ساختار حافظه کامپیوتر :

- هرجا صحبت از حافظه کامپیوتر می‌کنیم، منظورما RAM هست و نه HDD.

RAM : Random access memory

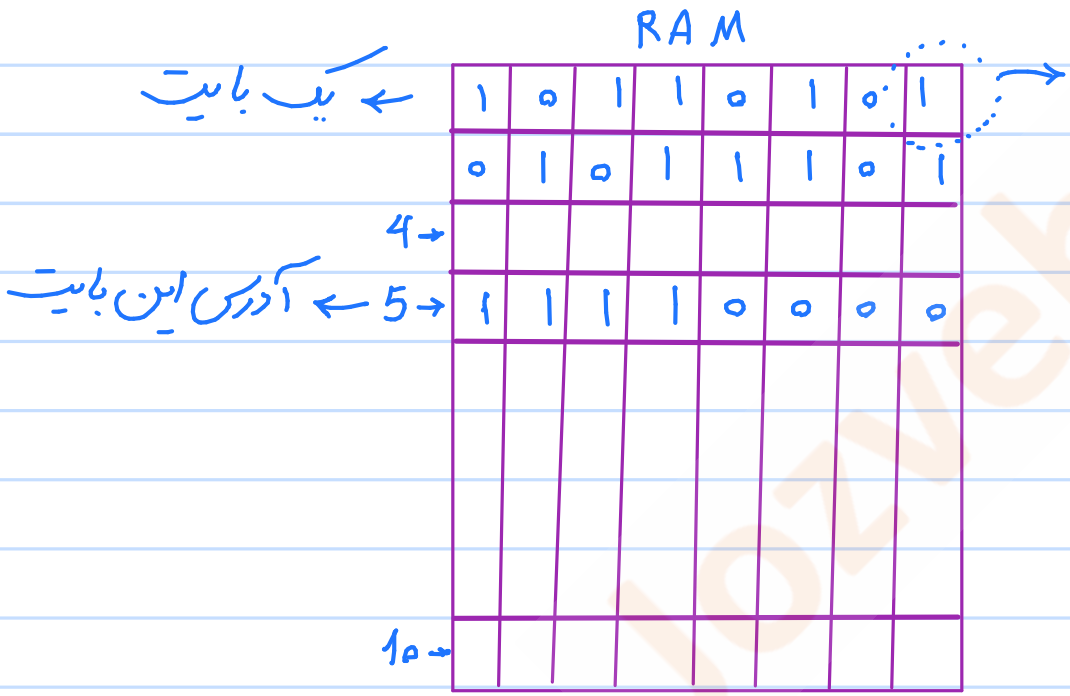
main.c —compiler→ main.exe —اجرا→ کد کامپایل شده به RAM منتقل می‌شود

→ خط به خط کد را می‌خواند و اجرا می‌کند cpu

RAM چه ساختاری دارد و برنامه و متغیرها را چگونه در آن ذخیره می‌شوند؟

افراد	اتاق‌ها	خانه‌ها	شهر
۰ یا ۱	Bit	Byte	RAM

اطلاعات باینری



هر بایت حاوی ۸ بیت است و یک واحد حافظه در RAM را تشکیل می‌دهد.

در شهر هر خانه یک آدرس یا کد پستی منحصر به فرد دارد.

در RAM هم هر بایت آدرس مخصوص خود را دارد.

اگر اداره پست شهر بگوید که حداکثر بسته قابل قبول فقط 3 رقم است، در آن صورت فقط 1000 شهر می‌تواند خانه داشته باشد.

آدرس هر بایت در ram نهایتاً باید در 32 بیت جا شود. ← 32 bit windows

آدرس هر بایت در ram نهایتاً باید در 64 بیت جا شود. ← 64 bit windows

سوال: حداکثر RAM قابل آدرس دهی در هر نسخه ویندوز چقدر است؟

32 بیت :

تعداد حالات : $2^{32} = 2^{10} \cdot 2^{10} \cdot 2^{10} \cdot 4 = 4G$ (K) ⇒ ماکزیمم RAM خواهد بود 4GB

64 بیت : تعداد حالات : 2^{64} ⇒ دیگر محدودیت آدرس دهی نداریم

این محدودیت های دیگر عمدتا به ram در ویندوز 10 Home edition است 128GB .

حالا آدرس دهی باید در ram به صورت 64 تا صفر و یک ذخیره شود که خوانایی آن را بسیار پایین می آورد.

به این دلیل اکثرا از فرمت Hex استفاده می کنیم.
که مبنای 16

تبدیل اعداد باینری به Hex :

Binary	Hex
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

مثال :

آدرس باینری 32بیت :

11011000 00111110 11010100 00010011

آدرس Hex :

0XD83ED413

نشان دهنده این است که عدد درفرمت Hex است

موادی که از زبان C خواهیم آموخت :

بخش ۱ - متغیرها

1. تعریف متغیرهای عددی (اعداد صحیح بدون علامت، با علامت و اعشاری) با ظرفیت‌های گوناگون
2. تعریف متغیرهای کاراکتری و ذخیره سازی آن‌ها
3. تعریف متغیر boolean - اپراتورهای منطقی - ایجاد عبارات منطقی - ترکیب عبارات منطقی
4. تعریف آرایه‌ها و ذخیره سازی آن‌ها - آشنایی با sizeof()
5. تعریف آرایه‌های کاراکتری و رشته‌های کاراکتری (string)
6. آرایه‌های چند بعدی

بخش 2 - کنترل برنامه

1. آشنایی با دستور if ، if_else ، if_else if_else
2. آشنایی با while
3. آشنایی با for
4. آشنایی با switch case
5. آشنایی با do while
6. آشنایی با break ، continue

بخش 3 ـ گرفتن ورودی از کاربر و چاپ خروجی‌ها

1. آشنایی با printf ـ format specifier‌ها و escape character‌ها
2. آشنایی با getchar()
3. آشنایی با gets()
4. آشنایی با scanf()
5. آشنایی با fgets()
6. آشنایی با sscanf()

بخش 4- توابع

1. تعریف تابع - انواع ورودی ها و خروجی ها
2. درک نحوه عملکرد تابع
3. اشاره گرها و کاربرد آنها در توابع : pass by value , pass by reference
4. آرایه ها به عنوان ورودی و خروجی توابع

بخش 5 - اشاره گرها

1. آدرس متغیرها و مشاهده آدرس ها
2. ایجاد متغیرهایی برای ذخیره آدرس دیگر متغیرها
3. کاربرد اشاره گرها
4. تعریف آرایه ها با استفاده از اشاره گرها
5. کار با حافظه و تخصیص دینامیک حافظه

بخش 6 - مفاهیم پیشرفته‌تر

1. structures
2. unions
3. enums
4. malloc , alloc
5. مدیریت حافظه و ...

ذخیره اعداد صحیح مثبت در حافظه:

اول اینکه تمام انواع داده (اعداد صحیح، اعداد اعشاری، رشته‌های کاراکتری و ...) به صورت 0 و 1 در ram ذخیره می‌شوند.

ساده‌ترین نوع داده، یک عدد صحیح مثبت است. ابتدا به این می‌پردازیم که هر عدد در مبنای 10 را چگونه با 0 و 1 (در مبنای 2) بیان کنیم

یک عدد فرضی در مبنای 2 : 1 0 1 1 0 1 1

ارزش هر جایگاه : $2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

عدد معادل در مبنای 10 : $\underbrace{1\times2^0}_{1}+\underbrace{1\times2^1}_{2}+\underbrace{1\times2^3}_{8}+\underbrace{1\times2^4}_{16}+\underbrace{1\times2^6}_{64}=91$

عدد در مبنای 10 : 91

تبدیل به باینری :

$91 \div 2 = 45$ ① کم اهمیت‌ترین بیت (LSB)

$45 \div 2 = 22$ ①

$22 \div 2 = 11$ ⓪

$11 \div 2 = 5$ ①

$5 \div 2 = 2$ ①

$2 \div 2 = 1$ ⓪

$1 \div 2 = 0$ ① → پراهمیت‌ترین بیت (MSB)

$\Rightarrow 1011011$

عدد در مبنای ۱۰ : 145

تبدیل به Hex : $145 \div 16 = 9$ ① $\Rightarrow$ 0x91 $\longrightarrow$ 10010001 باینری

$9 \div 16 = 0$ ⑨

عدد در مبنای 16 : 0XD3E3

تبدیل به مبنای ده (Decimal) : $3 \times 16^0 + 14 \times 16^1 + 3 \times 16^2 + 13 \times 16^3 = 54243$

3×16^0 = 3 ، 14×16^1 = 224 ، 3×16^2 = 768 ، 13×16^3 = 53248

نوع داده unsigned int در زبان C :

```
unsigned int a = 5 ;
```

- unsigned int → نوع متغیر یک عدد صحیح بدون علامت است.
- a → نام متغیر
- ; → در پایان تمام دستورات در C ضروری است

هنگام اجرای این دستور، 4 بایت فضا در حافظه RAM به متغیر a اختصاص پیدا می‌کند و عدد 5 تبدیل به باینری شده و در این 4 بایت ذخیره می‌شود.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1

← آدرس شروع متغیر a

5 → 0101 (Binary)

سوال : بزرگترین عددی که نوع متغیر unsigned int می تواند در خود نگه دارد چقدر است ؟

بزرگترین عدد = $2^{32} - 1 = 4\,294\,967\,295$ ⇐ 32 بیت ⇐ 4 بایت

آشنایی با تعریف متغیر در C ، دستور Printf ، چاپ عدد و آدرس عدد و تابع sizeof

```
printf ("value of a is %u.\n", a);
```

* هر آنچه بین " قرار می‌گیرد چاپ می‌شود.

* عباراتی که با % شروع می‌شوند format specifier نام دارند و کامپایلر C به جای آن‌ها، متغیرها و عباراتی که بعد از " می‌آیند را جایگزین می‌کند. برای چاپ اعداد صحیح بدون علامت از %u استفاده می‌کنیم.

* عباراتی که با \ شروع می‌شوند escape sequence نام دارند و برای C معنی خاصی می‌دهند. \n یعنی برو به خط بعد.

```
unsigned int a;
```
variable declaration (تعریف متغیر)

```
a = 5;
```
variable assignment (تخصیص متغیر)

```
unsigned int a = 5;
```
تعریف و تخصیص

تعریف هر متغیر فقط یک بار اتفاق می‌افتد. اما تخصیص آن می‌تواند هر چند بار باشد.

```
unsigned int a = 2;
unsigned int a = 7; => ERROR
a = 43; ✓✓
```

انواع دیگر داده برای ذخیره اعداد صحیح بدون علامت :

unsigned char → 1 Byte 0 255

unsigned short → 2 Byte 0 65535

unsigned long → 4 Byte 0 4294967295

unsigned long long → 8 Byte .....

ممکن است در یک کامپیوتر دیگر سایز این انواع داده متفاوت باشد. اگر حتماً بخواهیم سایز مورد نظرا داشته باشیم

uint8_t , uint16_t , uint32_t , uint64_t از کتابخانه stdint.h استفاده می‌کنیم.

خلاصه جلسه قبل

1. کوچکترین واحد حافظه در کامپیوتر bit نام دارد که صرفاً ۱ یا ۰ را می‌تواند ذخیره کند.
2. هشت bit در کنار هم یک بایت (Byte) تشکیل می‌دهند.
3. RAM حاصل قرارگرفتن این Byte ها کنار هم است. هر بایت در RAM آدرس مخصوص به خود را دارد.
4. تمام اطلاعات (اعداد، کاراکترها و ...) برای ذخیره باید تبدیل به ۱ و ۰ شوند.
5. اعداد در مبنای ۱۰، ابتدا به مبنای 2 برده شده و سپس در حافظه ذخیره شوند.
6. این اعداد می‌توانند در ۱، 2، 4 یا 8 بایت ذخیره شوند (وابسته به بزرگی عدد)

7 تعریف متغیر در C :

تعریف | تخصیص

نوع متغیر نام متغیر = مقدار متغیر ;

نوع متغیر: داده عدد بدون علامت است / یا علامت دار است / یا اعشاری است / یا کاراکتر است و چند خانه حافظه برای آن تخصیص پیدا کند

نام متغیر:
- % ✓
- num ✓
- number1 ✓
- firstName ✓
- last_Name ✓
- _name ✓
- 1 ✗
- num-1 ✗
- 1name ✗

خلاصه اعداد صحیح غیرمنفی:

تعریف متغیر باید متناسب با حداکثر بازه تغییرات آن متغیر باشد.

unsigned int a;	4 Byte = 32 bit	$a: [0, 2^{32}-1]$
unsigned char b;	1 Byte = 8 bit	$b: [0, 2^{8}-1] = [0, 255]$
unsigned short c;	2 Byte = 16 bit	$c: [0, 2^{16}-1]$
unsigned long long d;	8 Byte = 64 bit	$d: [0, 2^{64}-1]$

دستور printf :

برای نمایش اطلاعات در console ← printf

```
printf("this will be displayed");
```
⇒
```
this will be displayed
```

Escape sequence :

خروجی لازم :

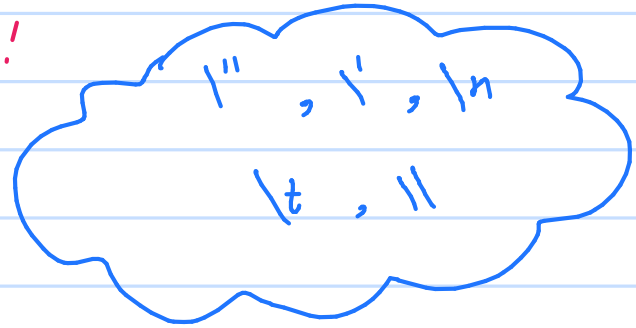
```
this is "c" programming course
```

```
this is 'c' programming course
```

```
My favorite numbers are : 3\7
```

```
Hello
world
```

several names each on 1 line



Format specifier :

```
unsigned int a = 10;
printf("Number a is equal to: %u", a);
```

وقتی کامپایلر به این می رسد، دنبال اولین متغیر بعد از علامت % می گردد و آن را با فرمت عدد صحیح غیرمنفی چاپ می کند.

