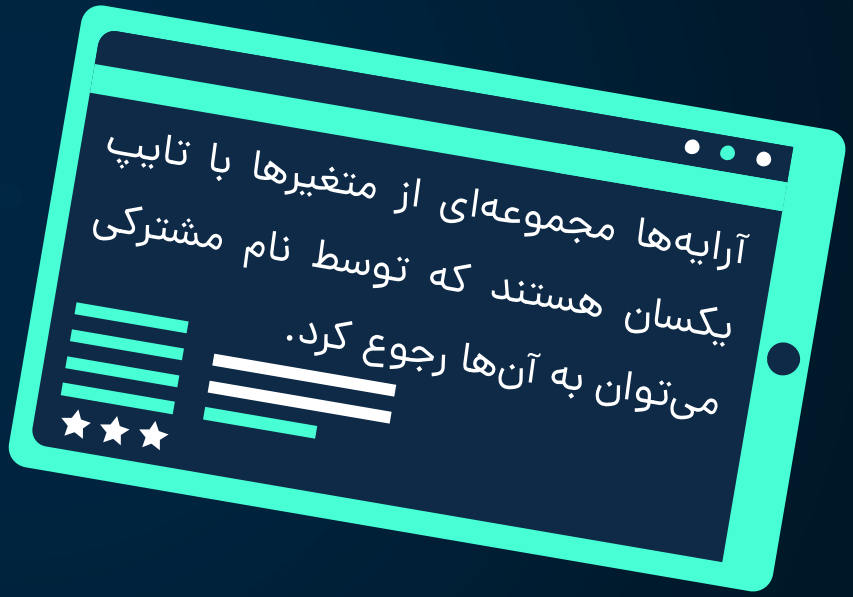

بسم الله الرحمن الرحیم

# آرایه‌ها و رشته‌ها

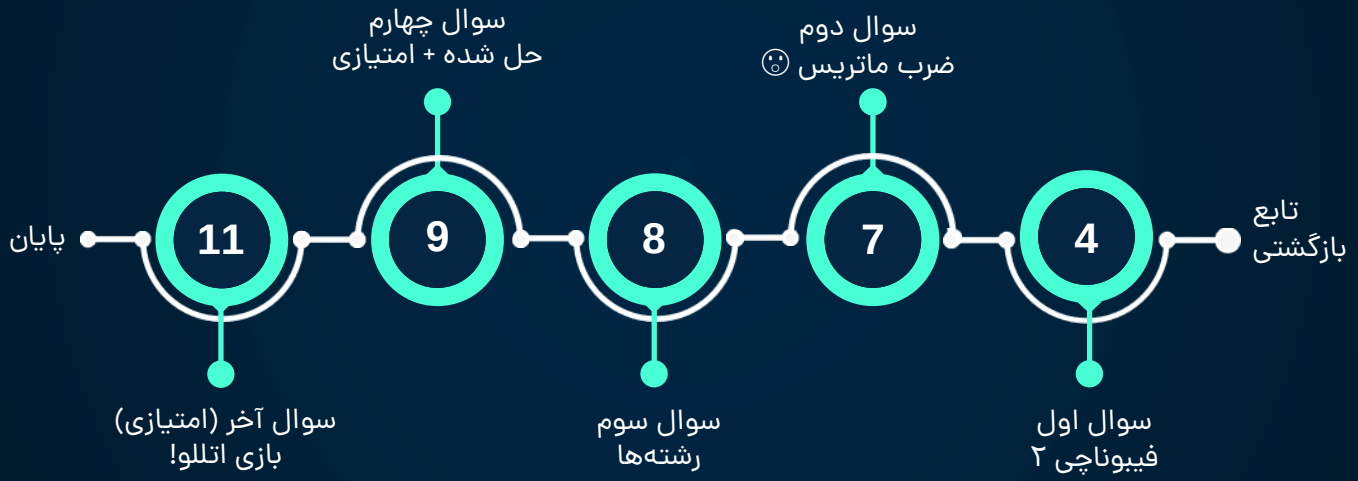
جلسه نهم

تا این‌جا، همیشه برای ذخیره‌ی اطلاعات مورد نیاز برنامه‌هایمان از متغیرها استفاده می‌کردیم.
تا به حال با این مشکل مواجه شده‌اید که به دلیل نیاز به ذخیره‌ی اطلاعات متعدد، متغیرهای برنامه‌ی شما بسیار زیاد و مدیریت آن‌ها پیچیده شود؟




آرایه‌ها در خیلی از موارد می‌توانند این نیاز ما را برطرف کنند. البته خیلی وقت‌ها هم نمی‌توانند! اما در هر صورت اهمیت استفاده از آرایه به زودی با نوشتن برنامه‌های طولانی‌تر و به نسبت پیچیده‌تر برایتان روشن خواهد شد. به همین دلیل، این جلسه از کارگاه به مبحث آرایه‌ها و در کنار آن کار با رشته‌ها اختصاص داده شده است.

