



جلسه یازدهم

حافظه

تخصیص

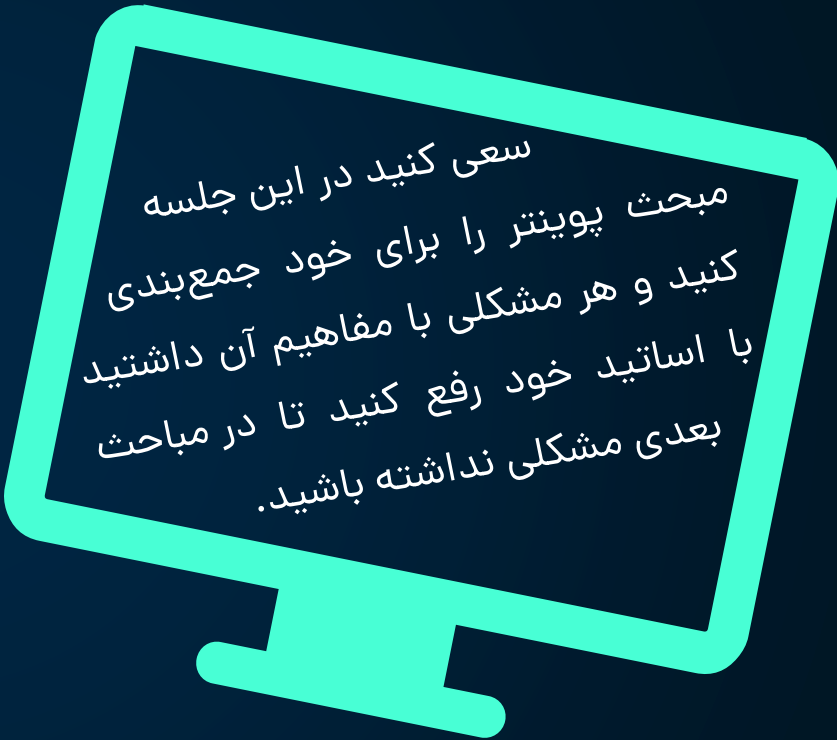
بسم الله الرحمن الرحيم



کارگاه مبانی برنامه‌نویسی - دانشکده مهندسی کامپیوتر دانشگاه امیرکبیر

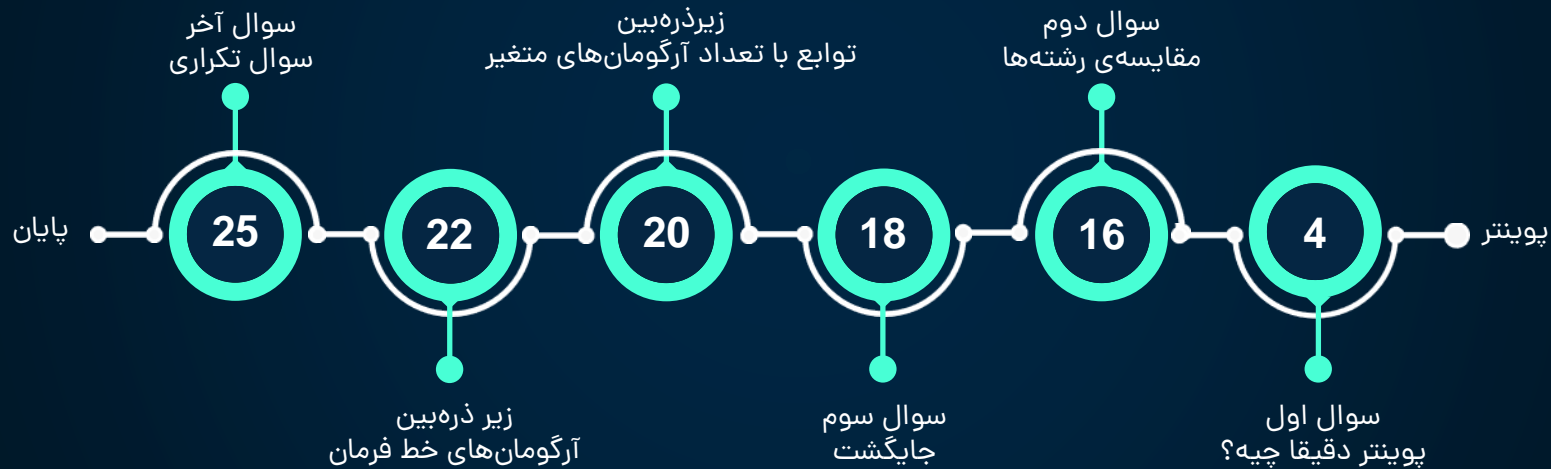
مبحث اشاره‌گرها و کار با آن‌ها، نیازمند تمرین و کد زدن‌هاش زیاد است تا حسابی جا بیوفتد. زیرا مباحث بعدی به درک عمیق و درست شما از مبحث پوینتر وابسته است.