برای اعداد صحیح غیرمنفی format specifier :

%u , %lu %llu

برنامه ۱: اعداد صحیح از صفر تا ۱۵ را به ترتیب هرکدام در یک خط چاپ کند.

الگوریتم:

1. شروع
2. متغیر x را برابر صفر قرار بده
3. x را چاپ کن
4. x را یکی زیاد کن
5. اگر x بزرگتر از ۱۵ شد برو به خط 6، در غیر اینصورت برو به خط 3
6. پایان

یکی دیگر از عباراتی که در الگوریتم نوشتن آن مجاز است عبارتی نظیر مثال زیر است :

③ تا وقتی که $i<10$ است، خط ④ تا ⑦ را انجام بده ⇒ حلقه

الگوریتم قبل با استفاده از این عبارت :

① شروع
② $i=0$
③ تا وقتی که $i<10$ است خط ④ و ⑤ را اجرا کن
④ i را چاپ کن
⑤ به i یک واحد اضافه کن
⑥ پایان

⇐ یک حلقه را تعریف می کنند.

برنامه 2 : برنامه‌ای که بررسی کند آیا یک عدد دریافتی اول است یا خیر.

$X \div 2 \quad \ldots\ldots \quad X \div (X-1)$

الگوریتم :

1. شروع
2. عدد x را از کاربر بگیر (فعلاً فرض می‌کنیم x یک عدد مشخص است)
3. div=2
4. متغیر rem را تعریف کن
5. تا وقتی که div<x است ، خط 6 تا 8 را اجرا کن
6. rem=x%div
7. اگر rem برابر صفر است از حلقه خارج شو ← خروج از حلقه
8. div=div+1
9. اگر rem صفر است عبارت not prime ، در غیر اینصورت prime را چاپ کن
10. پایان

ذخیره اعداد منفی :

روش اول : استفاده از بیت علامت

بیت علامت : اگر صفر باشد عدد مثبت و اگر ۱ باشد عدد منفی است.

ایرادات : اولاً دو صفر داریم.

ثانیاً جمع و تفریق اینها جواب درست را به دست نمی‌دهد.

$5-3=5+(-3)$:

```
  1 1 1
  0 1 0 1
+ 1 0 1 1
---------
1 0 0 0 0 →
×
```

معادل صفر است!

0 0 0 0	0	1 0 0 0	-0
0 0 0 1	1	1 0 0 1	-1
0 0 1 0	2	1 0 1 0	-2
0 0 1 1	3	1 0 1 1	-3
0 1 0 0	4	1 1 0 0	-4
0 1 0 1	5	1 1 0 1	-5
0 1 1 0	6	1 1 1 0	-6
0 1 1 1	7	1 1 1 1	-7

روش دوم : One's complement (مکمل ۱)

عدد		مکمل ۱	
0000	0	1111	-0
0001	1	1110	-1
0010	2	1101	-2
0011	3	1100	-3
0100	4	1011	-4
0101	5	1010	-5
0110	6	1001	-6
0111	7	1000	-7

دوباره دو صفر داریم. بازهم جمع و تفریق جواب درست را نمی دهد.

5 - 3 :

$$\begin{array}{r} {\scriptstyle 1} \\ 0101 \\ +\ 1100 \\ \hline 10001 \end{array}$$

معادل ۱ است. → $\overset{1}{\times}0001$

روش سوم: Two's complement (مکمل 2)

عدد	مکمل 2	
0000	0000	0
0001	1111	$-1 = -8+7$
0010	1110	$-2 = -8+6$
0011	1101	$-3 = -8+5$
0100	1100	$-4 = -8+4$
0101	1011	$-5 = -8+3$
0110	1010	$-6 = -8+2$
0111	1001	$-7 = -8+1$
	1000	$-8 = -8+0$

$-2^3\ 2^2\ 2^1\ 2^0$

تمام ایرادات قبلی رفع می‌شوند.

$5-3$:

$$\begin{array}{r} 0101 \\ +\ 1101 \\ \hline 10010 \\ \times \end{array} \equiv 2$$

توسط 4 بیت اعداد علامت‌دار

در بازه $[-8, 7)$ قابل بیان هستند.

تعریف اعداد صحیح علامت دار در C :

char x; 1 Byte → 8 bit → $[-2^7, 2^7-1] = [-128, 127]$
int8_t x;

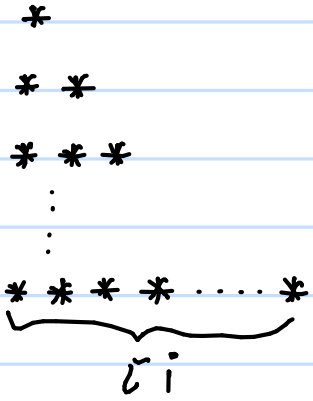
short x; 2 Byte → 16 bit → $[-2^{15}, 2^{15}-1] = [-32768, 32767]$
int16_t x;

int x; → 4Byte → 32 bit → $[-2^{31}, 2^{31}-1] = [-2,147,483,648 , 2,147,483,647]$
int32_t x;

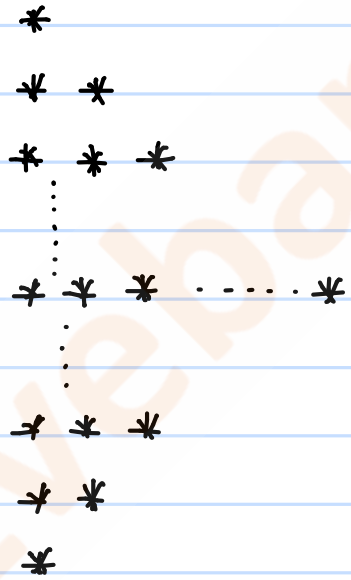
long long int x; → 8 Byte → 64 bit → $[-2^{63}, 2^{63}-1]$
int64_t x;

فرمت specifier برای اعداد علامت دار hhi یا i است. برای اعداد 64 بیتی از hhi یا i استفاده می‌کنیم.

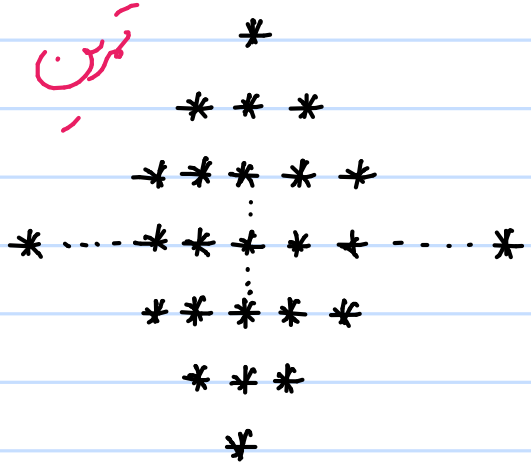
برنامه 3 : برنامه‌ای بنویسید که یک عدد n را گرفته و الگوی زیر را چاپ کند :



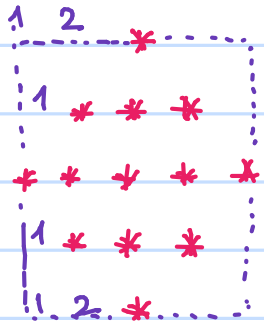
تمرین (الگوریتم و برنامه)



تمرین (الگوریتم و برنامه)



"خیلی" $n=5$



سطر اول چاپ 1 ستاره
سطر دوم " 3 "
سطر سوم " 5 "
سطر چهارم " 3 "
سطر پنجم " 1 "

$\Rightarrow$

$1,3,5 \rightarrow 3$
$1,3,5,7 \rightarrow 4$
$1,3,5,7,9 \rightarrow 5$

$\rightarrow$ سطر $\frac{n+1}{2}$

تعداد ستاره در هر سطر:
- شماره سطر (k) $\leq \frac{n+1}{2}$: تعداد قبلی $+2$
- شماره سطر (k) $> \frac{n+1}{2}$: تعداد قبلی -2

تعداد space های لازم در هر سطر:

سطر اول: $n=5 \rightarrow 2$ space , $n=3 \rightarrow 1$ space $\rightarrow \frac{n-1}{2}$ تا space
$n=7 \rightarrow 3$ space , $n=9 \rightarrow 4$ space

شماره سطر $\leq \frac{n+1}{2} \rightarrow$ تعداد قبلی -1 تا

" " $> \frac{n+1}{2} \rightarrow$ تعداد قبلی 4 تا

ذخیره سازی اعداد اعشاری در کامپیوتر :

عدد اعشاری باینری : 101.1101

ارزش 2^{-1}

ارزش 2^{-2}

ارزش 2^{-3}

ارزش 2^{-4}

$2^{3}\ 2^{2}\ 2^{1}\ 2^{0}\ .\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}$

Decimal عدد اعشاری : 47.203

ارزش 10^{-1}

ارزش 10^{-2}

ارزش 10^{-3}

$\Rightarrow$ 101.1101 = 5.8125

باینری در مبنای 10

فرض کنید 4 بایت داده داریم. 2 بایت برای ذخیره قسمت اصلی و 2 بایت برای ذخیره قسمت اعشاری.
کوچکترین عدد اعشاری قابل بیان چقدر است؟ (بدون علامت)

اعداد اعشاری به دقت کوچکتری نیاز داریم. $\Rightarrow$ کوچکترین عدد $= 2^{-16} = 1.525 \times 10^{-5}$

بازهم خیلی بزرگ است. $\Rightarrow$ کوچکترین عدد $= 2^{-32} = 2.32 \times 10^{-10}$

اگر اعشار را در 4 بایت ذخیره کنیم:

IEEE 754 استانداری برای ذخیره عدد اعشاری

$17.675 = 1.10\,46875 \times 2^{4}$ ⇒ توان ۲

پایه — اینجا حتماً باید ۱ باشد.

ذخیره در ۴ بایت:

۱ (Neg), ۰ (Pos)	expanent	significant
← ستعلامت	ذخیره توان 2 در 8 بیت	ذخیره پایه در 23 بیت

ذخیره توان 2:

۰ ۱۱۱۱۱۱۱ → ۰ (127)

2^{0}

۰۰۰۰۰۰۰۱ → -126 (1)

2^{-126}

۱۱۱۱۱۱۱۰ → 127 (254)

2^{127}

مثال: $10.0 = 1.25 \times 10^{3}$ ⇒ بیت علامت: 0

0 → 127

exponent : 3 3 → 130 ⇒ 10000010

significand : 0.25 01000000000000000000000

⇒ 10.0 : به شکل اعشاری و

01000001001000000000000000000000

نوع داده عدد اعشاری در C :

float a; → 4 Byte : 1.2×10^{-38} _ 3.4×10^{38}

double y; → 8 Byte : 2.3×10^{-308} _ 1.7×10^{308}

long double z; 10 Byte : 3.4×10^{-4932} _ 1.1×10^{4932}

float a = 10.23;

float q = 1.6E-19

format specifier : %f , %lf مناسب %e برای نوع علمی

%W.pf

حداقل تعداد کل کاراکترهای برای نشان داد عدد اعشاری (شامل نقطه)

تعداد ارقام بعد نقطه

float x = 15.43

%7.3f : ∅15.430

space

%7.5f : 15.43000

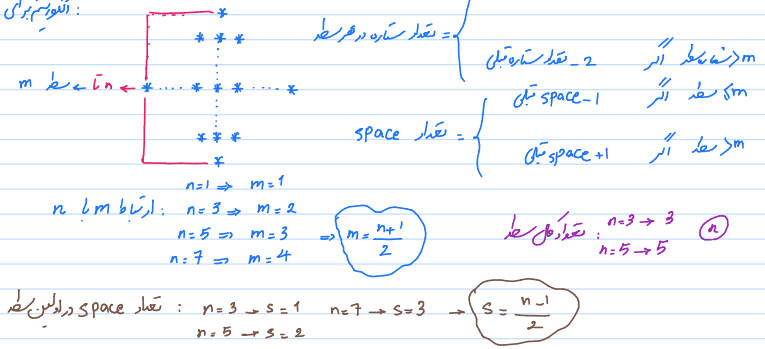
خودش کاراکتر لازم را اضافه می‌کند.

