مفسرها در مدت زمان کمتری به بررسی می‌پردازند. از لحاظ زمان کلی اجرا در نهایت از کامپایلرها کندتر هستن.
از cpu بیشتر استفاده می‌کنه.	استفاده کمتری از cpu داره.
کارآمد تره.	کارایی کمتری داره.
چون یک فایل به زبان ماشین خروجی میده، میشه اون فایل رو تو هر سیستمی اجرا کرد. (البته به شرطی که توی یه سیستم عامل مشابه کامپایل شده باشه)	برای اجرا، نیازه حتما اون مفسر روی سیستم نصب باشه.
زبان هایی مثل C , C++ از کامپایلر استفاده می‌کنن.	زبان هایی مثل Python , Javascript , Ruby از interpreter استفاده می‌کنن.

BITWISE OPERATION

1010111110010110011

«خب، باشه؛ الان فهمیدم که عملگرهای بیتی چی هستن و چیکار می‌کنن. ولی آخه به چه دردی می‌خورن؟ کجا قراره ازشون استفاده کنیم؟»
شاید شما هم بعد از سولات «سلایق متشابه» و «کامپیوترش خرابه!»، این سوال به ذهنتون آمده باشه. توی این قسمت می‌خوایم یکی از کاربردهای عملگرهای بیتی رو با هم بررسی کنیم و یکمی با استفاده‌هاش در عمل آشنا بشیم.

امیرعلی سلیمی

یکی از داده‌ساختارهایی که در علوم کامپیوتر مورد استفاده قرار می‌گیره، داده‌ساختار Bitset یا Bit Array هست. این داده‌ساختار وظیفه ساده‌ای داره: بیت‌هارو برامون نگه می‌داره! فرض کنید میخوایم تعداد ۸ بیت را در حافظه ذخیره کنیم. در ساده‌ترین حالت می‌تونیم به این شکل عمل کنیم:

```
int b0, b1, b2, b3, b4, b5, b6, b7;
b0 = 0;
b1 = 1;
b5 = b0 | b1;
b6 = b5 & 1;
b7 = b2 ^ b3 ^ b4;
```

در این روش، به ازای هر بیت که می‌خواهیم ذخیره کنیم، ۴ بایت از حافظه (معادل ۳۲ بیت) استفاده می‌کنیم؛ چون برای هر بیت، از دیتا تایپ int استفاده می‌کنیم.

حالا چطور می‌شه مقدار استفاده از حافظه رو کمتر کرد؟ اینجاست که عملیات‌های بیتی به کار ما میان!
در روش قبل، از ۸ متغیر int برای ذخیره ۸ بیت استفاده کردیم. اما اگر کمی دقت کنیم، می‌تونیم از یک متغیر int برای ذخیره کردن ۳۲ بیت استفاده کنیم. از آنجایی که هر متغیر int، چهار بایت از حافظه را اشغال می‌کند، می‌توانیم با استفاده از عملیات‌های بیتی، به هرکدام از ۳۲ بیت آن دسترسی پیدا کنیم. تکه کد زیر را ببینید:

```
int b;
b |= (1 << 10); // set bit 10 to 1
b ^= (1 << 15); // switch bit 15
b &= ~(1 << 20); // set bit 20 to 0
printf(«%d», b & (1 << 23)); // print bit 23
```

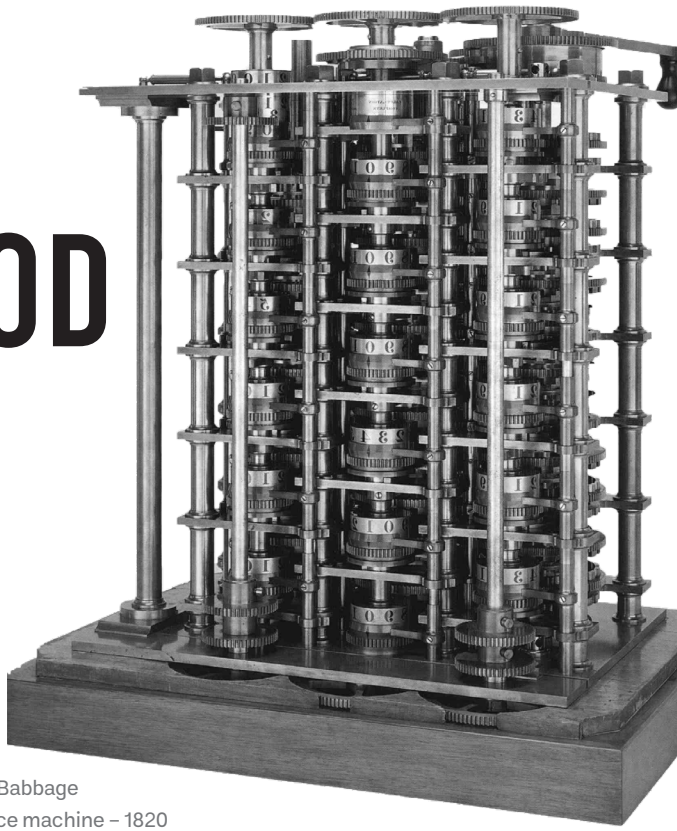
به این شکل، می‌تونیم هر یک از عملیات‌های بیتی را روی هر بیت b انجام دهیم. مقدار حافظه مورد استفاده نیز در روش دوم، ۱/۳۲ روش اول است، که پیشرفت خیلی خوبی هست!