# فهرست


تابع
بازگشتی
4
سوال اول
فیبوناچی ۲
7
سوال دوم
ضرب ماتریس
8
سوال سوم
رشته‌ها
9
سوال چهارم
حل شده + امتیازی
11
سوال آخر (امتیازی)
بازی اتللو!
پایان

# سوال اول: فیبوناچی ۲


در جلسه‌ی گذشته تابع فیبوناچی را به روش‌های مختلفی پیاده‌سازی کردید. در این جلسه قصد داریم روش‌هایی برای بهبود عملکرد این تابع بازگشتی ارائه دهیم.

همان‌طور که احتمالا متوجه شده‌اید، تابع بازگشتی فیبوناچی به ازای nهای بزرگ عملکرد ضعیفی دارد و به مدت زمان زیادی برای محاسبه لازم دارد. آیا می‌توانید با توجه به نمودار بالا، علت این زمان اجرای طولانی را توضیح دهید؟

ت) یکی از روش‌های بهبود زمان اجرا در توابع بازگشتی، استفاده از روش memoization (با memorization اشتباه نگیرید) است. در این صفحه، کد C تابع فیبوناچی را که با این روش بهبود یافته است، می‌بینید:

```c
#include <stdio.h>
// stores fibonacci sequences in the array.
// global variables are initialized to zero in C

int memory[1000];

int fibonacci(int n) {
    if (n == 1 || n == 2) {
        return 1;
    }

    if (memory[n] == 0) {
        // memorizes what it calculated now.
        memory[n] = fibonacci(n - 1) + fibonacci(n - 2);
    }

    return memory[n];
}
```

به نظر شما این روش چگونه سرعت محاسبه را افزایش می‌دهد؟

یکی دیگر از روش‌های افزایش سرعت در محاسبات بازگشتی، روش برنامه‌نویسی پویا است.

ث) در روش برنامه‌نویسی پویا، به جای فراخوانی تابع به صورت بازگشتی، از یک آرایه استفاده می‌کنیم و به این ترتیب مقدارهای قبلی در آرایه ذخیره می‌شوند و می‌توان از آن‌ها به سادگی استفاده کرد.

تابع فیبوناچی را با استفاده از برنامه‌نویسی پویا پیاده‌سازی کنید. توجه کنید که برای محاسبه مقدار تابع به ازای هر ورودی n، باید تمام مقادیر تابع به ازای nهای کوچکتر از ورودی در یک آرایه ذخیره شوند.

# سوال دوم: ضرب ماتریس

همان‌طور که می‌دانید و در ابتدای دستورکار هم ذکر شد، آرایه‌ها مجموعه‌ای از متغیرها با تایپ یکسان هستند که می‌توان توسط نام مشترکی به آن‌ها رجوع کرد. ما به کمک یک index مشخص می‌کنیم که کدام خانه از آرایه مد نظر ماست؛ اما می‌توانیم تعداد بیش‌تری index داشته باشیم و مثلا به کمک دو index بتوانیم یک آرایه‌ی دوبعدی را مدیریت کنیم.

یکی از مهم‌ترین برنامه‌ها در زمینه‌ی کار با آرایه‌های دو بعدی، مسائل مربوط به ماتریس‌ها هستند.

برای تمرین برنامه‌نویسی با آرایه‌های دوبعدی، برنامه‌ای بنویسید که یک ماتریس n در m را در یک ماتریس p در q ضرب کند. این برنامه را به صورت بازگشتی بنویسید.

راهنمایی: دقت کنید که شرط‌های ضرب ماتریس رعایت شوند. برای بازگشتی نوشتن ضرب دو ماتریس می‌توانید از انواع مختلف متغیرها که در جلسه‌ی گذشته با کدخدا و Botfather تمرین کردید، استفاده کنید. سعی کنید به طور کاملا اتفاقی یاد متغیرهای static بیفتید =)

# سوال سوم: رشته‌ها

برنامه‌ای بنویسید که با کمک آن بتوانیم کاراکتری که بیش‌ترین تکرار را در یک رشته (string) دارد پیدا کنیم؟

سوال همین قدر خشک و خالی :)) هدف از این سوال این هست که متوجه شوید آیا مبحث رشته‌ها را به خوبی فرا گرفته‌اید یا خیر. در صورتی که احساس می‌کنید نیاز است تا مثال حل شده‌ای از این مبحث را ببینید، می‌توانید به سوال بعدی مراجعه کنید.