به همین دلیل در این کارگاه همچنان به اشاره‌گرها و بررسی عمیق‌تر آن‌ها خواهیم پرداخت و یکی از کاربردهای مهم اشاره‌گرها در زمینه‌ی تخصیص حافظه را حسابی تمرین خواهیم کرد.



سعی کنید در این جلسه مبحث پوینتر را برای خود جمع‌بندی کنید و هر مشکلی با مفاهیم آن داشتید با اساتید خود رفع کنید تا در مباحث بعدی مشکلی نداشته باشید.

فهرست



سوال اول: پوینتر دقیقا چیه؟

سلام به همگی، امیدواریم حالتون خوب باشه. برنامه‌ی امروز یه کم با همیشه فرق داره. ما دیدیم توی دستورکار این کارگاه، سوالای مختلفی برای کار با پوینتر هست که شاید بهتر باشه قبل از دست به کد شدن و انجام اون‌ها، یکم دیدمون به پوینترها رو قوی‌تر کنیم. برای همین این جلسه با سوال اول در کنارتون هستیم.



بله... برای شروع، یکی از سوالای پر تکراری رو که اکثرا جز تمرین‌ها بوده، انتخاب کردیم تا با کمک هم‌دیگه اولاش رو بررسی کنیم و بعد که دستتون حساسی گرم شد، ادامه‌اش رو خودتون بررسی کنید.





خب، کدی رو که قراره با هم بررسیش کنیم، این کده:

```
int j = 0, i = 0, *p1;
int **p2, **p3;
p2 = p3 = (int **) malloc(3 * sizeof(int *));
```



یه کار عجیب می‌خوام ازتون. سریع برین

```
printf("%d\n", sizeof(int *));
```

کاغذ و قلم با خودتون بیارین.

```
do {
    *p2 = (int *) calloc(2 * (i + 1), sizeof(int));
    for(j = 1, p1 = (*p2) + 1; j < 2 * (i + 1); j++, p1++)
        *p1 += j + *(p1 - 1);
    p2++;
    i++;
} while(i < 3);
```

و تازه قول هم بدین که

برنامه رو اجرا نکنین.

می‌خوایم با هم تبدیل بشیم به یه کامپایلر و خودمون بگیم این

کد چی چاپ می‌کنه.

پس الان فقط یه اسکرین‌شات، عکسی، چیزی از صفحه بگیرین تا

```
p2 = p3;
```

با هم شروع کنیم به قدم قدم اجرا کردن کد روی کاغذ!

```
for(i = 2; i >= 0; i--)
    for(j = 0; j < 2 * (i + 1); j++)
        printf("[%d][%d] = %d\n", i, j, *((*(p2 + i))+j));
```

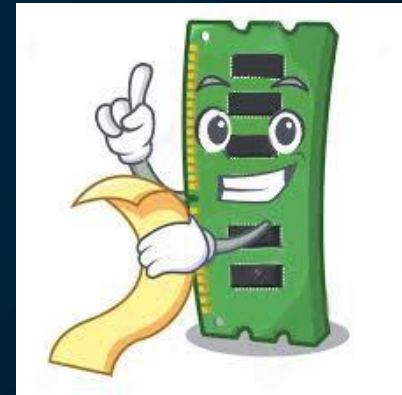


تو دو خط اول چی داریم؟

هر چی متغیر داریم با مقدار اولیه شون رو
کاغذ بنویسین.