%7.1f : ∅∅∅15.4 در صورت لزوم عدد رندی می‌شود.

%.5f : 15.43000

تعداد W ضروری نیست. هم چنین تعداد P

الگوریتم برای:



الگوریتم:

1. شروع
2. n را دریافت کن
3. $middle = \frac{n+1}{2}$, $space = \frac{n-1}{2}$, $stars = 1$
4. $currentline = 1$
5. تا وقتی ($currentline \leq n$) است، خطوط 6 تا 8 را اجرا کن
6. به اندازه space، کاراکتر space و به اندازه stars، کاراکتر ستاره چاپ کن
7. $currentline + 1 \rightarrow currentline$
8. $currentline < middle \Rightarrow stars + 2 \rightarrow stars$, $spaces - 1 \rightarrow spaces$
   $currentline \geq middle \Rightarrow stars - 2 \rightarrow stars$, $spaces + 1 \rightarrow spaces$
9. پایان

Up to now:

تعریف متغیرهای عددی : int, float, double, uint8_t , int16_t , ...

آشنایی با دستور if : if (عبارت منطقی) { .... }

آشنایی با دستور while : while (عبارت منطقی) { ..... }

آشنایی با دستور printf :

escape characters: \n, \t, \b, ...

farmat specifier: %d, %u, %f , %5d , %-4d , %7.3f, ...

ذخیره سازی کاراکترها :

چند نمونه از کاراکترها : 'a', 'A', '{', '7', ' ', ...

ذخیره سازی کاراکترها در کامپیوتر توسط کد ASCII است. یعنی برای هر کاراکتر، یک کد معادل متشکل از 0 و 1 در نظر گرفته شده است که آن را ذخیره می‌کنیم.

در 1 بایت → 01100001 : 'a' مثلا

در 1 بایت → 01000001 : 'A'

کاراکتر در 1 بایت قابل ذخیره سازی است.

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

کاراکتر یا حرف c :

```
char c = 'a';
```

single quotation (حتماً ")

نوع داده

نام متغیر

```
printf("%c", c);
```

format specifier مخصوص چاپ کاراکتر

char قبلاً دیده بودیم که ابایت عدد را می‌توانند ذخیره کنند.

اینجا هم همین است. صرفاً چاپ آن توسط %c انجام می‌شود.

```
char c = 65;
printf("%d", c); => 65
printf("%c", c); => A
```

گرفتن ورودی از کاربر :

```
char c;

c = getchar();
```

برای گرفتن یک کاراکتر

getchar() : یک کاراکتر را از ورودی می‌گیرد. اگر شما چند کاراکتر وارد کنید بقیه در بافر باقی می‌ماند و اگر جای دیگری در برنامه getchar() داشتید، تا وقتی کاراکتری در این بافر است، دیگر منتظر ورودی شما نمی‌ماند.

```
c1 = getchar();
```

ورودی شما : wea\n (enter)

c1 = 'w'

باقی‌مانده در بافر ← ea\n

```
getchar();   getchar();   getchar();
```

← خالی شدن بافر

خالی کردن بافر در c :

```
char c = getchar( );

while ( getchar( ) != '\n' ) ;
```

تا وقتی به کاراکتر enter فرستیده است داخل این حلقه بگیر می‌افتد.

ذخیره آرایه ای از انواع داده در C :

انواع داده که تا الان داشتیم : int, float, double, char, ...

فرض کنید بخواهیم آرایه ای از این نوع داده ها را بخواهیم ذخیره کنیم :

Array of ints : {1, 2, 3, 4, 5}

Array of chars : {'a', 'c', 'e', 'g'}

تعریف آرایه در C :

```
int x[10];
```

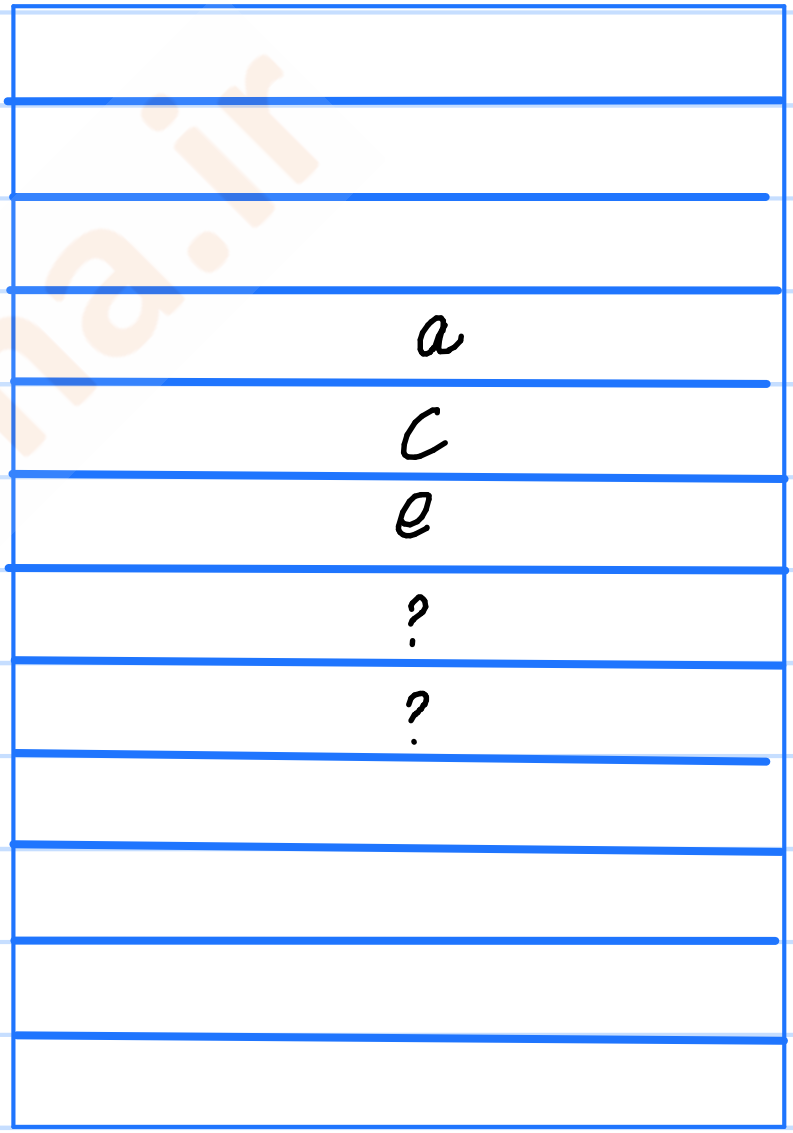
- int : نوع عضو داده آرایه
- x : نام آرایه یا همان نام متغیر
- 10 : تعداد المان های داخل آرایه

```
char  c_array[5];
```

در RAM، 5 بایت آزاد می‌شود برای متغیر c_array

حالا می‌نویسیم:

```
c_array[0] = 'a';
c_array[1] = 'c';
c_array[2] = 'e';
   .
   .
   .
```



```
int numbers[10];
```

اندیس اعضای آرایه : 0, 1, ... 9

چند بایت فضا در RAM اشغال خواهد شد ؟ $10 \times 4\ Byte = 40\ Byte$

```
char name[5] = {'h', 'a', 'd', 'i'};
```

می توان آرایه ای از کاراکترها را به صورت رشته کاراکتری هم نشان داد. برای اینکار از %s استفاده می کنیم.

```
printf("Your name is %s", name); => hadi
```

C برای اینکه متوجه شود یک رشته کاراکتری کجا تمام می شود از null character استفاده می کند ('\0'). آخرین اعضای آرایه کاراکتری را روی صفر بگذارید برای این کاراکتر:

```
char name[5] = {'h', 'a', 'd', 'i', '\0'};
```

راه ساده‌تر برای وارد کردن یک رشته کاراکتری، استفاده از double quotation (") است. دو دستور زیر معادلند:

```
char name[5] = {'H', 'a', 'd', 'i', '\0'};
char name[5] = "Hadi";
```

به هر رشته کاراکتری که داخل " و " نوشته شده باشد، توسط C یک null character به انتهای آن اضافه می‌شود. پس حداقل طول آرایه باید 1 واحد از تعداد کاراکترهای رشته بیشتر باشد.

```
char name[3] = "Ali"  ⇒  name[0]='A', name[1]='l', name[2]='i';
```

⇐ جایی برای '\0' نیست. C نمی‌داند این رشته کجا تمام شده. بنابراین درست اجرا نخواهد شد.

راه درست:

```
char name[4] = "Ali";  ⇒  name[3] = '\0';
```

به طور اتوماتیک

گرفتن یک رشته کاراکتری از کاربر:

```
char name[10];

gets(name);
```

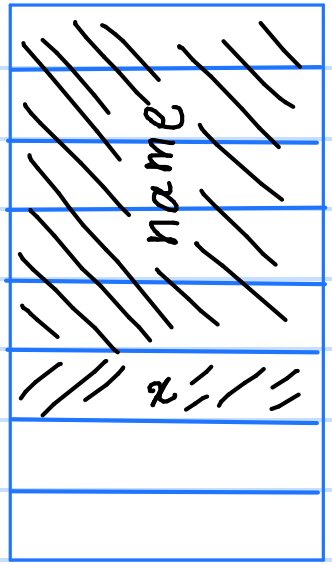
↳ برای دریافت رشته کاراکتری از کاربر

مهم: باید تعداد کاراکترهایی که وارد می‌کنیم از طول آرایه 1 واحد کمتر باشد تا جا برای null character هم باشد. وگرنه اتفاقات عجیب و غریبی می‌افتد.

(1)

```
char name[5];
uint8_t x = 10;
```

⇒



(2)

```
gets(name);
```

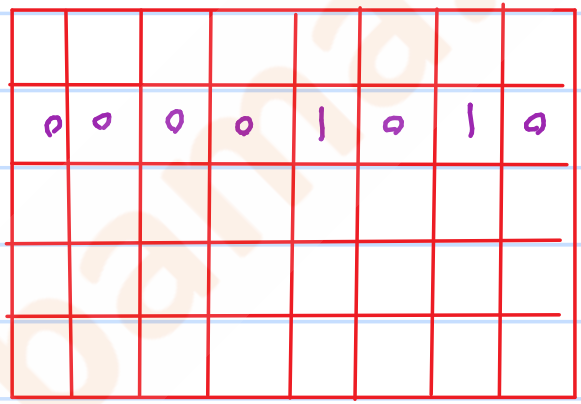
ورشته کاراکتری وارد می‌کنید که در این 5 کاراکتر تعریف شده جایی نشود. مقدار x تغییر می‌یابد. در واقع overwrite می‌شود.

اولین ویروس‌های کامپیوتری از چنین چیزی استفاده می‌کردند.

کار با آدرس متغیرها در C

```
uint8_t a = 10;
printf("%p\n", &a);
```

آدرس متغیر a در حافظه. یعنی a در RAM در کجا ذخیره شده است.



0X0.....061FE1F

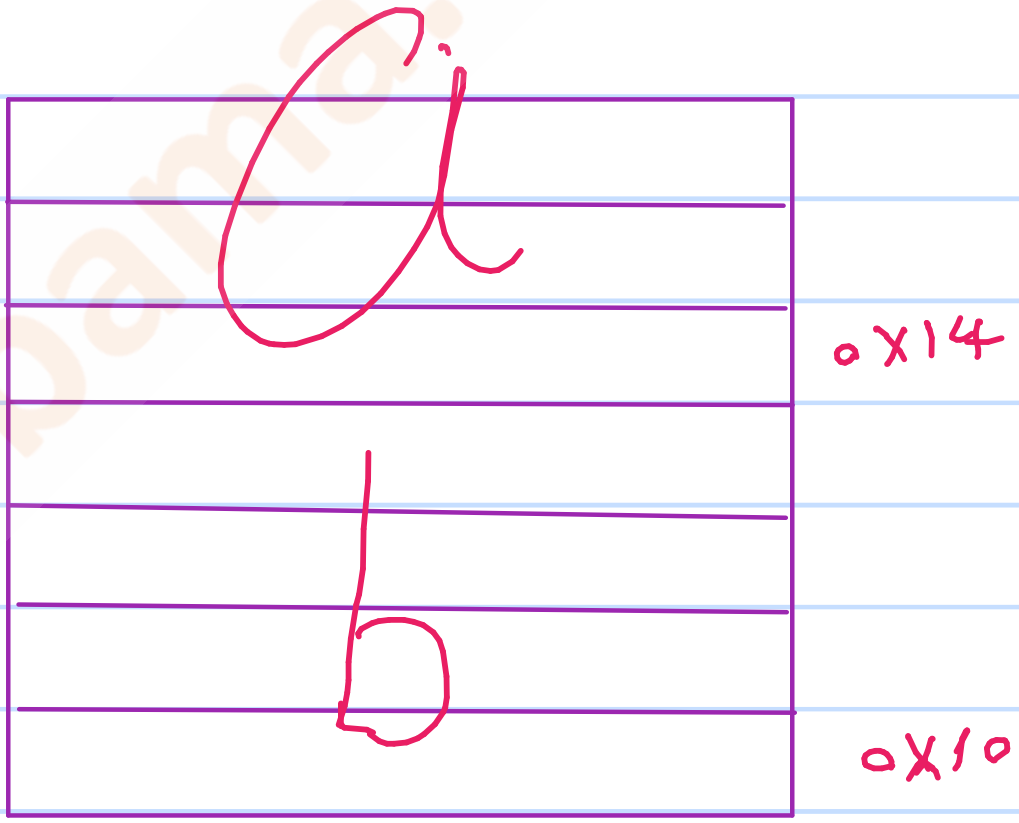
آدرس این واحد حافظه به صورت مبنای Hex

```
uint8_t a = 20;
uint8_t b = 10;
```

آدرس متغیرها را print و بررسی کن.

```
int a;
int b;
```

آدرس بالاتر



آدرس پایین‌تر

① الگوریتمی بنویسید که یک عدد صحیح مثبت را از کاربر گرفته و برعکس آن را چاپ کند.