سوال

فرض کنید می‌خواهیم هر یک از بیت‌ها را، با بیت قبلی خود XOR کنیم. مثلا اگر آرایه نهایی را با c نشان دهیم، باید c14 برابر با b13 ^ b14 باشد. این عملیات باید روی همه بیت‌ها انجام شود. اگر از روش دوم برای پیاده‌سازی BitArray خود استفاده کنیم، چطور می‌تونیم این کار رو با تنها یک عملیات بیتی انجام بدیم؟

سیدمحمدحسین حسینی

COMPUTERS, UNDER THE HOOD



Charles Babbage
difference machine – 1820

پیشتر ...

شاید برای شما هم سوال شده‌باشد که پیش از اینکه اساسا کامپیوترها به شکل امروزی و با معماری‌ای شبیه به حالت کنونی وجود داشته‌باشند، انسان‌ها نیازهایشان را چگونه برطرف می‌کردند. نیاز به انجام محاسبات به طور خودکار و با سرعت بالا به خصوص از زمان جنگ جهانی دوم به بعد رو به رشد بود. احتمالا داستان شکاندن رمز انیگما توسط ماشینی که تورینگ طراحی کرده‌بود را شنیده‌اید، ماشینی که طبیعتا با کامپیوتر الکترونیکی و دیجیتالی که امروز با آن سر و کله می‌زنیم تفاوت بسیار داشت. (اگر به این موضوعات علاقه‌مندید در فیلم the imitation game داستان این قضایای مربوط به رمزگشایی انیگما به تصویر کشیده شده‌است)

یا مثلا یکی از این موشک‌های دوران جنگ سرد مثل اسکاد شوروی را در نظر بگیرید. به‌شخصه بعید می‌دانم در آن زمان (دهه ۵۰ میلادی) کامپیوتر خیلی خفنی روی آن کار گذاشته باشند. اما چیزی که می‌دانیم این است که موشک را سیخ روی سکو می‌گذارند و مختصات هدف (شاید نسبت به مکان کنونی) را به طریقی به آن وارد می‌کنند و دکمه. این در حالی‌ست که تاریخچه عملیاتی‌شدن سیستم GPS حداقل به اواخر دهه ۷۰ میلادی بازمی‌گردد. یعنی احتمالا موشک خودش توسط سیستمی که درونش تعبیه شده «متوجه» می‌شود که مثلا کدام طرفی باید برود و... . یعنی چه «متوجه» می‌شود؟ منظور این است که موشک «یک جوری» ورودی‌هایی می‌گیرد که به زبانی است که قابل پردازش برای سیستم آن است. «یک جوری» این ورودی‌ها را پردازش میکند و «یک جوری» بر حسب نتایج پردازش عمل می‌کند، به زبان خودش. یعنی چه به زبان خودش؟
یک مقدمه دیگر اینجا بگوییم و برویم سراغ پاسخ به این پرسش.

کامپیوتر

برای کامپیوتر هم مثل مفاهیم مشابه تعاریف متعددی می‌توان ارائه داد که بعضا با هم نمی‌خوانند. اما یک تعریف سرراست برای کامپیوتر، دستگاهی است که اطلاعاتی را به عنوان ورودی می‌پذیرد، پردازش‌هایی روی آن‌ها انجام می‌دهد و سپس خروجی‌ای می‌دهد (این پردازش‌ها می توانند بر پایه دستورات نرم‌افزاری یا سخت‌افزاری باشند). با این اوصاف بسته به اینکه کامپیوتر چگونه به اصطلاح معماری شده‌باشد، ورودی را به شکلی می‌پذیرد و خروجی را به شکلی می‌دهد. مثلا کامپیوترهای امروزی «زبان»شان دیجیتال است. اگر بخواهیم یک محاسبه عددی، مثلا جمع دو عدد بزرگ، را انجام دهیم، این اتفاق در کامپیوترهای امروزی نمی‌افتد که دو سیم که ولتاژهایشان دقیقا عملوندهای این محاسبه است فرضا با یک مداری به یک سیم خروجی که ولتاژشان برابر جمع این دو ولتاژ است منتج شوند. منظور ما از زبان یک ماشین همین است که چیزی که ما به عنوان داده می‌شناسیم چگونه در این ماشین تفسیر می‌شود؟ مثلا یک داده عددی در کامپیوتر امروزی عمدتا در مبنای ۲ تفسیر می‌شود و انتقال بیت‌ها با بالا یا پایین‌بودن نسبی ولتاژها انجام می‌گیرد.