اگرم مقدار اولیه نداره که فعلا هیچی فقط
اسمش رو بذارین باشه.

منم اینجا براتون با شکل حافظه متغیرها رو
می‌کشم...



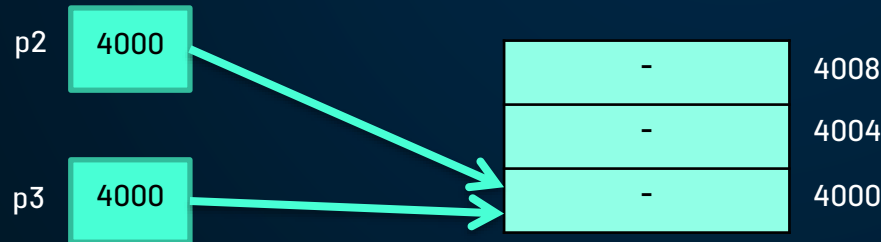
شماره سطر	اسم سطر (اسم متغیر)	نوع متغیر	مقدار متغیر در حافظه (بقیه ستون‌ها اطلاعات تکمیلی هستن)
2^n-1
...
9400	p3	int **	-
...
9004	p2	int **	-
9000	p1	int *	-
...
...
3600	j	int	0
...
2000	i	int	0
...
1
0



خب رسیدیم به دستور malloc. بیاین یهو malloc و calloc رو با هم بگیم و مقایسه کنیم. اولین فرق ظاهری شون که باید بدونین اینه که malloc یه دونه ورودی می‌گیره ولی calloc دو تا. در اصل به malloc می‌گیم **چقدر فضا** می‌خوایم، اونو بهمون بده. به calloc می‌گیم ما **چند** تا خونه با **فضای دلخواه** می‌خوایم. لطفا این رو برای ما در نظر بگیر. دومین فرقشون هم اینه که malloc اون فضا رو برای ما می‌گیره ولی دیگه مقداری توشون نمی‌ذاره تا ما بگیم. اما calloc تموم خونه‌هایی که گرفته رو با 0 مقداردهی اولیه می‌کنه.



حالا خط سوم داره به malloc می‌گه به اندازه‌ی 3 تا **int *** برای من حافظه در نظر بگیر. بعد آدرس شروعش رو که داری میدی، اول به **int **** تبدیلش کن (چون تایپ متغیر هامون **int **** هست) و بعد هم مقدار p2 و p3 رو برابر اون قرار بده. پس انگار p2 و p3 به شروع این بخش اشاره می‌کنند.



شماره سطر	اسم سطر (اسم متغیر)	نوع متغیر	مقدار متغیر در حافظه (بقیه ستون‌ها اطلاعات تکمیلی هستند)
2 ⁿ -1
9400	p3	int **	4000
...
9404	p2	int **	4000
9000	p1	int *	-
...
4008	...	int *	-
4004	...	int *	-
4000	*p3 یا *p2	int *	-
...
3600	j	int	0
...
2000	i	int	0
...

تو خط چهارم سائز **int *** رو چاپ کرده که اندازه‌اش برابر 4 هست یعنی 4 بایت. و به همین دلیل هم هست که ما سه تا خونه‌ای که به کمک malloc گرفته شد رو 4بایت 4بایت شماره‌گذاری کردیم.



پس این خونه شد شروع بخش malloc شده‌ی کدمون.



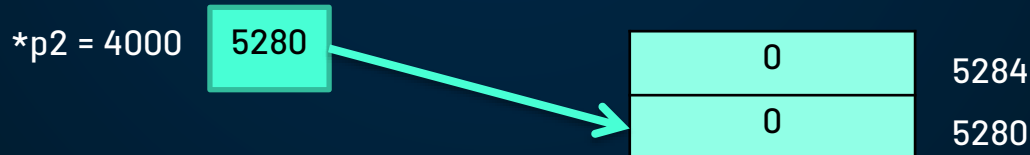

```
do {
    *p2 = (int *) calloc(2 * (i + 1) , sizeof(int));
    for(j = 1, p1 = (*p2) + 1; j < 2 * (i + 1); j++, p1++)
        *p1 += j + *(p1 - 1);
    p2++;
    i++;
} while(i < 3);
```

حالا رسیدیم به بخش اجقوجق کد (((=



بر خلاف قیافه‌ی ترسناکش اگه تیکه تیکه بریم جلو خیلی راحت می‌شه.