ورودی خروجی

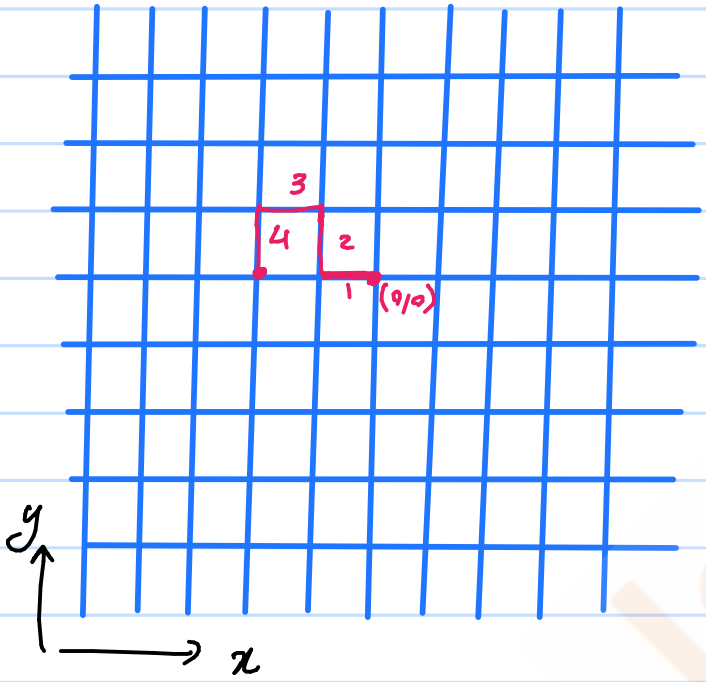
3409 => 9043

② الگوریتمی بنویسید که یک عدد صحیح مثبت را گرفته و ارقام آن را چاپ کند:

ورودی خروجی

2073 => 2 , 0 , 7 , 3

③ فرض کنید فردی در مختصات (۰،۰) در شبکه زیر ایستاده است. الگوریتمی بنویسید که یک عدد مثبت صحیح را از کاربر گرفته و این فرد را به صورت رندم در این شبکه به تعداد آن عدد حرکت دهد و مختصات نهایی او را چاپ کند. هر حرکت تنها می‌تواند بالا، پایین، چپ و راست باشد.



مثال: $n = 4$ ⇐ به تعداد ۴ بار حرکت رندم انجام بده.

(مثل شکل) ⇐ مختصات نهایی (۰،-۲)

④ الگوریتمی بنویسید که یک عدد صحیح مثبت را دریافت کرده و آن را به مولفه‌های خود تجزیه کند مثل مثال زیر:

ورودی

243 ⟹ عبارات مقابل چاپ شود: "243 = 200 + 40 + 3"

8321 ⟹ "8321 = 8000 + 300 + 20 + 1"

تمرین هایی برای زبان C

این تمرین ها از نظر الگوریتمی ساده هستند. هدف از آن‌ها، تمرین دستورات زبان C است.

① برنامه‌ای که یک عدد n را گرفته و کلیه اعداد از ۰ تا n را چاپ کند با فرمت های زیر:

n = ۱۰ ز

ب)

۱ و ۰

2 و 3

4 و 5

⋮

9 و ۱۰

الف)

0
1
2
3
4
5
⋮
۱۰

② برنامه‌ای بنویسید که دو عدد را بگیرد (شما فعلاً فرض کنید) و عبارات زیر را چاپ کند.

مثلاً $a = 4.3 \quad b = 2.1$

عبارات زیر همینا چاپ شوند:

$$a + b = 4.3 + 2.1 = 6.4$$
$$a - b = 4.3 - 2.1 = 2.2$$
$$a * b = 4.3 * 2.1 = 9.03$$
$$a / b = 4.3 / 2.1 = 2.05$$

به عنوان تمرین بیشتر یک شرط بگذارید که اگر b صفر شد، به جای خط چهارم عبارت زیر چاپ شود:

ERROR: Cannot divide by zero

③ برنامه‌ای که سال تولد یک فرد را گرفته و بسته به سن او پیام مناسب را چاپ کند.

سن ≤ 10 → یک پیام

10 < سن ≤ 20 → یک پیام

20 < سن ≤ 40 → یک پیام

40 < سن ≤ 60 → یک پیام

سن > 60 → یک پیام

۹) برنامه‌ای که یک عدد بین ۰ تا 255 را بگیرد. اگر آن عدد زوج بود آن را چاپ کند و برنامه خاتمه یابد. اما اگر فرد بود، دوباره یک عدد بگیرد و این روند آنقدر ادامه یابد تا عدد دریافتی زوج وارد شود.

مثال:

enter a number : 31 → enter a number : 135 → enter a number : 210 →

اتمام برنامه

تمرینات دیگری برای الگوریتم ها .

برای هر مسئله ، علاوه بر الگوریتم سعی کنید برنامه مناسب را هم بنویسید .

(۱) برنامه حدس زدن عدد : کامپیوتر عددی بین ۰ تا ۱۰۰۰ را به تصادف انتخاب کند . کاربر این عدد را حدس بزند اگر عدد حدس زده از عدد فرض شده بزرگتر بود ، عبارت "My number is smaller" و اگر کوچکتر بود "My number is larger" نشان داده شود و کاربر حدس دیگری بزند و این روند آنقدر ادامه یابد تا عدد پیدا شود و در این حالت عبارت "You found it" ظاهر شود .

② یک آرایه از اعداد داریم:

$\{5, 3, 0, 8, 4, 3\}$: حداقل 3 عدد بزرگتر یا مساوی 3 داریم. اما 4 عدد بزرگتر یا مساوی 4 نداریم

$\Longrightarrow$ 3 را چاپ کند

$\{6, 2, 7, 5, 3, 1, 8, 9, 4, 10\}$ $\Longrightarrow$ 5 را چاپ کند

6 عدد بزرگتر یا مساوی 6 نداریم. اما حداقل 5 عدد بزرگتر یا مساوی 5 داریم

$6 \rightleftarrows 9$ دوران 180°

$8 \rightleftarrows 8$ دوران 180°

$1 \rightleftarrows 1$ دوران 180°

$888 \rightleftarrows 888$ دوران 180°

$61819 \rightleftarrows 61819$ دوران 180°

$619 \rightleftarrows 619$ دوران 180°

برنامه یک عدد n را گرفته و تمام اعدادی که n رقم دارند و با دوران 180° خودشان می‌شوند را چاپ کند.

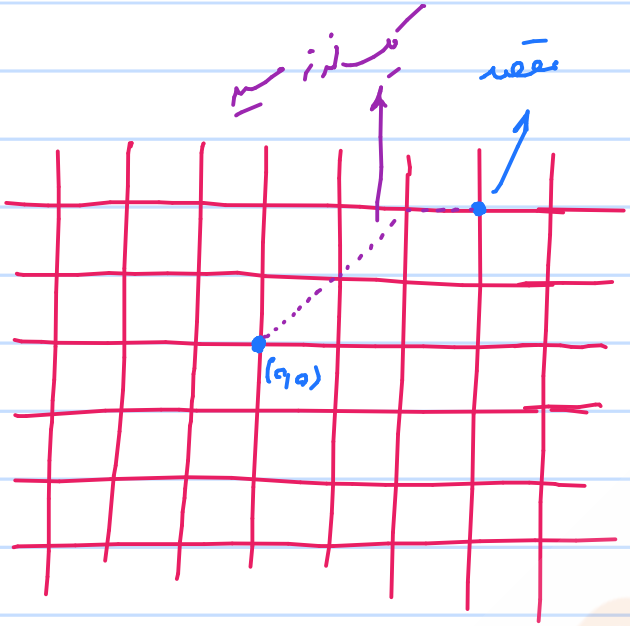
۴) برنامه‌ای بنویسید که اعداد آرایه را به اندازه k واحد شیفت دهد و آن را چاپ کند.

$\{5,4,3,2,1\} \xrightarrow{k=1} \{1,5,4,3,2\}$

$k=2$ ↳ $\{2,1,5,4,3\}$

⋮

(5) یک شبکه دوبعدی داده شده است و نقطه اولیه مبدأ مختصات است. یک نقطه مقصد داده می‌شود. کوتاه‌ترین مسیر شرقی به این نقطه را بنویسید.



مقصد : (3,2)

حرکت مورب بین خانه‌ها مجاز است. یعنی در هر نقطه ۳ حرکت می‌توان انجام داد.

⑥ برنامه‌ای بنویسید که n عدد "دوست 7" اول را پیدا کنید.

عدد دوست 7 : یا توان‌های 7 یا مجموع

توان‌های 7

$n = 5$ مثلاً ، 1, 7, 8, 49, 50, ...

نوع داده bool در C :

برای ذخیره مقادیر منطقی (درست یا غلط) :

```
bool b = true;
bool b = false;
```

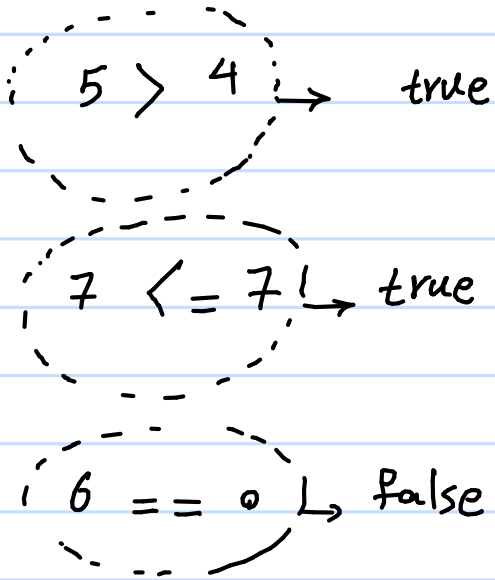
برای استفاده از نوع داده bool ، باید عبارت #include <stdbool.h> را ابتدای کد اضافه کنید .

```
bool b = true;
```

- bool ← نوع متغیر
- b ← نام متغیر
- true ← یک عبارت منطقی که حاصل آن true یا false است .

نمونه‌هایی از عبارات منطقی :

بزرگتر یا مساوی	>=
کوچکتر یا مساوی	<=
بزرگتر	>
کوچکتر	<
مساوی	==
نامساوی	!=



برنامه‌ای که بررسی می‌کند آیا کاراکتر وارد شده توسط کاربر 'a' است یا خیر:

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>

int main()
{
    char ch;
    printf("Enter a character: ");
    ch = getchar();
    while (getchar() != '\n');

    bool b = ch == 'a';

    if (b) {
        printf("You have entered \"a\"\n");
    } else {
        printf("You have entered a character other than \"a \"\n");
    }
}
```

یک کاراکتر می‌گیرد.

بافر را خالی می‌کند.

یک عبارت منطقی ch == 'a' است. یعنی یا true است یا false

b = ch == 'a'

مثل b = 5 > 4 است. یعنی b را برابر یک عبارت منطقی قرار می‌دهیم

می‌بینید که داخل پرانتز همان عبارت منطقی قرار گرفته.

مبلغ دستور if و while را معرفی کردیم. این دو دستور برای اجرا درستی یک شرط را بررسی می‌کنند.
این شرط در واقع یک عبارت منطقی است. یعنی داده‌ای از نوع bool است.

ترکیب عبارات منطقی: عبارات منطقی را می‌توان با هم ترکیب کرد. به شکل زیر:

عبارت منطقی ۱ — عملگر ترکیب — عبارت منطقی ۲

عملگر ترکیب:
&& : به معنی "و"
|| : " " "یا"

در هر حالت بنویسید که b ، true است یا false :

```
b = 5 > 4 && 8 <= 6

b = 3 < 0 || 7 > 4

b = 9 <= 20 && 10 <= 11 && 100 < 50

b = (10 < 5 || 20 <= 20) && 0 != 2

b = (10 != 0 && 5 > 10) || 3 >= 0
```

داخل شرط if و while می توانید از عبارات منطقی یا ترکیب عبارات منطقی استفاده کنید.

برنامه‌ای که بررسی می‌کند کاراکتر وارد شده یکی از 'a' ، 'b' ، 'c' و 'd' است یا خیر .

عین برنامه قبل است . فقط از ترکیب عبارات منطقی استفاده شده است .

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>

int main()
{
    char ch;
    printf("Enter a character: ");
    ch = getchar();
    while (getchar() != '\n');

    bool b = ch == 'a' || ch == 'b' || ch == 'c' || ch == 'd';

    if (b) {
        printf("You have entered one character among (a,b,c,d)\n");
    } else {
        printf("You have entered a character other than (a,b,c,d)\n");
    }
}
```

چند برنامه تمرینی همراه با جواب . توصیه می‌کنم قبل از مطالعه جواب خودتان تلاش کنید کد را بنویسید .

