اصول کامپیوتر ۱

مبانی کامپیوتر و برنامه‌سازی

«جلسه‌ی شانزدهم»

دانشکده‌ی علوم ریاضی - دانشگاه شهید بهشتی

نیم‌سال اول ۹۰-۱۳۸۹

مدرس: سید علی کتان‌فروش

مروری بر آنچه از زبان برنامه‌نویسی C/C++ آموخته‌ایم

هر دستور برنامه در یکی از رده‌های زیر قابل دسته‌بندی است:

- **تعریف متغیر Variable declaration**
- **نسبت‌دهی Assignment**
- **ساختار شرط** `if ... else`
- **حلقه‌ی تکرار** `for / while / do ... while`
- **جهش** `break / continue / return`
- **عملگرها و توابع**

تعریف متغیر Variable declaration

```
int n;
double x, S, max;
S = 0;
cin >> x;
max = x;
for ( n=0; x>0; n++ ) {
  S += x;
  if ( max < x )
    max = x;
  cin >> x;
}
cout << "Mean = " << S/n << endl;
cout << "Max = " << max << endl;
system("PAUSE");
```

نسبت‌دهی Assignment

```
int n;
double x, S, max;
S = 0;
cin >> x;
max = x;
for ( n=0; x>0; n++ ) {
  S += x;
  if ( max < x )
    max = x;
  cin >> x;
}
cout << "Mean = " << S/n << endl;
cout << "Max = " << max << endl;
system("PAUSE");
```

ساختار شرط

```
int n;
double x, S, max;
S = 0;
cin >> x;
max = x;
for ( n=0; x>0; n++ ) {
  S += x;
  if ( max < x )
    max = x;
  cin >> x;
}
cout << "Mean = " << S/n << endl;
cout << "Max = " << max << endl;
system("PAUSE");
```

حلقه‌ی تکرار

```
int n;
double x, S, max;
S = 0;
cin >> x;
max = x;
for ( n=0; x>0; n++ ) {
  S += x;
  if ( max < x )
    max = x;
  cin >> x;
}
cout << "Mean = " << S/n << endl;
cout << "Max = " << max << endl;
system("PAUSE");
```

عملگرها و توابع

```
int n;
double x, S, max;
S = 0;
cin >> x;
max = x;
for ( n=0; x>0; n++ ) {
  S += x;
  if ( max < x )
    max = x;
  cin >> x;
}
cout << "Mean = " << S/n << endl;
cout << "Max = " << max << endl;
system("PAUSE");
```

- دقیق‌تر آن است که دستورات نسبت‌دهی را نیز در رده‌ی عملگرها قرار دهیم.

- توابع و عملگرها از این حیث در یک دسته قرار می‌گیرند که عملگرها علیرغم شکل نگارش متفاوتی که با توابع دارند در عمل‌کرد همانند توابع دو متغیره هستند.

```
z = x + y;          z = ADD(x,y);
```

- اصطلاحاً به متغیرهای یک عملگر، عملوند (oprand) می‌گوئیم.

برخی از توابع C/C++ که تاکنون مورد استفاده قرار داده‌ایم

cmath	description
`sqrt(x)`	$\sqrt{x}$
`pow(x,y)`	x^y
`floor(x)`	$\lfloor x \rfloor$
`exp(x)`	e^x

cstdlib	description
`system(`***command***`)`	execute *command* in system command shell

برخی دیگر از توابع استاندارد C/C++

cmath	description
sin(x)	$\sin(x)$
cos(x)	$\cos(x)$
tan(x)	$\tan(x)$
atan(x)	$\arctan(x)$

cmath	description
fabs(x)	$\lvert x \rvert$
log(x)	$\ln(x)$
tanh(x)	$\tanh(x)$
atan2(y,x)	$\arctan(y/x)$

cstdlib	description
exit(*exit_code*)	terminate the process and return *exit_code* to parent process (operation system).
abs(n)	$\lvert n \rvert$
atoi(s)	return the integer value of string s.

برخی از عملگرهای C/C++ که تاکنون مورد استفاده قرار داده‌ایم

اولویت محاسبه

بیشترین

کمترین

operator	description
++ --	increase decrease
!	logical not
-	negative
* / %	multiply division remainder
+ -	add subtract
<< >>	inserter extractor
< <= > >= == !=	relational
&&	logical and
||	logical or
? :	conditional
= += -= *= /= %=	assignment compound assignment

فراخوانی توابع و عملگرها

Function / operator call

استفاده از یک تابع (یا یک عملگر) به عنوان یک دستورالعمل برنامه، مثل

```
cin >> x;
```

یا در بخشی از یک دستورالعمل، مثل

```
x = (-b+sqrt(d))/(2*a);
```

متضمن به اجرا درآمدن برنامه‌ی مرتبط با آن تابع یا عملگر است. اصطلاحاً به این فرایند، فراخوانی تابع (یا عملگر) می‌گوئیم.