مقدار i که برابر 0 بود. بنابراین این‌جا تابع `calloc` برای ما 2 تا خونه به سایز `int` در نظر می‌گیره و آدرس شروع آنها رو به $*p2$ می‌ده. در درس فرق $p2$ و $*p2$ تدریس شده، ولی برای یادآوری می‌گم که $p2$ توی خودش یه آدرس رو نگهداری می‌کنه. اگه مقدار خودش رو عوض کنیم، اونوقت به یه جای دیگه اشاره می‌کنه. اما اگه $*p2$ رو عوض کنیم، در حقیقت با محتوای خونه‌ای که $p2$ بهش اشاره می‌کنه کار داریم.





بچه‌ها برای این‌که اطلاعات تو شکل جا بشه یکم فشرده‌تر کردم‌شون و برای همین ممکنه ببینین که p1 و p2 پشت هم قرار گرفتن، در حالی که شماره‌ی آدرس‌هاشون می‌گه کلی فاصله بین‌شونه. اینو به بزرگی خودتون ببخشین جا نمی‌شد:

اطلاعات جدید رو با رنگ مشکی یا قرمز نشون دادیم.



تو صفحه‌ی بعد می‌ریم سراغ for نسبتاً ترسناکی که داریم. ادامه‌ی داستان رو دیگه من و Botfather صحبت نمی‌کنیم. شما فقط با دنبال کردن شکل‌ها و توضیحات استادتون این بخش رو پیش می‌برید.

شماره سطر	اسم سطر (اسم متغیر)	نوع متغیر	مقدار متغیر در حافظه (بقیه ستون‌ها اطلاعات تکمیلی هستن)
9400	p3	int **	4000
9404	p2	int **	4000
9000	p1	int *	-
...
...
5284	...	int	0
5280	...	int	0
...
4008	...	int *	-
4004	...	int *	-
4000	*p3 یا *p2	int *	5280
...
3600	j	int	0
2000	i	int	0

شماره سطر	اسم سطر (اسم متغیر)	نوع متغیر	مقدار متغیر در حافظه (بقیه ستون‌ها اطلاعات تکمیلی هستند)
9400	p3	int **	4000
9404	p2	int **	4000
9000	p1	int *	5284
...
...
5284	...	int	0
5280	...	int	0
...
...
...
...
4008	...	int *	-
4004	...	int *	-
4000	*p3 یا *p2	int *	5280

3600	j	int	1
2000	i	int	0

```
for(j = 1, p1 = (*p2) + 1; j < 2 * (i + 1); j++, p1++)
    *p1 += j + * (p1 - 1);
```

شماره سطر	اسم سطر (اسم متغیر)	نوع متغیر	مقدار متغیر در حافظه (بقیه ستون‌ها اطلاعات تکمیلی هستند)
9400	p3	int **	4000
9404	p2	int **	4000
9000	p1	int *	5284
...
...
5284	...	int	0 \rightarrow (0+1) = 1
5280	...	int	0
...
...
...
...
4008	...	int *	-
4004	...	int *	-
4000	*p3 یا *p2	int *	5280

3600	j	int	1
2000	i	int	0

for(j = 1, p1 = (*p2) + 1; j < 2 * (i + 1); j++, p1++)
 *p1 += j + *(p1 - 1);

شماره سطر	اسم سطر (اسم متغیر)	نوع متغیر	مقدار متغیر در حافظه (بقیه ستون‌ها اطلاعات تکمیلی هستند)
9400	p3	int **	4000
9404	p2	int **	4000
9000	p1	int *	5288
...
...
5284	...	int	1
5280	...	int	0
...
...
...
...
4008	...	int *	-
4004	...	int *	-
4000	*p3 یا *p2	int *	5280

3600	j	int	2
2000	i	int	0

for(j = 1, p1 = (*p2) + 1; j < 2 * (i + 1); j++, p1++)
 *p1 += j + *(p1 - 1);



بخش for هم انجام شد. فقط مونده $p2$ و i هر کدوم یکی مقدارشون اضافه بشه. بعد چون شرط $3 < i$ برقراره دوباره کل این بخش به ازای $i = 1$ انجام می‌شه.