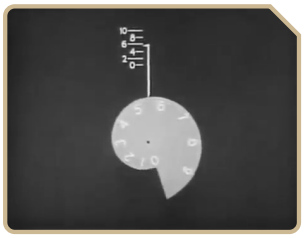
زبان

آیا لزومی دارد که کامپیوترها همیشه به همین زبان کار کنند؟ (لفظ «زبان» که اینجا استفاده میکنیم همانطور که بالاتر تعریف کردیم با چیزهایی که در آینده با عنوان «زبان» با آنها آشنا می‌شوید اندکی متفاوت است) آیا لزومی دارد داده‌ها حتما دیجیتال باشند؟ آیا لزومی دارد داده‌ها حتما به شکل الکترونیکی منتقل شوند و پردازش شوند؟
بیایید خودمان سعی کنیم یک دستگاه توی ذهنمان طراحی کنیم که پردازش‌هایی هم انجام دهد :)

فرض کنیم اول میخواهیم یک داده را نشان دهیم. مثلا یک عدد داخل یک بازه مشخص. یک کاری که میتوان انجام داد این است که یک میله بگذاریم که می‌تواند بالا و پایین شود. یا مثلا یک پیچ یا چرخ‌دنده که زاویه چرخشش نسبت به حالت اولیه نشان‌دهنده عدد باشد. (این دو حالت نمایش داده قابل تبدیل به هم هستند. مثلا یک چرخ‌دنده شانه‌ای که به یک چرخ‌دنده معمولی متصل است اگر بالا یا پایین برود به همان نسبت چرخ‌دنده معمولی می‌چرخد و بالعکس. لذا از اینجا به بعد هر داده را میتوانیم به هر کدام از این دو شکل تصور کنیم)

یا مثلا فرض کنید یک داده را (یک عدد) می‌خواهیم در یک عدد ثابت ضرب کنیم. می‌دانیم اگر دو چرخ‌دنده را به هم متصل کنیم و یکی را بچرخانیم سرعت دورانی چرخش چرخ‌دنده دوم برابر سرعت دورانی چرخش چرخ‌دنده اول ضربدر نسبت شعاع این ۲ چرخ‌دنده است. پس می‌توان این فرایند را هم روی داده انجام داد.

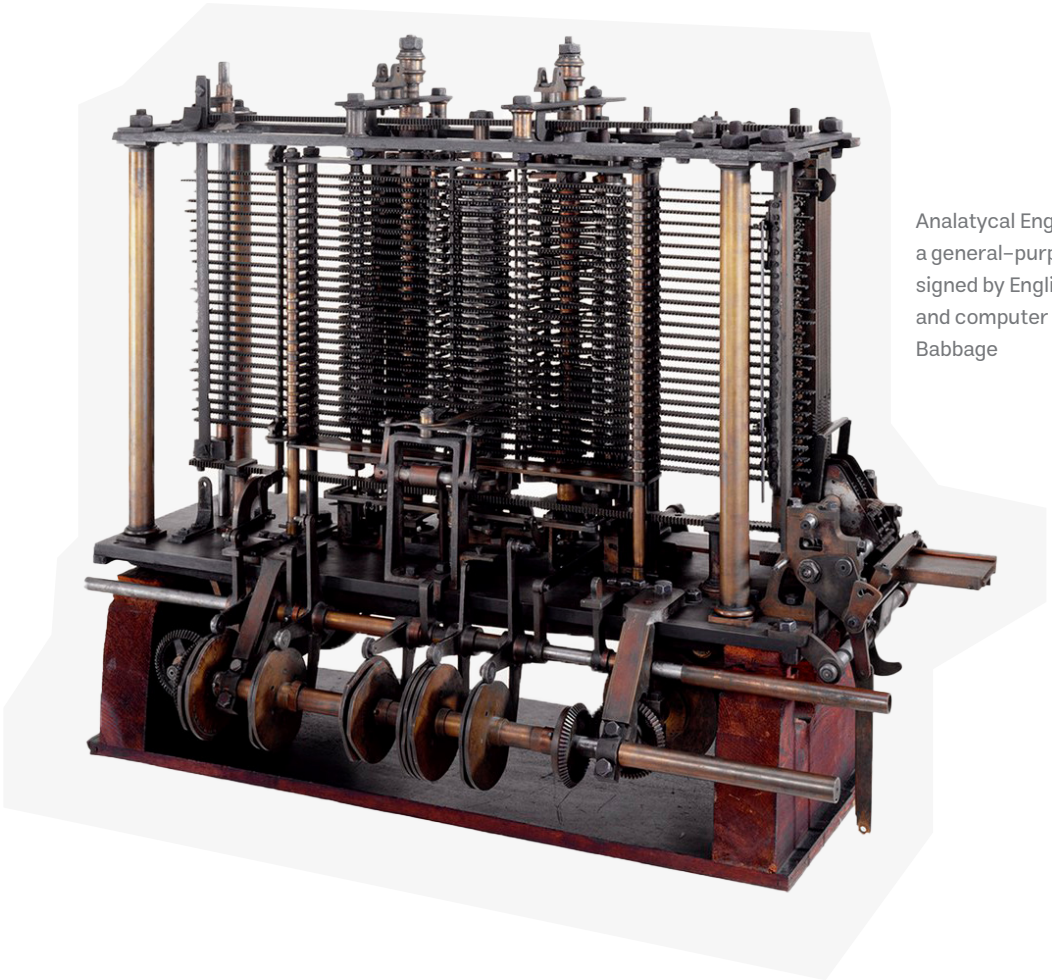
یا مثلا فرض کنید می‌خواهیم یک تابع مشخص با یک ورودی پیاده کنیم. انگار می‌خواهیم به ازای هر مقدار چرخش یک محور، یک مقدار مشخص خروجی بدهیم. احتمالا در طرف خروجی استفاده از نمایش چرخ‌دنده‌ای عدد خیلی کارگشا نیست. فرض کنید میخواهیم بر حسب زاویه چرخش یک محور، یک میله به مقدار مشخصی بالا یا پایین برود. در اینجا از میل بادامک استفاده می‌کنیم و بادامک مخصوص آن تابع را طراحی می کنیم .