برنامه تشخیص سن :

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>

int main()
{
    uint8_t age = 87;
    if (age <= 10){
        printf("Your age is less than or equal to 10\n");
    } else if (age <= 20) {
        printf("Your age is more than 10 and less than or equal to 20\n");
    } else if (age <= 40) {
        printf("Your age is more than 20 and less than or equal to 40\n");
    } else {
        printf("Your age is more than 40\n");
    }
}
```

برنامه چاپ جمع، تفریق، ضرب و تقسیم دو عدد :

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>

int main()
{
    float a = 10, b = 12;
    printf("a = %.2f, b = %.2f\n", a, b);
    printf("a + b = %.2f + %.2f = %.2f\n", a, b, a + b);
    printf("a - b = %.2f - %.2f = %.2f\n", a, b, a - b);
    printf("a * b = %.2f * %.2f = %.2f\n", a, b, a * b);
    printf("a / b = %.2f / %.2f = %.2f\n", a, b, a / b);
}
```

main.c

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

int main()
{
    int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int arrSize = sizeof(arr)/sizeof(arr[0]);

    int newArr[arrSize];
    int index = 0;
    while (index < arrSize) {
        newArr[index] = arr[arrSize - 1 - index];
        index++;
    }

    index = 0;
    printf("New Array is: [");
    while (index < arrSize) {
        printf("%d,", newArr[index]);
        index++;
    }
    printf("\b]\n");
}
```

برنامه‌ای که یک آرایه از اعداد را معکوس کرده و آن را چاپ کند.

مثال:

{0,1,2,3,4,5} ⇒ {5,4,3,2,1,0}

برنامه‌ای که یک رشته کاراکتری را گرفته و اعداد داخل آن را حذف کند و به جای آن‌ها ، ستاره بگذارد : *

main.c

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

int main()
{
    char charArray[10];
    printf("I will remove any digit from the string you enter! :)\n");
    printf("Enter any String: ");

    int index = 0;
    charArray[0] = getchar();
    while (index < sizeof(charArray)/ sizeof(charArray[0]) && charArray[index] != '\n') {
        index++;
        charArray[index] = getchar();
    }
    charArray[sizeof(charArray)/ sizeof(charArray[0]) - 1] = '\0';
    printf("You Entered: %s\n", charArray);

    index = 0;
    while (index < sizeof(charArray)/ sizeof(charArray[0])) {
        if (charArray[index] <= 57 && charArray[index] >= 48) {
            charArray[index] = '*';
        }
        index++;
    }
    printf("Your string without digits: %s\n", charArray);
}
```

مثال :

ورودی :

x24f09c

خروجی :

x**f**c

دستور scanf : روش دیگر گرفتن ورودی از کاربر.

getchar() → فقط یک کاراکتر

gets() → گرفتن یک رشته کاراکتری از کاربر

scanf() → گرفتن عدد صحیح، اعشاری یا رشته کاراکتری (طبق فرمت)

```
int a;

scanf("%d", &a);
```

فرمت تعریف شده

آدرس متغیر a

در دستور scanf، فرمت تعیین می‌کند دقیقاً شما انتظار دارید ورودی به چه نحوی وارد شود. به مثال‌های صفحه بعد دقت کنید.

```
int a,b;
scanf("%d%d",&a,&b);
```

فاصله در فرمت تعریف شده

ورودی مورد انتظار : 10 321 → نتیجه: a = 10, b = 321

```
int a;
float b;
scanf("%d %f",&a,&b);
```

ورودی مورد انتظار : 20 12.02 → نتیجه a = 20, b = 12.02

```
char name[30];
scanf("%s",name);
```

هنگام اسکن کردن رشته کاراکتری دیگر لازم نمی‌کنیم "خیلی مهم"

ورودی مورد انتظار : "Ali" → نتیجه: name = {'A','l','i','\0',....}

ملاحظات مهم :

① scanf : معنی scan formatted است. کنترل کردن تایپ space ادامه می‌یابد. یعنی scanf، به طور کلی کاراکتری که داخلش space دارد را نمی‌توانید دریافت کنید و gets() برای اینکار مناسب است.

مثل getchar() داده بعد از space داخل بافر باقی می‌ماند.

مثال:

```
char a[30], b[30];
scanf("%s", a);   → ورودی : Amir Ali  =>  a = Amir ,  "Ali" داخل بافر
scanf("%s", b);   → بافر خالی نیست. دیگر کنترل نمی‌کند ,  b = Ali
printf("a is %s, b is %s\n", a, b);  => a is Amir, b is Ali
```

(2)

scanf درست مثل gets، محافظت از سرریز (overflow) را ندارد. مثلاً در کد زیر در اثر سرریز مقدار x تغییر می‌کند.

```
char name[5];
int  x=10;
scanf("%s", name);
printf("%d", x);
```

→ مثلاً اینجا ورودی می‌دهیم که در name جا نمی‌شود. Amirhosein

↳ به دلیل سرریز مقدار x تغییر خواهد کرد و ۱۰ از بین خواهد رفت.

③ برای جلوگیری از سرریز برای تعداد کاراکترهای اسکن شده حد قرار می‌دهیم. این کار برای تمام انواع ورودی‌ها مثل عدد و رشته کاراکتری قابل انجام است.

مثال: `int x;`
`scanf("%2d", &x);`
یعنی تعداد کاراکتر مجاز

ورودی	x
12	12
981	98
-123	-1

مثال: `char name[10];`

`scanf("%9s", name);`
یعنی تعداد کاراکتر مجاز، از ۱۰ یکی کمتر است تا جا برای '\0' هم باشد

ورودی	name
Ali	Ali
AmirAliArdalan	AmirAliAr

(۱۱) راه دیگر برای جلوگیری از سرریز این است که به جای gets() از fgets() و به جای scanf() از sscanf() استفاده کنیم.

```
char name [10];

fgets (name, 10, stdin )
```

نام متغیر خوانده شده

تعداد کل کاراکترهایی خواندن یا دریافت ورودی از ورودی استاندارد است (stdin).

* در واقع ۹ کاراکتر خواهد خواند و کاراکتر آخر '\0' خواهد بود.
* fgets، کاراکتر newline را هم داخل رشته دریافتی نگه می‌دارد و آن را حذف نمی‌کند. '\n'

چاپ اعداد زوج

```
int x = 0;
while (x <= 50) {
    printf("%d ", x);
    x = x + 2;
}
```

✓ مناسب و درست

```
int x = 0;
while (x = x+2, x <= 50) {
    printf("%d ", x);
}
```

✗ نامناسب و مردود

چرا باید داخل پرانتزی که محل شرط حلقه while است، مقدار متغیر x عوض شود؟ این روش کدنویسی به هیچ وجه توصیه نمی‌شود. به شدت دارای ابهام است و هیچ مزیتی هم ندارد و مردود است.

حلقه for : مانند while ، دستور for هم یک حلقه تعریف می‌کند. اما حلقه for برای تکرار به دفعات معین و مقدحقی مناسب‌تر است.

syntax : for (عبارت اول ; عبارت دوم ; عبارت سوم) {

// داخل حلقه

}

مثال ها صفحه بعد

عبارت اول : فقط یک بار قبل از ورود به داخل حلقه اجرا می‌شود.

عبارت دوم : شرط تکرار حلقه است. در واقع یک عبارت منطقی است.

عبارت سوم : پس از اجرای کد داخل حلقه، "هر دفعه" اجرا می‌شود.

```c
int i;
for (i = 0; i < 10; i = i + 1) {
    printf("%d,", i);
}
printf("\b \n");
printf("Value of is is now: %d\n", i);
return 0;
```

نتیجه :

```
0,1,2,3,4,5,6,7,8,9
Value of is is now: 10
```

عبارت اول : $i=0$ ← یک بار اجرا شده و ابزار صفردهی شود.

عبارت دوم : $i<10$ ← شرط اجرای کد داخل حلقه

عبارت سوم : $i=i+1$ ← پس از هر بار اجرای کد داخل حلقه اجرا می شود.

روند اجرا : $i=0$ → chek if $i<10$ ✓ → Printf → $i=i+1$ → chek ✓ → Printf → $i=i+1$ → ..... → $i=i+1$ → checkX → خروج از حلقه → چاپ نهایی i

$i=i+1$: 10

چند مثال دیگر:

```
for (int i = 20; i > 0; i -= 3) {
    printf("%d,", i);
}
printf("\b \n");
return 0;
```

اگر متغیر داخل حلقه "i" فقط داخل حلقه لازم است، می‌توان تعریف متغیر را هم داخل عبارت اول انجام داد.

i = i - 3 معادل با

نتیجه: 20,17,14,11,8,5,2

```
for (int i = 65; i <= 90; i++) {
    printf("%c,", i);
}
printf("\b \n");
return 0;
```

توضیح: 65 به صورت decimal معادل با کاراکتر 'A' است. مطابق جدول ASCII.

نتیجه: A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z

```
for (int i = 10, j = 0; i >= 0, j <= 10; i--, j++) {
    printf("[%d,%d],", i,j);
}
printf("\b \n");
return 0;
```

توسعه گاهی تواند چند دستور را از هم جدا کرد. اینجا هم از هم ز : int i = ۱۰ و j = ۰ : عبارت اول
تعریف و مقدار دهی شده است.

هر دو بین از هر بار اجرای حلقه تست : i >= ۰ و j <= ۱۰ : عبارت دوم
می شوند.

عبارت سوم : i-- ⟵ معادل با i = i - ۱ و j++ ⟵ معادل با j = j + ۱

نتیجه :
```
[10,0],[9,1],[8,2],[7,3],[6,4],[5,5],[4,6],[3,7],[2,8],[1,9],[0,10]
```

```c
int i = 0;
for (; i < 10;) {
    printf("%d,", i);
    i += 1;
}
printf("\b \n");
return 0;
```

* می‌توانیم هرکدام از 3 عبارت را که لازم ندارم ننویسیم. اما ; اجباری است.

* اینجا عبارت اول (int i = 0) خارج حلقه نوشته شده است و در حلقه جای آن خالی است.

* i += 1 هم داخل حلقه نوشته شده است و لذا جای آن در دستور for باز هم خالی است.

نتیجه :

```
0,1,2,3,4,5,6,7,8,9
```

```
for (int i = 1; i < 10; i++) {
    for (int j = 1; j < 10; j++) {
        printf("%5d", i * j);
    }
    printf("\n\n");
}
return 0;
```

اینجا از دو حلقه for تودرتو برای چاپ جدول ضرب استفاده شده است.

نتیجه:

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

```
for (int i = 1; i < 20; i++) {
    for (int j = 1; j < 20; j++) {
        if (pow((i - 10),2) + pow((j-10),2) < 30) {
            printf("*");
        } else {
            printf("-");
        }
    }
    printf("\n");
}
return 0;
```

چاپ یک الگوی دایره‌ی:

اگر $(x-10)^2 + (y-10)^2 < 30$ ← "*"

در غیر اینصورت ← "-"

نتیجه:

```
--------------------
--------------------
--------------------
--------------------
-------*****--------
------*******-------
-----*********------
----***********-----
----***********-----
----***********-----
----***********-----
----***********-----
-----*********------
------*******-------
-------*****--------
--------------------
--------------------
--------------------
--------------------
```

دستور break : دستور break داخل حلقه نوشته شده و باعث خروج از حلقه می شود.

اگر چند حلقه تو در تو داشته باشیم ، break باعث خروج از نزدیکترین حلقه می شود.

دستور continue : دستور continue موجب می شود که اجرای فرمان از داخل حلقه به اول حلقه منتقل شود.

منظور از حلقه ، هم حلقه while است و هم حلقه for.

به مثال اسلایدبعد دقت کنید.

```
for (int i = 0; i < 10; i++) {
    if (i > 5) {
        break;
    }
    printf("%d,",i);
}
printf("\b ");
return 0;
```

وقتی i=6 شده، شرط i>5 برقرار می‌شود و true و لذا دستور break اجرا شده و لذا اجرای کد از حلقه خارج شده و اعداد 6 و بالاتر چاپ نخواهند شد.