در شکل صفحه‌ی بعد آخرین وضعیت ما که با هم بررسی کردیم رو می‌بینین. از اینجا به بعدش رو شما پیش ببرین تا ببینیم در نهایت این برنامه چه چیزی رو چاپ می‌کنه. بعدش برنامه رو run کنید و ببینید آیا کامپایلر خوبی هستین یا نه :))




اگر احساس می‌کنید که سطح سوال برای شروع بالا بوده، بله کاملاً درست فکر می‌کنید (=) اما مطمئن باشید بعد از تکمیل سوال به درک خوبی از پوینترها و همچنین توابع malloc و calloc رسیدین. پس یکم این اولش رو به خودتون سخت بگیرید تا بقیه تمرین‌ها و سوالا براتون مثل آب خوردن بشه.




خب ما اینجا ازتون خداحافظی می‌کنیم. به امید دیدار تا کارگاه بعد ☺

شماره سطر	اسم سطر (اسم متغیر)	نوع متغیر	مقدار متغیر در حافظه (بقیه ستون‌ها اطلاعات تکمیلی هستند)				
9400	p3	int **	4000	3600	j	int	2
9404	p2	int **	4004	2000	i	int	1
9000	p1	int *	5288				
...				
...				
5284	...	int	1				
5280	...	int	0				
...				
...				
...				
...				
4008	...	int *	-				
4004	...	int *	-				
4000	*p3 یا *p2	int *	5280				

سوال دوم: مقایسه‌ی رشته‌ها

در دنیای روزمره بسیار مشاهده می‌کنیم که ذهن ما در حال مقایسه است، پس می‌توانیم حدس بزنیم که احتمالاً در دنیای برنامه‌نویسی هم به توابعی برای مقایسه نیاز داریم. 

یکی از این توابع، تابعی است که به ما کمک می‌کند تا دو string را با هم مقایسه کنیم و ببینیم که آن‌ها عیناً یکی هستند یا نه. 

به نظر شما کاربرد این تابع چیست؟ 
وقتی می‌خواهیم که از یک لیست یک اسم خاص را بیابیم این تابع چه کمکی به ما می‌کند؟



احتمالا از قدیمی‌ترها دیده‌اید (یا شاید هم برای خودتان اتفاق افتاده) که وقتی دنبال یک اسم خاص در دفترچه تلفن می‌گردند، بلند بلند آن اسم را تکرار می‌کنند و صفحه‌های دفترچه را ورق می‌زنند تا آن را پیدا کنند. حالا می‌خواهیم برنامه‌ای بنویسیم که دقیقا همین کار را برای راحتی کار آن‌ها انجام دهد.



بنابراین شما برنامه‌ای بنویسید که دو رشته را در ورودی دریافت کند و برابر بودن یا نبودن آن‌ها را در نهایت گزارش دهد.



دو رشته را عینا شبیه هم وارد کنید، اما با این تفاوت که یک جمله با حرف‌های uppercase باشد و دیگری با حرف‌های lowercase.

به نظر شما خروجی تابع چه خواهد بود؟ آیا کد هم با شما هم‌نظر است؟



با توجه به پرکاربرد بودن عملیات مقایسه، زبان C تابع آماده‌ای برای آن دارد. بعد از نوشتن برنامه‌ی خود، سعی کنید آن را پیدا کنید و نحوه‌ی کارش را با کد خود مقایسه کنید.

سوال سوم: جایگشت

فرض کنید قرارست یک سری مسابقات برنامه‌نویسی در دانشگاه برگزار شود. شما وظیفه دارید که به عنوان یکی از اعضای کادر اجرایی این مسابقات، اطلاعات تیم‌ها را ذخیره کنید و به صورت تصادفی یک ترتیب برای انجام مسابقات بین تیم‌ها انتخاب کنید.