پیاده‌سازی تابع y=x برای بازه ۰ تا ۱۰

یا مثلا جمع دوتا داده با استفاده از مکانیزمی تحت عنوان دیفرانسیل انجام می‌شود. تقریبا همان مکانیزمی که به اتومبیل‌ها این اجازه را می‌دهد که در پیچ‌ها چرخی که به سمت داخل پیچ هست مسافت کوتاه‌تری را نسبت به چرخ بیرونی طی کند :)

القصه، مکانیزم‌هایی وجود دارد که کارهای دیگری هم با آنها می‌توان انجام داد. مثلا مکانیزمی که ضرب دو عدد را ممکن می‌کند یا حتی مکانیزمی که از یک تابع انتگرال می‌گیرد :) نکته جالب اینکه مکانیزم‌ها و ماشین‌هایی که اینجا معرفی کردیم واقعا کاربردی بوده‌است. در یک ماشین کنترل آتش که فیلم



Analatycal Engine
a general-purpose computer designed by English mathematician and computer pioneer Charles Babbage

آموزشی آن را میتوانید از این لینک ببینید مسئله‌ای که ماشین قرار بود حل کند این بود که یک کشتی می‌خواست به یک کشتی دیگر شلیک کند اما اینجا هم سرعت دو کشتی و هم متغیرهای دیگری در زاویه مناسب شلیک توپ تاثیرگذار بود که محاسبه دستی جهت درست پرتاب توپ را تقریبا غیرممکن می‌کند.

اما کامپیوترهایی که امروزه استفاده می‌کنیم اکثرا «زبان»شان دیجیتال است. آیا می‌توان یک کامپیوتر مکانیکی دیجیتال ساخت؟ اگر ۲ مکانیزم بتوانیم بسازیم که دو میله به آن‌ها وارد می‌شود و یک میله خروجی دارند و خروجی در یکی or و در دیگری and ورودی‌ها باشد در درس مدارهای منطقی خواهید دید که اثبات می‌شود که می‌توان هر تابعی از هر چند ورودی را پیاده کرد. (مثلا میله دو حالت داشته باشد. یا فرو برود که ۱ است یا بیرون باشد که ۰ است)

در کل، اینکه ماشین به چه زبانی کار می‌کند خیلی در نتیجه تاثیرگذار نیست. مهم این است که ورودی را بتوان به آن داد و نتیجه مورد انتظار را از آن در زمان مناسب گرفت. می‌تواند دیجیتال باشد یا آنالوگ. الکترونیکی باشد یا مکانیکی. به مثال موشک اسکاد برگردیم؟ احتمالا (یعنی حدس می‌زنم) الکترونیکی و آنالوگ است با تعداد زیادی سنسور و مدارهای پیچیده و... . مثلا احتمالا در ترم ۳ در درس مبانی مدارهای الکتریکی و الکترونیکی با مداری به نام آپ‌امپ آشنا می‌شوید که با استفاده از آن می‌شود ولتاژی را ضرب در ضریب ثابتی کرد یا انتگرال گرفت یا مشتق گرفت یا ... . اما الان شهود داریم که چطور ممکن است کار کند.
مثلا فرض کنید یک نفر فاصله هدف را در ۳ بعد به آن می‌دهد (مثلا با چرخاندن یک چرخ‌دنده یا محور) و بعد دکمه را می‌زند و موتور روشن می‌شود.
فرض کنید که یک ژیروسکوپ در کلاهک موشک است که شتاب را در هر لحظه اندازه می‌گیرد. با یکی دو بار انتگرال گرفتن از شتاب می‌توان موقعیت کنونی و در نتیجه فاصله کنونی از مقصد و همچنین سرعت کنونی را به دست آورد و بعد با استفاده از تابعی فهمید که باید سر موشک را به کدام سمت خم کرد. چیزی که ما اکنون از آن شهود داریم که بدون کامپیوترهای امروزی هم ممکن است. انسان بدون کامپیوترهای امروزی توانسته‌است به ماه برود :)
اگر به موضوع علاقه‌مندید می‌توانید در مورد این ۳ موضوع جستجو کنید و چیزهای جالبی پیدا کنید: کامپیوترهای مکانیکی دیجیتال ، کامپیوترهای مکانیکی قابل برنامه‌نویسی و ماشین آقای چارلز بابیج که به اولین کامپیوتر هم معروف است.