نتیجه.

```
0,1,2,3,4,5
```

```
for (int i = 0; i < 10; i++) {
    if (i == 6) {
        continue;
    }
    printf("%d,",i);
}
printf("\b ");
return 0;
```

وقتی i=6 شد، شرط i==6 می شود true و

لذا دستور continue اجرا می شود.

این دستور موجب می شود اجرای برنامه برگردد به ابتدای

حلقه ولذا عدد 6 پرینت نخواهد شد.

نتیجه

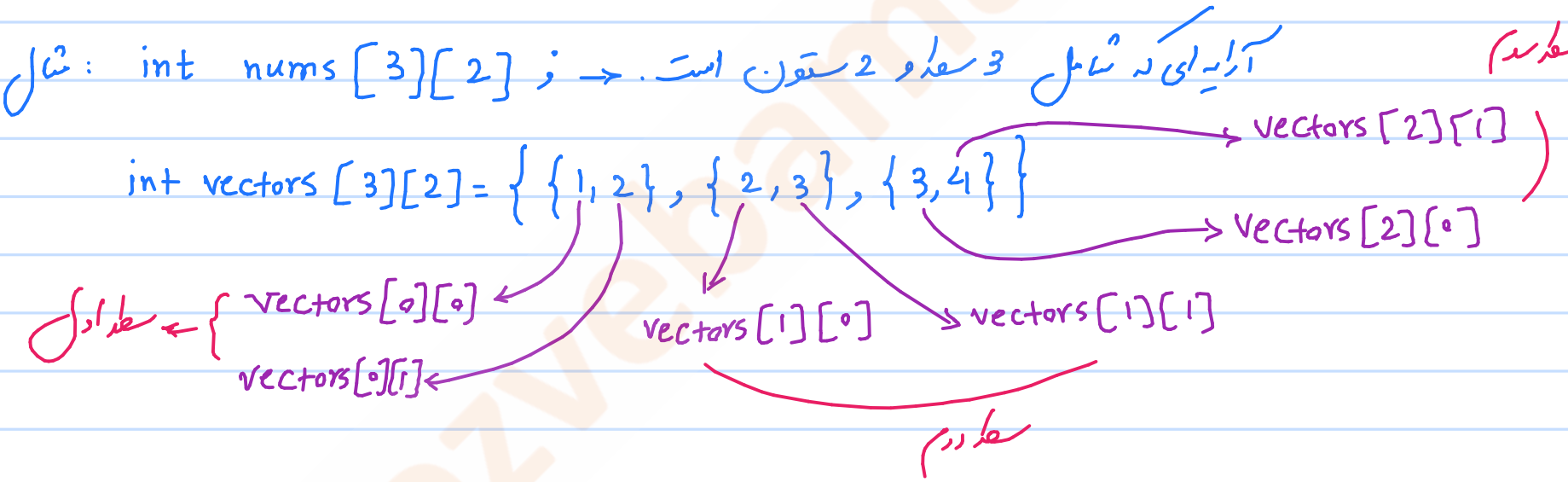
```
0,1,2,3,4,5,7,8,9
```

مشاهده عدد 6 نیست.

آرایه‌های چند بعدی در C :

تعریف type name [x][y][z] ...

مثال : int nums [3][2] ; → آرایه‌ای که شامل 3 سطر و 2 ستون است.



```
// define 2D arrays
// type name[row size][column size];
int vectors[4][2] = {{1,0},{0,1}, {1,2}, {4,2}};
int sets[][3] = {{1,2,3}, {4,5,6}};

// access the elements => x[i][j]
vectors[1][1] = 3;

printf("{");
for (int i = 0; i< sizeof(vectors)/ sizeof(vectors[1]); i++) {
    printf("{");
    for (int j = 0; j < sizeof(vectors[1])/sizeof(vectors[1][1]); j++) {
        printf("%d,", vectors[i][j]);
    }
    printf("\b},");
}
printf("\b}");
```

4 = sizeof(vectors)/ sizeof(vectors[1])

2 = sizeof(vectors[1])/sizeof(vectors[1][1])

sizeof(vectors[1][1]) = 4

sizof(vectors[1]) = 8

sizeaf(vectors) = 32

```c
char names[5][10] = {"Ali", "Farzad", "Peyman", "Hassan", "Taymaz"};
char languages[][10] = {"C", "C++", "C#", "Java", "php", "python"};
//ERROR : languages[2] = "html"; *
strcpy(languages[2], "html");
for (int i = 0; i < sizeof(languages)/sizeof(languages[0]);i++) {
    printf("%s\n", languages[i]);
}
```

* پس از ایجاد آرایه‌ای از رشته‌های کاراکتری، نمی‌توان مقادیر آن را به صورت languages[2]="html" تغییر داد.

برای تغییر باید از strcpy استفاده کرد.

استفاده از regular expression ها در scanf :

```
char str[11];
scanf("%10[.....]", str);
```

طول رشته امکان رشته

هر چیزی که بین [و] قرار گرفته باشد، یک scanset را تعریف می‌کند. یعنی کاراکترهای وارد شده می‌توانند شامل آن‌ها باشند

به محض رسیدن به کاراکتری غیر از آن‌ها، اسکن متوقف می‌شود.

مثال‌ها صفحه بعد.

```
scanf("%9[0-9]", str);
```
→ رشته کاراکتری ورودی فقط شامل عدد باشد

```
scanf("%9[a-z]", str);
```
→ حروف کوچک انگلیسی " " " " "

```
scanf("%9[A-Z]", str);
```
→ بزرگ " " " " "

```
scanf("%9[0-9a-zA-Z]", str);
```
→ رشته کاراکتری ورودی شامل اعداد و حروف کوچک و بزرگ انگلیسی

```
scanf("%9[a-zA-Z ]", str);
```
→ " " " " " space " " "

space

```
scanf("%9[^\n]", str);
```
newline کاراکتر \n به جز هر چیزی یعنی scanset را ایجاد می‌کند. این یعنی منفی است.

مقایسه رشته های کاراکتری در C :

چون نام هر رشته کاراکتری به آدرس آن اشاره می کند مقایسه آنها با == غلط است.

```
char name1[10] = "Ali";
char name2[10] = "Ali";
```

name1 == name2

آدرس این دو متغیر را مقایسه می کند

و می بیند متفاوت هستند. لذا حاصل آن False است.

برای مقایسه رشته های کاراکتری در C ، از دستور strcmp استفاده می شود.

strcmp (name1, name2) → اگر برابر باشند مقدار ۰ و اگر برابر نباشند مقدار

دیگری غیر از صفر برگردانده می شود.

```
char name1[10] = "Ali";
char name2[10] = "Ali";
if (strcmp(name1, name2) == 0) {
    printf("names are equal");
} else {
    printf("names are not equal");
}
```

در اسلایید قبل به جای بلوک if_else می توانیم از ternary expression استفاده کنیم.

ternary expression:

شرط (عبارت منطقی) ? عبارت دوم : عبارت سوم

اگر شرط درست باشد عبارت دوم اجرا می شود، اگر غلط باشد عبارت سوم اجرا می شود.

```
strcmp(name1, name2) == 0 ? printf("names are equal") : printf("names are not equal");
```

```
char x = getchar();
while(getchar()!= '\n');

switch (x) {
case 'a':
    printf("your choice is: a");
    break;
case 'b':
    printf("your choice is: b");
    break;
case 'c':
    printf("your choice is: c");
    break;
case 'd':
    printf("your choice is: d");
    break;
default:
    printf("Your choice is not acceptable");
}

return 0;
```

دستور switch_case :

* switch می تواند روی انواع داده های صحیح و کاراکتری انجام شود.

* دستور break برای هر case الزامی است. اگر قرار داده نشود، اجرای دستور به case بعدی ادامه می یابد.

* اگر هیچ کدام از case ها اجرا نشود، default اجرا می شود.

* بهتر است default آخرین گزینه باشد. در غیر اینصورت باید برای آن هم break قرار داد.

```c
char x = getchar();
while(getchar()!= '\n');

switch (x) {
case 'a' ... 'd':
    printf("Your choice is acceptable");
    break;
default:
    printf("Your choice is not acceptable");
}

return 0;
```

یک ... range را تعریف می‌کند. یعنی همه آنچه داخل بازه a تا d است.

دوروش برای داشتن چند case با یک خروجی:

```c
char x = getchar();
while(getchar()!= '\n');

switch (x) {
case 'a':
case 'b':
case 'c':
case 'd':
    printf("Your choice is acceptable");
    break;
default:
    printf("Your choice is not acceptable");
}

return 0;
```

توابع

- بلوک هایی از کد که کار مشخصی را انجام می دهند
- قابل استفاده مجدد هستند.
- خوانایی کد و توانایی نگهداری آن را افزایش می دهند.

قسمت های الزامی هر تابع:

```
return_type function_name (inputs) {
    body of function
}
```

- نام تابع
- ورودی های تابع
- خروجی تابع
- بدنه تابع

اگر بدنه تابع را تعریف نکنیم، به آن function prototype می گوییم.

مثال:

```
int   functionOne (int , int );
int   functionTwo ( );
char  functionThree (char );
void  functionFour ( float );
void  functionFive ( );
```

ورودی دو عدد صحیح و خروجی یک عدد صحیح است.

هیچ ورودی ندارد و یک عدد صحیح برمی گرداند.

ورودی و خروجی هر دو کاراکتر هستند.

ورودی عدد اعشاری است، اما هیچ خروجی ندارد.

نه ورودی دارد و نه خروجی

حال بدنه توابع را تعریف کنیم.

در این اسلاید توابعی را مشاهده می‌کنید که هرکدام یک کار را انجام می‌دهند: ماکزیمم دو عدد را پیدا کرده و برمی‌گردانند.

(1)

```
int max(int x, int y) {
    int z;
    if (x > y) {
        z = x;
    } else {
        z = y;
    }
    return z;
}
```

هر 3 تابع، دو ورودی int دارند که x و y نام‌گذاری شده‌اند و نوع خروجی هم int است. در کد (1)، یک int z داخل بدنه تابع تعریف شده که برابر عدد بزرگ‌تر خواهد بود و نهایتاً z برگردانده شده است.

(2)

```
int max(int x, int y) {
    if (x > y)
        return x;
    else
        return y;
}
```

در کد (2)، از متغیر اضافی استفاده نشده است. اگر $x > y$ باشد، x برگردانده شده و درغیر اینصورت، y برگردانده می‌شود.

(3)

```
int max(int x, int y) {
    if (x > y)
        return x;
    return y;
}
```

در کد (3)، دیگر "درغیراینصورت" تعریف نشده است. چون return از تابع خارج می‌شود و اگر $y \geq x$ باشد، اجرای کد اصلاً به return y نمی‌رسد.

```c
int main()
{
    int number = 201;
    printf("%d\n", number);
*   reverseNumber(number);
    printf("\n");
**  int rev = reverseNumber2(number);
    printf("%d\n", rev);

}

void reverseNumber(int n) {
    while (n > 0) {
        printf("%d", n % 10);
        n = n / 10;
    }
}

int reverseNumber2(int n) {
    int r = 0;
    while (n > 0) {
        r = r * 10 + n % 10;
        n = n / 10;
    }
    return r;
}
```

(1)

(2)

تابع 1، نوع ورودی int و نوع خروجی void دارد. یعنی عمیج چیزی برنمی‌گرداند.
می‌بینید که printf داخل بدنه این تابع است و این تابع برای پرینت وارون یک عدد است.

تابع 2، وارون عدد را برمی‌گرداند. همان طور که می‌بینید یک r از نوع int تعریف شده و نهایتاً بازگردانده می‌شود.

*: چون تابع reversenumber چیزی برنمی‌گرداند، در این خط آن را فراخوانی کرده‌ایم و نتیجه آن را برابر چیزی قرار نداده‌ایم.

**: چون reverseNumber2، int برمی‌گرداند، آن را داخل rev ذخیره و سپس در خط بعد چاپ کرده‌ایم.

مهم: دقت کنید نام متغیر ورودی در تابع n است ولی در main، متغیری با نام number به آن داده شده است. هیچ لزومی ندارد نام متغیرها یکسان باشند.

```c
int main()
{
    int number = 201;
    reverseNumber();
}

void reverseNumber() {
    while (number > 0) {
        printf("%d", number / 10);
        number = number / 10;
    }
}
```

local variable

در اینجا یک تابع reverseNumber نوشته‌ایم که هیچ ورودی و خروجی ندارد. چون number داخل main تعریف شده است، با خود گفته‌ایم که احتمالاً داخل تابع دیگر هم دردسترس باشد. اما همان طور که می بینید، ارور داده شده است، چون number یک متغیر محلی (local) است و فقط داخل تابع main دردسترس است.

```
note: previous implicit declaration of 'reverseNumber' was here
In function 'reverseNumber':
error: 'number' undeclared (first use in this function)
note: each undeclared identifier is reported only once for each function it.
=== Build failed: 1 error(s), 3 warning(s) (0 minute(s), 0 second(s)) ===
```

```c
int main()
{
    int x = 10, y = 20;
    printf("Before swap: x = %d, y = %d\n", x,y);
    swapNumbers(x,y);
    printf("After swap: x = %d, y = %d\n", x,y);
}

void swapNumbers(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
    printf("Inside swap: x = %d, y = %d\n", x, y);
}
```

برنامه روبرو برای عوض کردن مقدار دو متغیر باهم نوشته شده است.