فراخوانی هر تابع (یا عملگر)، روال اجرای برنامه را به دنباله‌ی دیگری از دستورات منتقل می‌کند که با اجرای آن‌ها، عمل‌کرد تابع (یا عملگر) معنا می‌یابد. مثلاً

```
cin >> x;
```

متغیر x را با آنچه کاربر به عنوان ورودی برنامه وارد می‌کند مقداردهی می‌کند.

- به یاد داشته باشید که cin یک دستورالعمل زبان C/C++ نیست بلکه متغیری از یک نوع داده‌ای پیشرفته است که توسط کتابخانه‌ی iostream تعریف می‌شود.

مقدار بازگشتی توابع و عملگرها

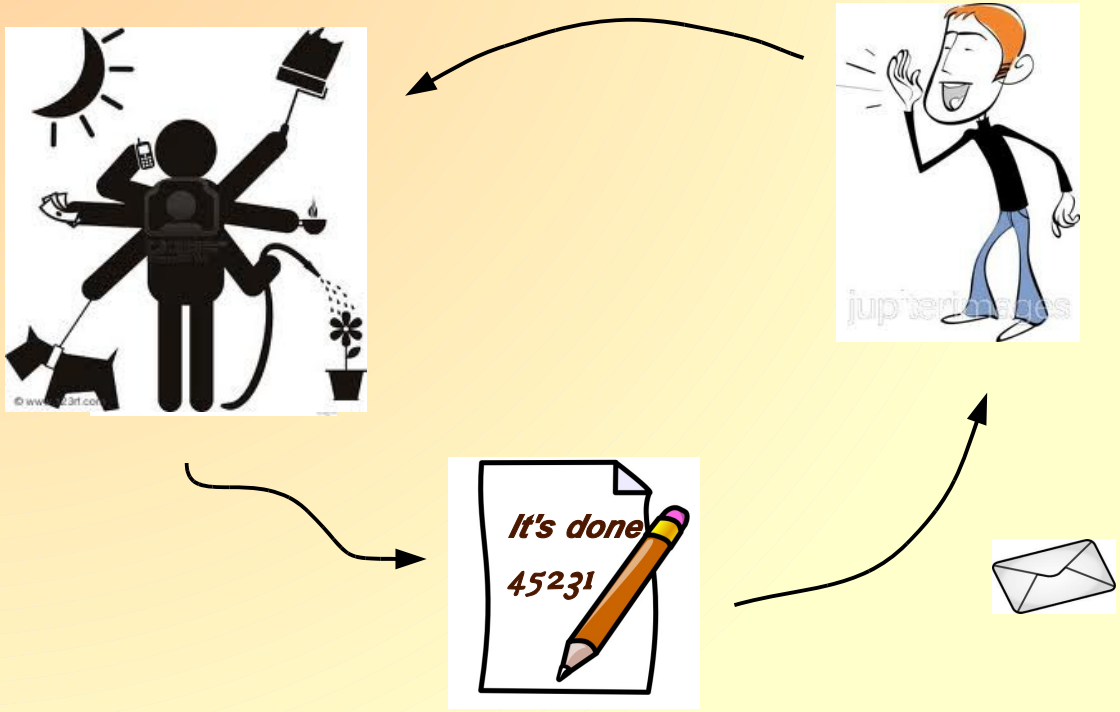
Function / operator returned value

با فراخوانی هر تابع (عملگر)، عملیات معینی اجرا می‌شود که اصولاً در پایان این عملیات، نتیجه‌ی مطلوب بدست می‌آید.

معمولاً توابع (عملگرها) در آخرین گام اجرای عملیاتشان، نتیجه‌ی محاسبات انجام شده را به برنامه‌ی فراخواننده ارسال می‌کنند.

اصطلاحاً به این مقدار، مقدار بازگشتی تابع (عملگر) می‌گوئیم یعنی مقداری که طی بازگشت روال اجرای برنامه از تابع به برنامه‌ی اصلی، به برنامه‌ی فراخواننده ارسال می‌شود.