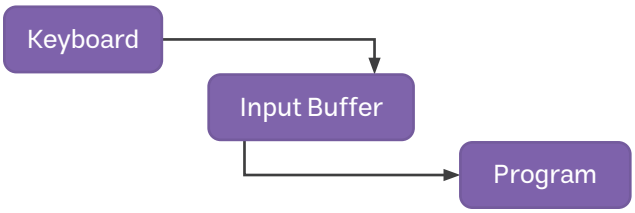
Anigma
(cypher device developed and used in the early 20th century to protect commercial, diplomatic, military communications)

GETS OR FGETS?

THAT'S THE PROBLEM

ترانه خسروجردی

برای آشنایی بیشتر با توابع مربوط به ورودی و خروجی، ابتدا لازم است با مفهوم Buffer آشنا شویم. به بخشی از حافظه که به طور موقت برای ذخیره کردن ورودی‌ها و خروجی‌های برنامه مورد استفاده قرار می‌گیرد buffer گفته می‌شود. زمانی که یک متغیر را به برنامه ورودی می‌دهید، این متغیر در Buffer ذخیره می‌شود، همچنین لازم است تا پیش از استفاده دوباره، حافظه buffer پاک شود.



ممکن است تا به حال هنگام استفاده از scanf به هشدار (warning) هایی بر خورده باشید، اغلب این هشدارها به این اشاره دارند که scanf تابعی ناامن است. یکی از مشکلاتی که سبب ناامن شدن scanf می‌شوند، این است که در این تابع هنگام ورودی گرفتن، طول مشخصی برای ورودی در نظر گرفته نشده است، در نتیجه حتی اگر طول ورودی بیشتر از فضای حافظه buffer شود تابع همچنان به ورودی گرفتن ادامه می‌دهد، در این صورت برنامه حین اجرا به اروری بر می‌خورد که به آن buffer overflow می‌گویند و سبب از کار افتادن برنامه می‌شود. از مشکلات دیگر scanf این است که برای ورودی گرفتن رشته‌هایی که شامل کاراکتر space باشند به مشکل می‌خورد. زمانی که در رشته ورودی کاراکتر space وجود دارد میتوان از دو تابع gets و fgets استفاده کرد. این دو تابع فضای مشخصی از حافظه را به عنوان buffer ورودی می‌گیرند و ورودی داده شده توسط کاربر را درون آن می‌نویسند. تفاوت اصلی این دو تابع در این است که gets تنها زمانی نوشتن ورودی در buffer را متوقف می‌کند که به کاراکتر enter یا کاراکتر EOF (که نشان دهنده ی پایان یک فایل است) برسد، در حالی که برای fgets علاوه بر کاراکتر enter و EOF روش دیگری هم برای اتمام ورودی گرفتن وجود دارد، زمانی که تعداد کاراکترهای نوشته شده در buffer به حد معینی برسد fgets ورودی گرفتن را متوقف می‌کند، در نتیجه از وقوع buffer overflow احتمالی جلوگیری می‌کند. همین قضیه سبب برتری fgets در مقایسه با gets می‌شود.

برای مثال در کد زیر، اگر رشته‌ای که کاربر به عنوان ورودی می‌دهد طولی بیشتر از ۱۰ داشته باشد، تابع gets همچنان به ورودی گرفتن ادامه می‌دهد و buffer overflow رخ می‌دهد.

```
int main() {
      char buffer[10];
      gets(buffer);
      printf(buffer);
      return 0;
}
```

برای علاقه مندان

در ادامه درس با مفهوم pointer آشنا می‌شوید. تابع gets به عنوان ورودی یک پوینتر که به بلوکی از حافظه اشاره دارد می‌گیرد. سینتکس آن به صورت زیر است که در آن str همان پوینتر گفته شده است.

```
gets (char* str);
```

همچنین سینتکس fgets در زیر آورده شده که در آن str پوینتری به یک بلوک از حافظه است، n ماکسیمم تعداد کاراکتری ست که این تابع از ورودی دریافت می‌کند و stream منبعی که ورودی باید از آن دریافت شود را مشخص می‌کند. برای مثال ممکن است نیاز باشد ورودی از یک فایل دریافت شود، اگر به عنوان ورودی استریم stdin را به تابع ورودی دهیم، ورودی را از کاربر در ترمینال دریافت می‌کند (مشابه gets).

```
fgets (char* str, int n, FILE* stream);
```

HOW SIRI WORKS?