همچنین در انتهای آن بخش، سوالی به نسبت سخت‌تر به عنوان سوال امتیازی وجود دارد که می‌توانید برای سر و کله زدن بیش‌تر با رشته‌ها، آن را انجام دهید.

# سوال چهارم: حل شده + امتیازی

- همه‌ی ما از ابزارهایی برای کار با متن‌های مختلف استفاده می‌کنیم که این ابزارها، رشته (string)ها را پردازش می‌کنند. حالا می‌خواهیم کدهای پشت پرده‌ی این برنامه‌ها را برای پردازش متن‌ها تست کنیم.
- حتما تا حالا برایتان پیش آمده که متنی را در word نوشته باشید و بعد بخواهید به جای یک حرف یا یک کلمه، کلمه یا حرف دیگری را جایگزین کنید.

- ما می‌خواهیم کدی بنویسیم که این کار را در پروژه‌های بزرگ‌تر برای ما انجام دهد و یک حرف (character) که در تمام جمله‌ی ورودی اشتباه نوشته شده را با حرف درست جایگزین کند. می‌توانید کد برنامه را در لینک زیر بیابید.

find_and_replace_character https://b2n.ir/p70580

```c
/* C Program to Replace All Occurrences of a Character in a String */
#include <stdio.h>
#include <string.h>

int main() {
    char str[100], ch, Newch;
    int i;
    printf("\n Please Enter any String: ");
    gets(str);
    printf("\n Please Enter the Character that you want to Search for: ");
    scanf("%c", &ch);
    getchar();
    printf("\n Please Enter the New Character: ");
    scanf("%c", &Newch);
    for(i = 0; i <= strlen(str); i++) {
        if(str[i] == ch) {
            str[i] = Newch;
        }
    }
    printf("\n The Final String after Replacing All Occurrences of '%c' with '%c' = %s ",
           ch, Newch, str);
    return 0;
}
```

قاعدتا تا الان انقدر در مهارت درک کد رشد کرده‌اید که در چشم بر هم زدنی این کد را بررسی کنید و متوجه شوید که دقیقا چطور عمل می‌کند.

حال به عنوان سوال امتیازی: آیا می‌توانید عملکرد find and replace در برنامه‌هایی مانند word را در برنامه‌ای پیاده‌سازی کنید؟

یعنی برنامه‌ی شما پس از دریافت جمله‌ای در ورودی، یک کلمه که به نظر کاربر در تمام جمله‌ی ورودی اشتباه نوشته شده را با کلمه‌ی درخواست شده توسط وی جایگزین کند. دست به کد بشید و انجام بدید!

# سوال آخر امتیازی: بازی اُتللو!

سلام بچه‌ها حالتون چطوره؟ با جذاب‌ترین بخش دستورکار دوباره در خدمتتون هستیم =)
توی این قسمت می‌خوایم یه بازی خیلی باحال و قشنگ و محبوب رو با هم دیگه پیاده‌سازی کنیم.
بازی اتللو! قول میدم آخرش کلی کیف کنین با برنامه‌تون.

این بازی چطوریه؟ خیلیاتون احتمالا آشنا هستین و قبلا بازی کردین. برای اون دسته از کسایی که آشنایی ندارن باید بگم که بازی اتللو یه صفحه داره که متشکل از تعدادی خونه هست که این صفحه معمولا هشت در هشته. هر بازیکن یه رنگ بین قرمز و سفید انتخاب می‌کنه و بازی شروع می‌شه. هرکسی توی نوبت خودش می‌تونه مهره‌شو هرجایی که خواست بذاره. حالا که چی؟ اگه یک یا تعدادی مهره از یک بازیکن بین دو تا مهره‌ی بازیکن حریف باشه، همه‌ی اون مهره‌ها رنگشون به رنگ مهره‌ی بازیکن حریف عوض می‌شه. مثلا من رنگم سفیده و یه مهره‌ی سفید توی خونه‌ی ۳ می‌ذارم. حالا حریفم یه مهره‌ی قرمز توی خونه‌ی ۴ می‌ذاره. الان اگه من بیام و یه سفید توی خونه‌ی ۵ بذارم، چون مهره‌ی قرمز بین دو تا مهره‌ی من قرار گرفته، رنگش عوض میشه و سفید میشه!