برای این کار باید برنامه‌ای بنویسید که به طور تصادفی یک ترتیب خاص را بین جایگشت‌های مختلف تیم‌ها انتخاب کند.

در ابتدا نیاز دارید که اطلاعات تیم‌ها را در ورودی دریافت کنید. از آن‌جا که مشخص نیست چند تیم قرارست در این مسابقه شرکت کند، شما نیاز دارید که این کار را به کمک تخصیص حافظه انجام دهید.



پس شما به کمک تابع‌هایی مثل malloc و یا calloc که در ابتدای کارگاه هم با آن‌ها کار کردید، باید هر بار یک string به عنوان نام تیم به اطلاعات قبلی ذخیره شده اضافه کنید.



پس از دریافت ورودی‌ها، تمامی جایگشت‌های مختلف تیم‌ها را بسازید، به هر کدام از جایگشت‌ها یک شماره اختصاص دهید و آن‌ها را چاپ کنید تا شرکت‌کنندگان اطمینان پیدا کنند که شما تمامی حالت‌های مختلف را ایجاد کرده‌اید.



در انتها با تولید یک عدد تصادفی مشخص خواهد شد که کدام جایگشت انتخاب شده و شما باید آرایه‌ی اولیه را با توجه به ترتیب انتخاب شده دوباره مرتب کنید. فرض کنید سیستمی که با آن این برنامه نوشتید، حافظه‌ی محدودی دارد پس نباید برای این بخش دوباره حافظه تخصیص دهید و تغییرات را در آن‌جا اعمال کنید. بلکه باید همان اطلاعات اولیه را جوری تغییر دهید که ترتیب رشته‌ها مشابه ترتیب منتخب باشد.

زیر ذره بین: توابع با تعداد آرگومان های متغیر^۱

گاهی در هنگام تعریف یک تابع نمی دانیم که چند آرگومان ورودی خواهیم داشت و علاقه مندیم تابعی بنویسیم که در زمان اجرا بتواند تعداد متفاوتی از آرگومان های ورودی را دریافت کند. در جلسات قبل کارگاه هم دیدید که تعریف این گونه توابع در زبان C امکان پذیر است. مانند مثال زیر:

```
void function (int a, int b, ...);
```

به کاربرد ... در این تابع توجه کنید. همچنین لازم به ذکر است که تمامی آرگومان هایی که قبل از ... باشند، برای فراخوانی تابع لازم اند. در این صورت هر گونه فراخوانی این تابع که دو عدد یا بیشتر از آن آرگومان داشته باشد صحیح است اما در داخل تابع فقط دسترسی به آرگومان های اول و دوم امکان پذیر است و باقی آرگومان ها در نظر گرفته نمی شوند.

در بعضی موارد لازم است که به تمامی آرگومان‌های ورودی یک تابع با تعداد آرگومان‌های متغیر دسترسی داشته باشیم. این کار در زبان C با استفاده از کتابخانه‌ای به نام `stdarg.h` امکان پذیر است.



<stdarg> (stdarg.h)

<https://b2n.ir/587466>


این کتابخانه همچنین امکاناتی برای کار با این گونه توابع فراهم می‌سازد که نمونه‌ای از آن را می‌توانید در قطعه کد زیر که تابعی برای محاسبه‌ی میانگین تعداد نامعلومی از اعداد را نشان می‌دهد مشاهده کنید.


```
double average (int count, ...) {
    va_list nums;
    va_start(nums, count); /* Requires the last fixed parameter */

    double sum = 0;
    for(int i = 0; i < count; i++) {
        sum += va_arg(nums, int); /* Increment nums to the next argument */
    }

    va_end(nums);
    return sum / count;
}
```

زیر ذره بین: آرگومان‌های خط فرمان^۱

آرگومان‌های خط فرمان آرگومان‌هایی هستند که در زمان اجرا توسط سیستم عامل به برنامه منتقل می‌شوند و برنامه در صورت نیاز می‌تواند از آن‌ها استفاده کند. این آرگومان‌ها که با یک کاراکتر فاصله از یکدیگر جدا شده‌اند، هنگام اجرای برنامه و در خط فرمان بعد از نام پرونده اجرایی وارد می‌شوند. 