امیرحسین رازلیقی

اگـر از دوسـتان طرفـدار اپـل (apple fan) باشـین، حتمـا یکـی از ویژگی‌هـای دوست‌داشـتنی اکوسیسـتم اپـل رو دسـتیار هوشـمندش میدونیـن. اگـر هـم جـزوی از ایـن اکوسیسـتم نباشـین، حتمـا تـا الان آوازه‌ی سـیری (siri) بـه گوشـتون خـورده و یـه چیزایـی راجـع بهـش میدونیـن. بعـد از سـیری، دسـتیارهای هوشـمند زیـاد دیگـه‌ای هـم اومـدن (مثـل alexa و google assistant و ...) ولـی بـا دقـت خوبـی میشـه گفـت کـه هیچکـدوم بـه محبوبیـت و جذابیـت جهانـی سـیری نرسـیدن.

یکـی از مهمتریـن کارهایـی کـه سـیری پشـت پـرده انجـام میـده تـا دسـتیار هوشـمند بهتـری باشـه، پـردازش زبـان طبیعـی (Natural Language Processing) هسـتش! کاری کـه از نظـر مـا انسـان‌ها شـاید خیلـی سـخت نباشـه امـا تـوی دنیـای کامپیوتـر اینطـوری نیسـت.

مـا انسـان‌ها وقتـی بـه ایـن دنیـا میایـم، از همـون لحظـات اول درحـال یادگیـری هسـتیم. یادگیـری اسـم اجسـام، کاربردشـون، نحـوه‌ی کار بـا اونهـا، یادگیـری زبـان مـادری و خیلـی چیزهـای دیگـه. یکـی از اولیـن چیزهایـی کـه یـک کـودک میتونـه انجـام بـده، یادگیـری معانـی کلمـات مختلـف و نحـوه‌ی بـکار بـردن اونهـا داخـل یـک جملـه بـرای بیـان نیـاز خودشـه. اولیـن بـار شـاید تمـام مـا بـا گفتـن کلمـات کوتـاه و بعضـا بـی معنـی، فرآینـد «حـرف زدن» رو شـروع کردیـم و رفته‌رفتـه، ترکیبـات پیچیده‌تـر و کلمـات طولانی‌تـر رو یـاد گرفتیـم و اسـتفاده کردیـم. امـا واقعـا آیـا یادگیـری یـک زبـان بـه همیـن سـادگیه؟!

یک عبارت معروفی در دنیای کامپیوتر هست که میگه:

> *کارهایـی کـه بـرای انسـان‌ها خیلـی سـخته، بـرای کامپیوترهـا خیلـی آسـونه ولـی کارهایـی کـه بـرای انسـانها خیلـی آسـونه، بـرای کامپیوترهـا خیلـی سـخته!*

شـاید از نظـر مـا یادگرفتـن یـک زبـان جدیـد اونقـدرا هـم فرآینـد پیچیـده و عجیبـی نباشـه! یـک کتـاب زبـان می‌خریـم و از الفبـای اون زبـان شـروع میکنیـم و کـم کـم وارد سـاختارهای زبانـی پیچیـده میشـیم (گرامـر). تـا اینجـای کار، انـگار یـاد دادن این‌هـا بـه کامپیوتـر خیلـی هـم سـخت نیسـت. امـا یـک ایـراد اساسـی وجـود داره. کامپیوتـر، برخـلاف انسـان، معنـی و مفهـوم کلمـات مختلـف رو نمیفهمـه!

مثـلا از نـگاه کامپیوتـر دو جملـه‌ی «ایـن کتـاب خیلـی طولانـی بـود» و «ایـن کتـاب خیلـی بلنـد بـود» یکسـانن! چـون تـوی لغتنامـه‌ی اون، معنـای دو کلمـه‌ی «طولانـی» , «بلنـد» یکسـانن (فـارغ از اینکـه تـوی چـه جملـه‌ای و چجـوری اسـتفاده بشـن)!

کم‌کـم داریـم بـا پیچیدگی‌هـای یـاد دادن به یـک کامپیوتر آشـنا میشـیم! درواقـع انـگار کامپیوتـر یـک بچـه‌ی تـازه متولـد شده‌سـت کـه هیچ‌چیزی راجـع بـه ایـن مفهـوم (concept) نمیدونه و ما سـعی داریـم آروم‌آروم این مفاهیـم رو (معانـی کلمات، گرامـر زبـان و ...) رو بهش آمـوزش بدیم. این، یـک تعبیـر خیلی سـاده و در عین حـال مفهومـی از چیزیه که ایـن روزها به عنـوان یادگیـری ماشـین (Machine Learning) ازش یـاد میشـه. پردازش زبـان طبیعـی هـم زیرمجموعـه‌ای از همین فیلد رشـته‌ی ما هسـت!

Going in depth!