این‌قدر این کار ادامه پیدا می‌کنه تا همه‌ی خونه‌ها پر بشه. نهایتا هر کی تعداد رنگ بیش‌تری ازش توی صفحه بود برنده‌س! نکته‌ی مهم اینه که نمی‌تونین مهره رو هر جایی که دلتون خواست بذارین و باید حتما مجاور یه مهره‌ی دیگه بذارین (اول کار ۴ تا مهره با یه ترتیب مشخصی وسط صفحه قرار گرفتن که باید کنار اینا گذاشت و از اونجا بازیو شروع کرد و هی گسترشش داد تا آخر)

یکم این توضیحات ممکنه گنگ باشه براتون، برای همین برید توی این لینک زیر و آنلاین بازی کنید تا یکم بازی رو بهتر درک کنین.

Play Othello online https://www.eothello.com/

حالا که یاد گرفتین بازی رو بریم سراغ پیاده‌سازیش.

ما یه کدی براتون آماده کردیم و یه بخشاییش رو پر کردیم و یه قسمتاییش رو خالی گذاشتیم تا خودتون پر کنین. یکی از مهم‌ترین مهارت‌های یک مهندس کامپیوتر، توانایی خوندن کد بقیه و درک کردن و فهمیدنش و در مرحله‌ی بعد ادامه دادن و کامل کردنشه. هدف ما هم از این مدل سوالا اینه که با این موضوع آشنا بشین و مهارت کدخوانیتون بره بالا =)

پس شما فقط کافیه بخش‌هایی که لازمه رو تکمیل کنین تا برنامه درست کار کنه. اما برای بهتر متوجه شدن روند کد، می‌تونید از توضیحات زیر استفاده کنید. در صورتی هم که احساس نیاز نکردین، کافیه بخش‌هایی که تمام متنش با رنگ سبز نوشته شده رو بخونین.

سبزآبیه. سبز چیه؟

چه فرقی داره سبز با سبزآبی؟

چیییی؟ هیچی ولش کن فعلا ادامه‌ی دستورکار رو توضیح بدیم تا فرقشو بهت بگم بعدا.

باشه. اول بریم سراغ تابع gameloop... این تابع جزو توابعیه که براتون کاملش رو گذاشتیم. حالا چی‌کار می‌کنه؟ این منطق و کلیت بازیه که میاد و هر بار که نوبت یه نفره، از اول run میشه. اولش که تابع help رو صدا می‌کنه، برای کمک به بازیکن‌ها موقع شروع بازی هست و بعدش هم میاد و بُرد بازی رو با تابع create_board درست می‌کنه.
وقتی بُرد درست شد و راهنما نشون داده شد، دیگه موقعشه که بازی شروع بشه.

از این‌جا به بعد تا زمانی که سلول خالی وجود داره و می‌شه بازی ادامه پیدا کنه، توی یک حلقه تکرار می‌شه.
میاد می‌بینه نوبت کیه، از کاربر ورودی می‌گیره که می‌خواد کجا مهره‌شو بذاره و مهره رو توی جدول قرار می‌ده.
اگر هم کاربر بخواد مهره‌شو جایی که مجاز نیست بذاره، بهش اخطار میده که یه جای دیگه بذاره.

تابع بعدی help هست که اینم کامل کردیم براتون. این تابع کارش فقط چاپ کردن راهنماست.
بعد از اون تابع create_board هست که میاد و بُرد ما رو با کمک کاراکترهای space یعنی " " می‌سازه. به کمک دو تا for به اندازه‌ی سایز صفحه، همه‌ی خونه‌ها رو " " می‌کنه. نهایتا هم اون ۴ تا خونه‌ی وسط بازی رو به شکلی که توی قوانین بازی مشخص شده، مهره‌ی سفید و قرمز قرار می‌ده.

تابع draw اسم ستون‌ها و سطرهای بازی رو می‌نویسه (ستون‌ها با حروف انگلیسی شروع میشن و سطرها با اعداد) چالش اینجا اینه که سطرها رو که عدد هستند توی این آرایه‌مون که کاراکتره ذخیره کنیم. اینجا اومدیم و از کد اسکی استفاده کردیم.