در زبان‌های C و C++ برای آن که بتوان در برنامه به این آرگومان‌ها دسترسی پیدا کرد، از آرگومان‌های تابع main استفاده می‌شود و این تابع باید به این صورت نوشته شود: 

```
int main(int argc, char const *argv[])
```



در این حالت، `argc` یک عدد صحیح است که تعداد آرگومان‌های ورودی به برنامه از طریق خط فرمان را نشان می‌دهد. توجه کنید که حداقل مقدار برای `argc` یک است؛ زیرا دستور اجرای برنامه (نام پرونده اجرایی) حتماً در زمان اجرای برنامه مورد استفاده قرار می‌گیرد و به عنوان اولین آرگومان خط فرمان وارد برنامه می‌شود. همچنین `argv` آرایه از رشته‌های مدل زبان C است که دربردارنده‌ی تمامی آرگومان‌های ورودی به برنامه است.



به مثال زیر توجه کنید:

```
#include <stdio.h>

int main(int argc, char const *argv[]) {
    printf("There are %d command line arguments\n", argc);

    for(int i = 0; i < argc; i++) {
        printf("Argument %d: %s\n", i + 1, argv[i]);
    }

    return 0;
}
```

فرض کنید پرونده اجرایی برنامه بالا با نام application.exe در محل فعلی قرار دارد. دستور ورودی به خط فرمان برای اجرای این برنامه و خروجی برنامه به شرح زیر است.

```
...>application.exe Amirkabir University  
There are 3 command line arguments  
Argument 1: application.exe  
Argument 2: Amirkabir  
Argument 3: University
```

پیوست: اگر در محیط Visual Studio کار می کنید، برای آشنایی با نحوه پاس دادن آرگومان های خط فرمان به بخش Command line arguments در لینک زیر مراجعه کنید.



Command line arguments

<https://b2n.ir/755375>

سوال آخر: سوال تکراری

یافتن بزرگ‌ترین عدد بین چند عدد...

اما این بار با استفاده از تخصیص حافظه‌ی پویا یا

dynamic memory allocation

به تصویر روبه‌رو نگاه کنید:

همانطور که مشخص است می‌خواهیم بزرگ‌ترین

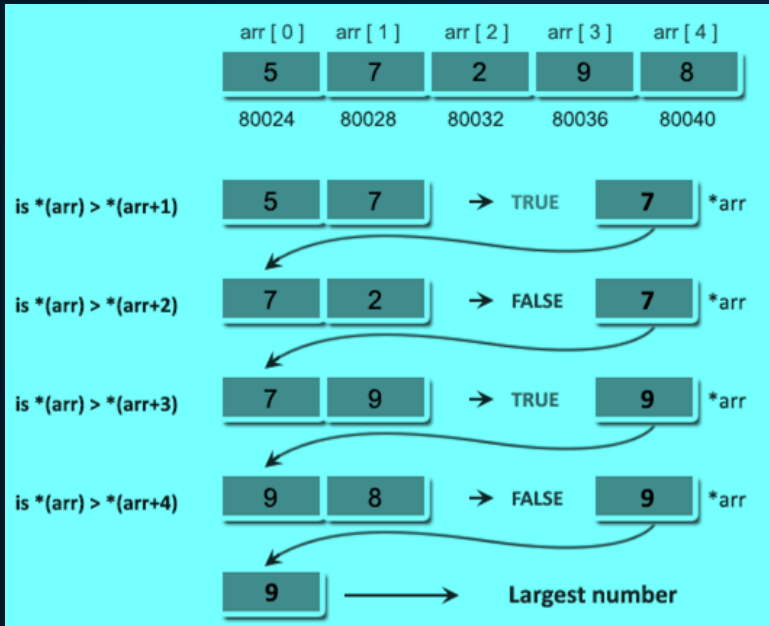
عدد بین چند عدد را بیابیم. اما دسترسی ما به

این اعداد تنها به کمک اشاره‌گرها قابل انجام

است. پس با توجه به این نکته برنامه‌ای بنویسید

که تعداد نامشخصی از اعداد را دریافت می‌کند و

بزرگ‌ترین آن‌ها را به ما نشان می‌دهد.



;