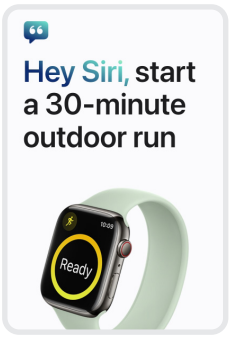
خـب، حـالا کـه بـا مقدمـات ابتدایـی ایـن مفاهیـم آشـنا شـدیم، کمـی جلوتـر بریـم و ببینیـم پشـت پرده‌هـای سـیری چه‌خبـره! سـیری در ابتـدا یـک نرم‌افـزار داره کـه (مثـل مغـز انسـان) مفاهیـم کلمـات و تفـاوت جایگاهشـون در جملـه و گرامـر زبـان و اینجور مـوارد رو یـاد گرفته. در واقـع، ایـن نرم‌افـزار، توسـط مهندسـین بـه روش‌هـای مختلـف طراحـی میشـه تـا کامپیوتـر بتونـه تـا حـد بسـیاری شـبیه بـه انسـان (و حتـی در مـواردی بهتـر از انسـان!) مفاهیـم زبـان طبیعـی رو درک کنـه و اصطلاحـا زیـر و بـم اون زبـان رو بشناسـه. مثـل همـه‌ی نرم‌افزارهـای

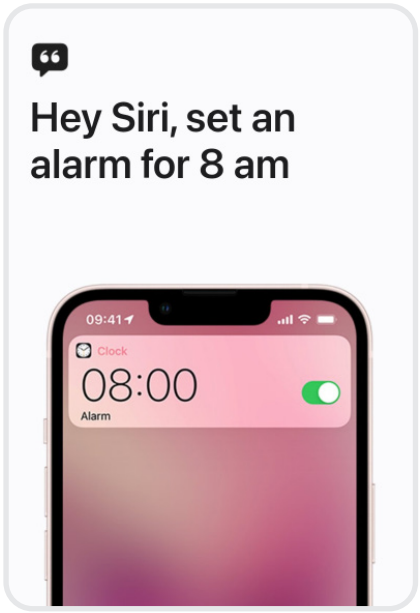
مختلـف، ایـن نرم‌افـزار هـم از اولیـن زمـان ارائـه‌ش تـا الان بارهـا بهبـود داشـته و حتـی در بعضـی مـوارد، برخـی بخش‌هـای اون از صفـر مجـددا نوشـته شـده‌ن! بـرای همیـن هـم هسـت کـه هـر وقـت شـما از سـیری اسـتفاده میکنیـن، تـوی پس‌زمینـه (background) سیسـتم عاملتـون دنبـال نسـخه‌ی جدیـدی از اون میگـرده و در صـورت وجـود، اون رو جایگزیـن نسـخه‌ی فعلـی روی گوشـی شـما میکنـه.

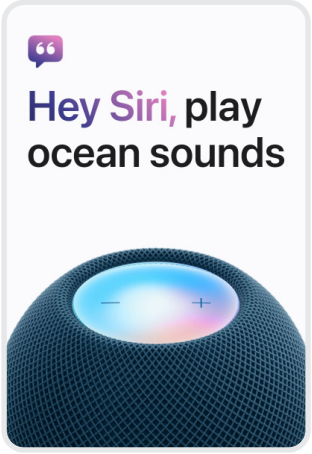
همونطـور کـه شـاید حـدس بزنیـن، درک جملاتـی کـه بـه ایـن دسـتیار صوتـی گفتـه میشـه، محاسـبات خیلـی زیـاد و پیچیـده‌ای داره و نیـاز بـه قـدرت محاسـباتی بالایـی داره. اپـل تـا مدت‌هـا، سـیری رو بصـورت آنلایـن ارائـه می‌کـرد. دلیلـش هـم ایـن بـود کـه گوشـی‌های هوشـمند تـوان پردازشـی کافـی بـرای ایـن کار رو نداشـتن و در واقـع جملاتـی کـه شـما میگفتیـن، طـی یـک رمزگـذاری (Encoding) خـاص بـرای سـرورهای اپـل فرسـتاده میشـدن و اونجـا محاسـبات مربوطـه روشـون صـورت میگرفـت و جـواب درسـت (کـه سـیری بایـد اون‌رو بـه شـما بگـه) توسـط سـرور بـه گوشـی شـما فرسـتاده میشـد. البتـه انقـدر ایـن فرآینـد مهندسـی شـده و اصولـی بـود کـه تمـام ایـن اتفاقـات در کسـری از ثانیـه (البتـه بـا یـک سـرعت اینترنـت معقـول) مـی افتادنـد. بـا پیشـرفت تکنولـوژی و طراحی‌هـای سـخت افـزاری جدیـد پردازنده‌هـای اپـل، چنـد سـاله کـه تمـام فرآیندهـای محاسـبات و تحلیـل و پـردازش مربـوط بـه سـیری، روی خـود گوشـی شـما انجـام میشـه و چیـزی بـرای سـرورهای اپـل ارسـال نمیشـه و در واقـع شـما میتونیـن بـه صـورت آفلایـن هـم از سـیری اسـتفاده کنیـن! ایـن اتفـاق رو مدیـون طراحـی پردازنده‌هـای جدیـد اپـل مثـل A15-Bionic هسـتیم کـه در اون‌هـا، طراحـی مخصوصـی در نظـر گرفتـه شـده تـا محاسـبات سـنگینی مثـل ایـن (محاسـبات مربـوط بـه پـردازش زبـان) و حتـی محاسـبات سـنگین‌تر، در کسـری از ثانیـه انجـام بشـن!