تابع insert همونطور که از اسمش پیداست، مهره‌ای که کاربر انتخاب کرده رو توی صفحه قرار می‌ده که این‌جا هم چون ما قبلا اعداد سطر و ستون رو با حروف ذخیره کردیم، دوباره باید از اسکی کمک بگیریم تا به عدد تبدیل بشه و خونه‌ی مدنظر رو تو آرایه پیدا کنیم.

تابع occupied کوتاه ولی مهمه! این تابع چک می‌کنه ما مهره‌مون رو توی خونه‌ی درستی گذاشتیم؟ (چون همون‌طور که گفتیم، مهره‌ها رو فقط تو خونه‌های مشخصی می‌شه گذاشت و ما اون خونه‌ها رو با * مشخص می‌کنیم. اگه تو خونه‌ی درستی نباشه، error می‌ده تا کاربر خونه رو مجدد انتخاب کنه.

تابع is_placeable میاد چک می‌کنه که می‌شه توی اون خونه مهره گذاشت یا نه. طبق قوانین! قوانین می‌گفتن ما در صورتی می‌تونیم توی خونه مهره بذاریم که اطرافش (۴خونه‌ی ۴طرفش) یه مهره از حریف باشه.
تابع show_placable اولین تابع خالیه =)

نوبت منه دیگه عه.

بله تابع show_placable اولین تابع خالیه. این‌جا باید خودتون کامل کنید. چطوری؟ باید به کمک تابع قبلی یعنی is_placeable چک کنید اینجا کاربر مجازه مهره بذاره یا نه. اگه مجازه باید اون خونه تبدیل به * بشه اگر هم نیست که خب یعنی کاربر قبلا اونجا مهره گذاشته و طبیعتا نباید به * تبدیل بشه دیگه.

تابع بعدی reset هست که آخر قرار دادن هر مهره صدا زده میشه و برد reset میشه و اون خونه‌هایی که * بودن دوباره عادی میشن تا از اول برای کاربر جدید خونه‌های مجاز رو با * مشخص کنیم.

بعد از reset می‌ریم سراغ مهم‌ترین تابع بازی! تابع apply_changes که عملا یه تنه کل منطق بازیو مدیریت می‌کنه. این تابع میاد و متناسب با مهره‌ای که توی insert قرار گرفته تغییرات لازم رو می‌ده، یعنی چک می‌کنه که کدوم مهره‌ها با insert شدن این مهره‌ی جدید رنگشون تغییر می‌کنه، رنگ اون‌ها رو عوض می‌کنه و جدولو آپدیت می‌کنه. این تابع برای مدیریت بازی، باید هشت حالت مختلف رو بررسی کنه که توی کد شش حالتش بررسی شده. دو حالت بعدی رو از شما می‌خوایم که کامل کنین تا مدیریت زمین به طور کامل انجام بشه.

تابع ناقص بعدی update_scores هست که از روی اسمش واضحه چیکار باید بکنه دیگه. میاد و امتیازات رو متناسب با تغییراتی که اتفاق افتاده، آپدیت می‌کنه. امتیاز هر نفر هم برابر تعداد مهره‌هایی به اون رنگ توی جدول بازیه؛ که باید یکی یکی بشمارید و جمع امتیازش رو مشخص کنید. این امتیاز بعد از قرار دادن هر مهره، به کمک تابع بعدی که display_scores هست، بهش نشون داده می‌شه.

تابع آخر هم game_over عه که بعد از خارج شدن از شرط اولیه‌ی بازی (باقی موندن خونه‌ی خالی) صدا زده میشه و متناسب با امتیازات میاد مشخص می‌کنه کدوم کاربر برنده شده یا شایدم بازی مساوی شده اصلا! و نهایتا امتیازات رو چاپ می‌کنه.

همین! کل این کد طولانی و ترسناکی که براتون آماده کردیم، همین بود که تقریبا ۹۰ درصدش آماده‌س و شما باید اون ۱۰ درصد رو که ۲ تا تابع کوچیکن کامل کنید.
اما پیشنهاد ما به شما اینه که خودتون یه بار برنامه رو بنویسین تا دستتون تو کدهای طولانی‌تر راه بیفته. اگر جاییش رو حس کردین کمکی لازم داره می‌تونین به کد این برنامه مراجعه کنین. شاید هم برنامه‌ای که شما می‌نویسین از برنامه‌ی ما خیلی بهتر باشه.

امیدوارم که از بازی اتللو خوشتون اومده باشه. تا کارگاه بعدی خدانگه‌دار ☺

;