یعنی می خواهیم برنامه ای بنویسیم که قبل از فراخوانی تابع x=10 و y=20 باشد، بعد از فراخوانی تابع، x=20 و y=10 شود.

سه دستور printf داریم. یکی قبل از فراخوانی تابع، دومی داخل تابع و سومی بعد از فراخوانی تابع که با شماره های 1 و 2 و 3 مشخص شده اند.

نتیجه اجرا

```
Before swap: x = 10, y = 20
Inside swap: x = 20, y = 10
After swap: x = 10, y = 20
```

اگر به نتیجه اجرا دقت کنید می بینید که داخل تابع، متغیرها مقدار خود را عوض کرده اند، اما پس از فراخوانی تابع، مقدار متغیرها همان قبلی است و عوض نشده. چرا؟

```c
int main()
{
    int x = 10, y = 20;
    printf("Before swap: x = %d, y = %d\n", x,y);
    swapNumbers(x,y);
    printf("After swap: x = %d, y = %d\n", x,y);
}

void swapNumbers(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
    printf("Inside swap: x = %d, y = %d\n", x, y);
}
```

خیلی مهم:

تابع swapNumbers، دو ورودی از نوع int دارد. وقتی که شما در تابع main، تابع swapNumbers را فراخوانی کرده و دو متغیر x و y را به آن می‌دهید، خود متغیرها را به تابع به عنوان ورودی نمی‌دهید. بلکه صرفاً مقادیر آن را می‌دهید. در تابع swapNumbers، دو متغیر جدید ایجاد شده و مقدار آن برابر با مقادیر دریافت شده قرار می‌گیرد. یعنی x و y در تابع main، همان x و y در تابع swapNumbers نیست. به این مفهوم Pass by value گفته می‌شود.

```c
int main()
{
    int x = 10, y = 20;
    printf("Before call: Address of x: %p, Address of y: %p\n", &x, &y);
    swapNumbers(x,y);
    printf("After call: Address of x: %p, Address of y: %p\n", &x, &y);
}

void swapNumbers(int x, int y) {
    printf("Inside call: Address of x: %p, Address of y: %p\n", &x, &y);
}
```

برای درک بهتر به برنامه روبرو دقت کنید. در این برنامه، آدرس متغیرهای x و y قبل از فراخوانی تابع، داخل فراخوانی تابع و بعد از فراخوانی تابع چاپ خواهد شد.

به خروجی برنامه دقت کنید.

خروجی:

```
Before call: Address of x: 000000000061FE1C, Address of y: 000000000061FE18
Inside call: Address of x: 000000000061FDF0, Address of y: 000000000061FDF8
After call: Address of x: 000000000061FE1C, Address of y: 000000000061FE18
```

همان طور که می‌بینید داخل تابع آدرس متغیرهای x و y متفاوت است. در واقع، آن‌ها متغیرهای جدیدی هستند که داخل آن تابع تعریف شده و دست‌کاری با آن‌ها انجام می‌دهید، داخل همان تابع باقی می‌ماند و به main منتقل نمی‌شود.

variable scope :

محدوده متغیری فقط داخل { } در دسترس است که داخل آن تعریف شده باشد.

```
int main( )  {

    int a = 0;
    for (int b = 0; b <10 ; b++) {

      //some work

    }

    return 0;

}
```

a داخل تابع main یعنی بین { } و b فقط داخل حلقه for یعنی داخل { } در دسترس است.

پس به a داخل حلقه for هم دسترسی داریم. چون { } داخل { } است. اما به b خارج از { } دسترسی نداریم.

بنابراین متغیرهایی که داخل هر تابع تعریف شده اند، فقط داخل آن تابع در دسترس هستند.

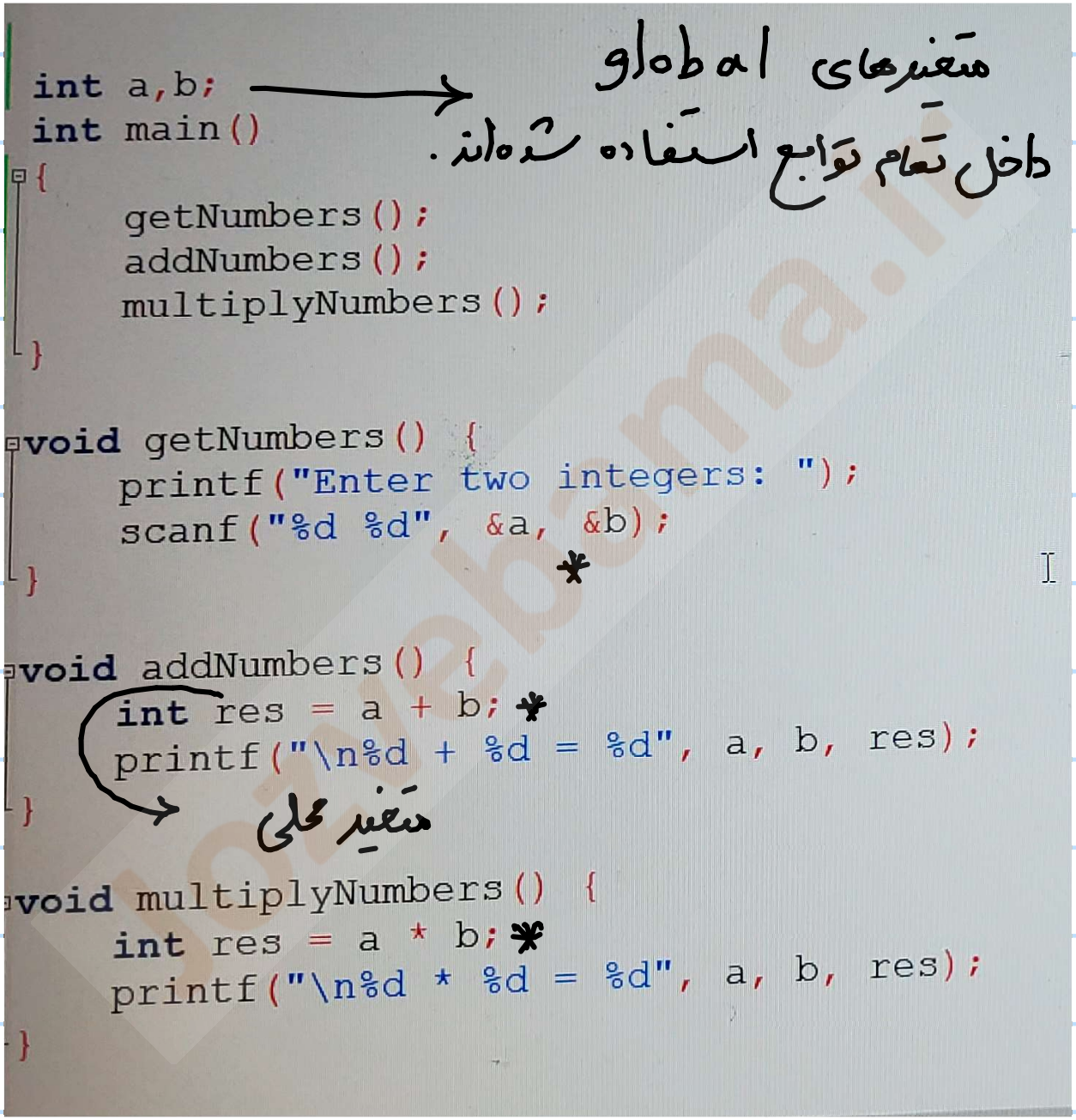
حال در C ، ما متغیرهای global را هم داریم. متغیرهای global خارج از همه توابع تعریف می شوند و در همه توابع در دسترس هستند.

اما نباید همه متغیری را global تعریف کنیم. متغیری را global تعریف کنیم که در تمام یا اکثریت توابع برنامه نیاز باشد.

مثال هایی از برنامه های حاوی متغیرهای global در اسلاید بعد آورده شده است.

<< بهتر است متغیرها کوچکترین scope ممکن را داشته باشند >>

<< global بزرگترین scope ممکن است >>




```c
int a,b;
int main()
{
    getNumbers();
    addNumbers();
    multiplyNumbers();
}

void getNumbers() {
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);
}

void addNumbers() {
    int res = a + b;
    printf("\n%d + %d = %d", a, b, res);
}

void multiplyNumbers() {
    int res = a * b;
    printf("\n%d * %d = %d", a, b, res);
}
```

تعریف ثابت ها در C :

```
const double PI = 3.141519;
```

ثابت ها، متغیرهایی هستند که قبل از اجرا، فقط یک بار مقدار آنها داده می شود و دیگر نمی توان آن را عوض کرد.

راه دیگر استفاده از #define است.

```
const float PI = 3.141519;
int main()
{
    float r, area, circumference;
    printf("enter radius: ");
    scanf("%f", &r );
    area = PI * r * r;
    circumference = 2 * PI * r;
    printf("Area: %.3f, circumference: %.3f", area, circumference);
    PI = 3;
}
```

تعریف یک ثابت

استفاده از PI

هرگونه تلاش برای تغییر PI ، ارور می دهد.

```
#define PI 3.141519
int main()
{
    float r, area, circumference;
    printf("enter radius: ");
    scanf("%f", &r );
    area = PI * r * r;
    circumference = 2 * PI * r;
    printf("Area: %.3f, circumference: %.3f", area, circumference);
}
```

دقت کنید و نوارد.

pointers

پوینترها متغیرهایی هستند که آدرس بقیه متغیرها را ذخیره می‌کنند.

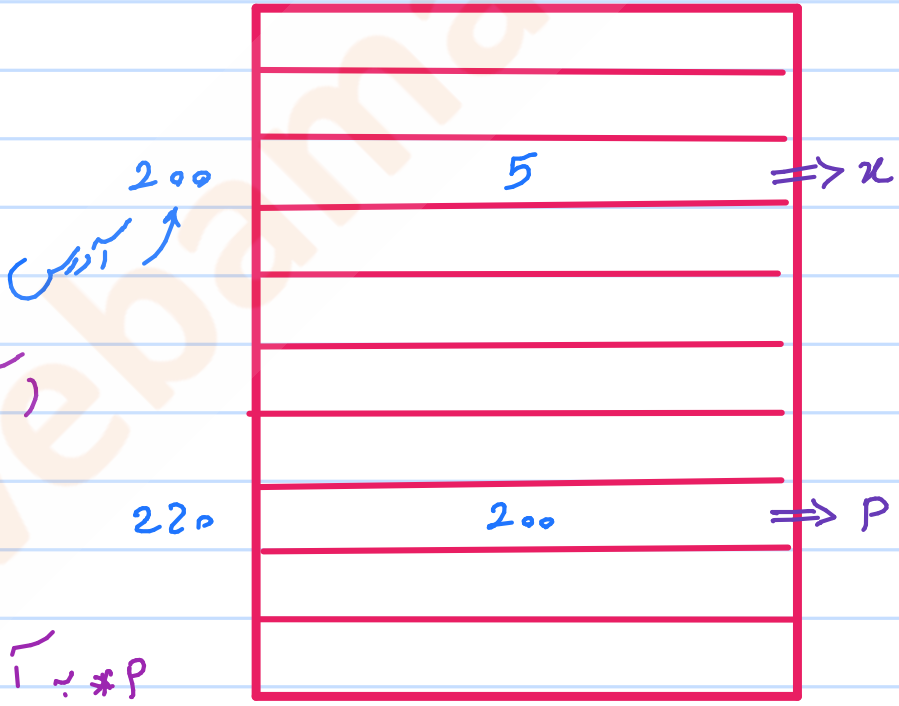
```
int x = 5;
```

```
int* p = &x;   یا   int *p = &x;
```

تعریف یک pointer که آدرس یک int (x) را ذخیره می‌کند.

```
print x  → 5   → مقدار x
print &x → 200 → آدرس x
print p  → 200 → مقدار p (آدرس x)
print &p → 220 → آدرس p
print *p → 5   → dereferencing
```