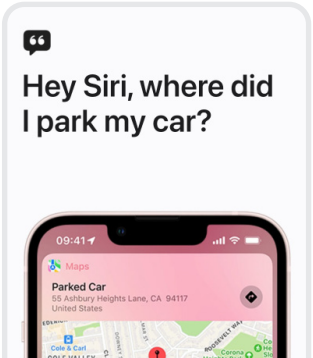
نکتـه‌ی جالـب اینکـه هنـوز خیلـی از دسـتیارهای هوشـمند دیگـه (مثـل google assistant روی برخـی گوشـی‌های اندرویـدی و amazon alexa) بخـش زیادیشـون بـه صـورت آفلایـن کار نمیکننـ و همچنـان وابسـته بـه سـرور (server-based) هسـتن.

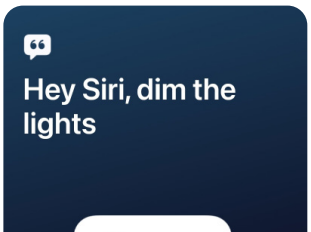
درآخـر ایـن متـن هـم یـک نکتـه‌ی جالـب رو اشـاره کنـم. شـما بـرای فعـال کـردن سـیری روی محصـولات اپـل کافیـه عبـارت «Hey Siri» رو بگیـن تـا دسـتیار هوشـمند اپـل فعـال بشـه و بـه جمـلات بعـدی شـما گـوش کنـه و بهشـون جـواب بـده. تـوی آخریـن مصاحبـه‌ای کـه رئیـس بخـش مهندسـی سـیری داشـته، گفتـه بـرای آپدیـت بعـدی سـیری (کـه احتمـالا قـراره بـا نسـخه‌ی بعـدی ios منتشـر بشـه)، تمـام تیمشـون درحـال کار روی یـک ویژگـی هسـتش. چـه ویژگـی‌ای؟ اینکـه بجـای «Hey Siri» شـما بتونیـن بـا گفتـن «Siri» و بـدون هیـچ کلمـه‌ی دیگـه‌ای، دسـتیار صوتـی اپـل رو فراخوانـی کنیـن! در نـگاه اول بنظـر خیلـی کار سـختی نمیـاد و اینکـه تیـم بـه اون بزرگـی، بـرای زمـان طولانـی روی ایـن موضـوع متمرکـز باشـن بنظـر مسخره‌ست! امـا بایـد بگـم کـه اصـلا اینطـور نیسـت نکتـه اینـه کـه خیلـی وقتـا مـا جملاتـی میگیـم کـه ممکنـه شـبیه عبـارت «Siri» بنظـر بیـان در حالـی کـه اصـلا نخوایـم از سـیری اسـتفاده کنیـم! اگـر کاربـر ایـن دسـتیارهای هوشـمند باشـین، احتمـالا براتـون پیـش اومـده کـه گوشـی تـوی جیبتونـه و میبینیـن کـه داره صـدا میـده و دسـتیار صوتیـش فعـال شـده! اینـا در واقـع دلایـل زیـادی (مثـل نویـز صـدای محیـط و غیـره) میتونـه داشـته باشـه. در واقـع داشـتن یـک عبـارت دو بخشـی (Hey و siri) احتمـال خطـا تـوی تشـخیص فراخوانـی ایـن دسـتیار صوتـی رو خیلـی کـم میکنـه. امـا اینکـه همیـن مقـدار دقـت رو بخوایـم بـا یـک عبـارت تـک کلمـه‌ای (siri) داشـته باشـیم، چالشـیه کـه ایـن روزهـا مهندسـین اپـل باهـاش دسـت و پنجـه نـرم میکننـ!

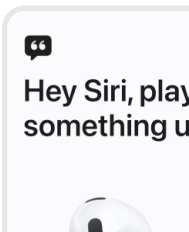












“YOU GET WHAT YOU WORK FOR, NOT WHAT YOU WISH FOR”

کدنامه

شماره: ۱
تاریخ انتشار: ۱ آذرماه ۱۴۰۱
نویسندگان: محمدپارسا بشری، ترانه خسروجردی، امیرعلی سلیمی
امیرحسین رازلیقی، سیدعلی جعفری، سیدمحمدحسین حسینی
ویراستاران: عرفان مجیبی، آرش یادگاری و امیرحسین رازلیقی
طراح: سجاد سلطانیان

دستیاران آموزشی دروس مبانی برنامه‌سازی و برنامه‌سازی پیشرفته دانشکده مهندسی کامپیوتر،مجموعه مطالب آموزشی‌ای برای درک بهتر درس و همینطور علاقه‌مند شدن بیشتر به مباحث، در اختیار دانشجویان قرار می‌دادند. پس از مدتی، تصمیم بر آن شد که این مجموعه مطالب، در قالب یک نشریه اختصاصی، در قالبی مشخص در اختیار دانشجویان قرار بگیرد تا هم منبعی برای تکمیل دانسته‌های دانشجویان از این دو درس باشد و هم راهی برای شناخت هر چه بیشتر دنیای وسیع مهندسی کامپیوتر!