*p به آدرس ذخیره شده در p مراجعه کرده و محتوای آن را برمی‌گرداند

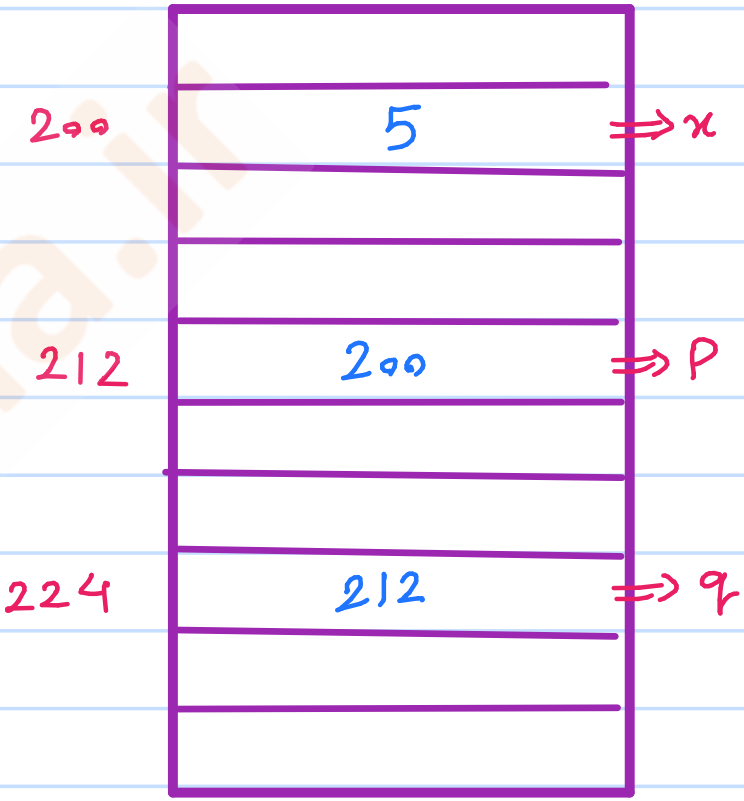


```
int x = 5 ;

int* p = &x ;

int ** q = &p ;
```

q هم یک پوینتر است. آدرس یک "int" آدرس یک (int*) را در خود ذخیره می کند.




```
print  x → 5
print  p → 200
print  *p → 5
print  q → 212
print  *q → 200
print  **q → 5
```

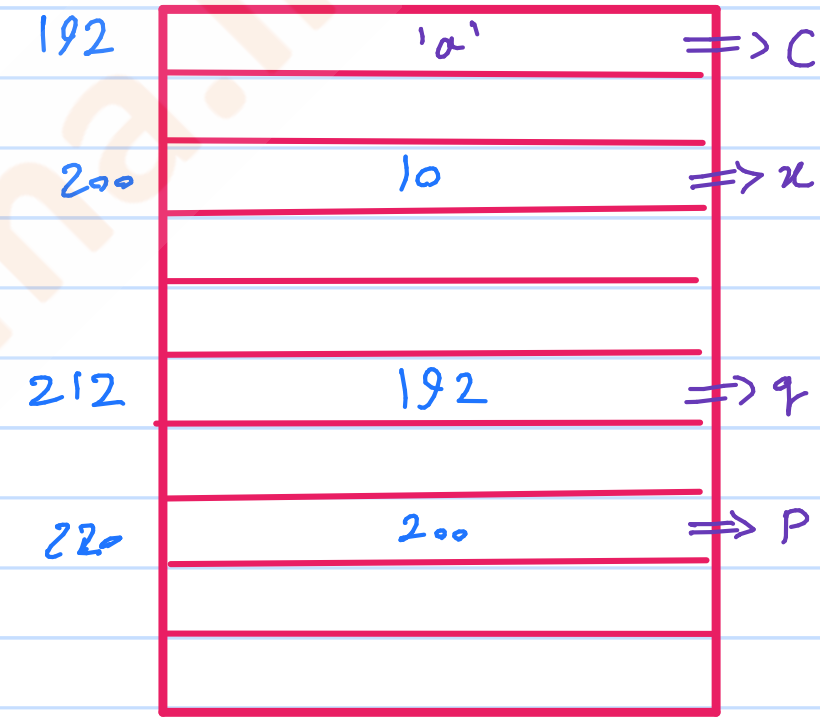
می‌توان به همین صورت پوینترهایی برای با هر مرتبه دلخواه ایجاد کرد. اما بزرگتر از 2 مرحله (**) کاربردی نیست.

نکته بعد اینکه هر پوینتر به آدرس یک نوع داده مشخص اشاره می‌کند.

```
float x = 6.28;
float* p = &x;
char a = 'a';
float *q = &a;
```

pointer arithmatic

```
char c = 'a';
int x = 10;
int *p = &x;
char *q = &c;
```



p ← آدرس یک int را ذخیره می‌کند.

q ← آدرس یک char را ذخیره می‌کند.

p → 200 ، p+1 → 204

چون آدرس int بعدی 200 می‌تواند 204 باشد. چون int، 4 بایت است.

p+2 → 208

q → 192 ، q+1 → 193 . چون char، یک بایت است.

"ضرب و تقسیم روی Pointer ها مجاز نیست"

سوال

اگر q = 200 باشد q+1 = ?

```
int x = 5;
int *p = &x;
int **q = &p;
```

q، آدرس یک int* را ذخیره می‌کند. int*، خود، آدرس است و به 64 بیت (8 بایت) فضا نیاز دارد. پس

q-1 → 192 ، q+1 → 208

Void pointer

می تواند آدرس مشروع هر نوع متغیری را ذخیره کند.

در مورد void pointer، pointer arithmatic مجاز نیست.

قبل از dereferencing، باید نوع داده مشخص شود.

```
int x = 5;
double y = 4.5;
char c = 'a';
bool b = true;

void * p = NULL;
p = &x;
p = &y;
p = &c;
p = &b;
```

می تواند به آدرس تمام انواع اشاره کند.

((int)p) → یعنی اول مقدار p را تبدیل به int* کن، بعد dereferencing انجام بده.

arrays and pointers

قبلاً گفته شده بود که نام آرایه به آدرس اولین المان آرایه اشاره می کند .

در واقع $a = \&a[0]$

```
int a[5] = {1, 2, 3, 4, 5};
```

print a → 200

print a+1 → 204

print a+2 → 208

⋮

آدرس اولین المان

200	1
204	2
208	3
212	4
216	5

print *a → 1

print *(a+3) → 4

print *(a+1) → 2

بهترین روش برای دادن ورودی آرایه به یک تابع استفاده از pointer ها است. در ضمن در C، اگر آرایه‌ای را به ورودی یک تابع بدهیم، باید تعداد المان‌ها یا در واقع طول آرایه را هم با یک متغیر دیگر به تابع بدهیم. چون C، راهی برای مشخص کردن آن ندارد.

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

int main() {
    int a[10] = {3, 0, 16, -9, 8, 1, -3, 6, 7, 10};
    int maximum = findmax(a, 10);
    printf("Maximum number is: %d\n", maximum);
    return 0;
}


int findmax(int* p, int size) {
    int max = *p;
    for (int i = 0; i < size; i++) {
        if (*(p+i) > max) {
            max = *(p+i);
        }
    }
    return max;
}
```

تابع findmax، یک پوینتر از نوع int* می‌پذیرد. یعنی آدرس یک int را.

هم‌چنین سایز آرایه هم به عنوان متغیر دوم به آن باید داده شود.

حال برای فراخوانی آن باید آدرس اولین المان آرایه را به آن بدهیم.

یعنی &a[0]. اما نام آرایه هم دقیقاً به همین آدرس اشاره دارد. یعنی در واقع a = &a[0]. پس برای فراخوانی از a و سایز (10)، استفاده می‌کنیم.

پس دادن ورودی آرایه به توابع با کمک pointer ها است. اما اگر از یک تابع خروجی آرایه بخواهیم چه؟

فرض کنید بخواهیم تابعی بنویسیم که از ما یک عدد صحیح را بگیرد و آرایه‌ای از اعداد صحیح به شکل زیر تولید کند:

$s = 5 \Rightarrow \{0, 1, 2, 3, 4\}$

ورودی — خروجی

در C داشتن خروجی آرایه برای یک تابع ممکن نیست. اما می‌توان یک مکانی را در حافظه ایجاد کرد و آدرس آن را به عنوان ورودی به تابع داد تا تابع مقادیر خروجی خود را در آن ذخیره کند. به مثال صفحه بعد دقت کنید.

```c
int main() {
    int s = 10;
    int a[s];
    intArrayCreator(s, a);
    printArray(a, s);
    return 0;
}

void intArrayCreator(int end, int* o) {
    for (int i = 0; i < end; i++) {
        *(o + i) = i;
    }
}

void printArray(int*a, int size) {
    printf("{");
    for (int i = 0; i < size; i++) {
        printf("%d,", *(a+i));
    }
    printf("\b}");
}
```

o در ورودی تابع intArrayCreator، درواقع آدرسی را در خود دارد که برای ذخیره خروجی ها از آن استفاده می شود. در واقع، گرچه به عنوان ورودی داده شده است، اما نقش خروجی تابع را دارد.

در این برنامه از یک تابع جداگانه برای پرینت آرایه هم استفاده شده است.

در صفحات پیش با pointer ها برای ذخیره آدرس متغیرها یا آرایه ها و کار با آنها آشنا شدیم.

در C می توانیم pointer هایی داشته باشیم که به یک تابع اشاره می کنند. آنها را function pointer می نامند. نحوه تعریف آن:

```
return_type (*name)(arguments);
name = &function;
```

↳ پرانتز ضروری است.

```
**********************
* HELLO WORLD *
**********************
**********************
* HELLO WORLD *
**********************
```

← خروجی

```c
void inBoxPrint(char* s) {
    int len = strlen(s);
    for (int i = 0; i < len + 4; i++) {
        printf("*");
    }
    printf("\n");
    printf("* ");
    printf("%s", s);
    printf(" *");
    printf("\n");
    for (int i = 0; i < len + 4; i++) {
        printf("*");
    }
    printf("\n");
}

int main()
{
    char s[30] = "HELLO WORLD";
    inBoxPrint(s);

    void (*myPrinter)(char*);
    myPrinter = &inBoxPrint;
    (*myPrinter)(s);

    return 0;
}
```

← روش مرسوم و همیشگی صدا زدن تابع

← تابع ها void برمی گردانند و ورودی char* دارند.

← function pointer به آدرس تابع inBoxPrint اشاره دارد.

← صدا زدن تابع توسط function pointer

function pointer ها این امکان را برای شما فراهم می‌کنند تا با توابع هم مثل متغیرها کار کنید. مثلاً توابع را به عنوان ورودی به یک تابع دیگر بدهید.

```
void func(void (*f)(int)) {
}
```

→ f. یک pointer به تابع است که void را برمی‌گرداند و ورودی آن از نوع int است.

در صفحات بعد، مثال‌هایی قرار داده شده است.

```
void inBoxPrint(char* s) {
    int len = strlen(s);
    for (int i = 0; i < len + 4; i++) {
        printf("*");
    }
    printf("\n");
    printf("* ");
    printf("%s", s);
    printf(" *");
    printf("\n");
    for (int i = 0; i < len + 4; i++) {
        printf("*");
    }
    printf("\n");
}
```

1 2

```
void toUpperCasePrinter(char* s, void (*f)(char*)) {
    int len = strlen(s);
    for (int i = 0; i < len + 4; i++) {
        if (s[i] > 96 && s[i] < 123)
            s[i] -= 32;
    }
    (*f)(s); ③
}
```

```
int main()
{
    char s[30] = "Hello World";
    toUpperCasePrinter(s, inBoxPrint);
    return 0;
}
```

همان تابع inBoxPrint است و تغییری نکرده.

تابع toUpperCasePrinter، تابعی است که دو ورودی دارد:

1 ـ یک رشته کاراکتری

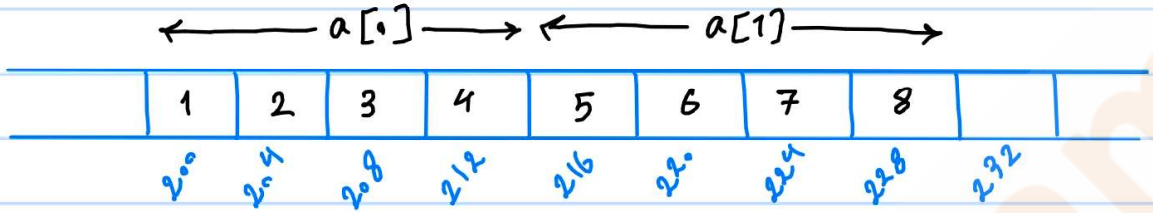
2 ـ آدرس تابعی که خروجی void و ورودی char* دارد.

برای فراخوانی f، آن را dereference کرده و ورودی آن را فراهم می‌کنیم. ③

در تابع main، toUpperCasePrint را فراخوانی می‌کنیم. این تابع دو ورودی نیاز داشت: یک رشته کاراکتری که همان s است و یک آدرس تابعی که ورودی char* و خروجی void دارد. inBoxPrint دقیقاً چنین تابعی است. مثل آرایه‌ها، نام تابع به آدرس اش اشاره دارد. پس نام تابع را به عنوان ورودی می‌دهیم.

```
int a[2][4] = { {1,2,3,4} , {5,6,7,8} }
```

هر درایه آرایه a خودیک آرایه است.



print a → 200 آدرس درایه ۰ آرایه a

print a+1 → 216 آدرس درایه ۱ آرایه a

print *a → 200 *a : درایه ۰ آرایه a خودیک آرایه است. لذا این عبارت آدرس اولین درایه آن آرایه را برخواهد گرداند

آدرس ↓ {1,2,3,4}

print *a+2 → 208

print *(a+1)+2 → 224

((a+i)+j) : مقدار المان زام از آرایه ام یک آرایه دوبعدی