فراخوانی و مقدار بازگشتی



فراخوانی و مقدار بازگشتی

```
if(x*x<1)
  r=1;
else {
  t=f2(x);
  ....
  ....
  r = 1+t*y;
}
```

```
z = f (x,y);
```

```
z = f (x,y);
```

```
return r;
```

بکارگیری مقدار بازگشتی در برنامه‌ی فراخواننده

- برنامه‌ی فراخواننده به دلخواه مجاز است از مقدار بازگشتی تابعی که فراخوانی کرده است استفاده کند یا آنرا بدون استفاده رها کند.
- به عنوان مثال، دستور زیر بنابر قراردادهای زبان C/C++ از نظر کامپایل و اجرا کاملاً معتبر است. هرچند فراخوانی و عدم استفاده از مقدار بازگشتی تابع `sqrt` از نظر منطقی توجیه‌پذیر نیست!

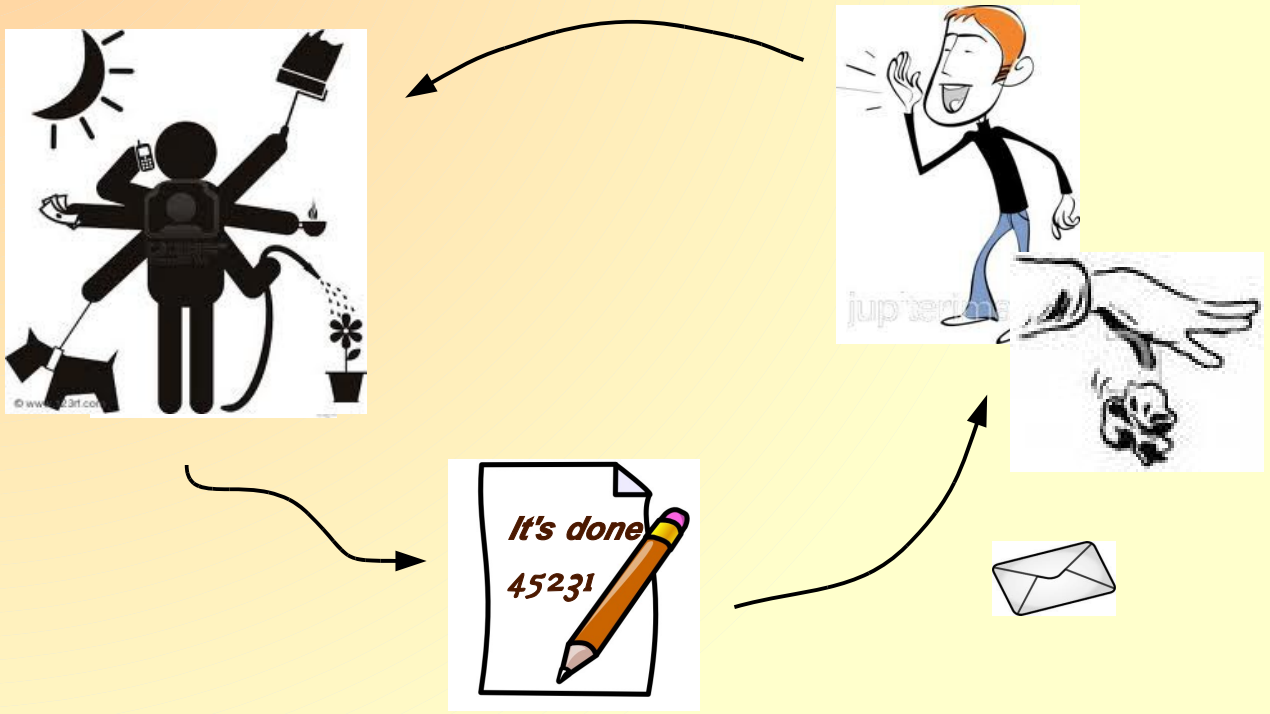
```
sqrt(x);
```

بکارگیری مقدار بازگشتی در برنامه‌ی فراخواننده

- به طور مشابه، بنابر ویژگی‌های زبان C/C++ برنامه‌نویس مجاز است تا عملگری را فراخوانی کند اما از مقدار بازگشتی (نتیجه‌ی) آن هیچ استفاده‌ای نکند.
- به عنوان مثال، دستور زیر از نظر کامپایل و اجرا کاملاً معتبر است. هرچند فراخوانی و عدم استفاده از نتیجه‌ی حاصلجمع منطقاً توجیه‌پذیر نیست!

```
x+y;
```

فراخوانی و مقدار بازگشتی



فراخوانی و مقدار بازگشتی

```
x+y;
```

```
MOV EAX,X
MOV EBX,Y
ADD EAX,EBX
```

```
PUSH EAX
```

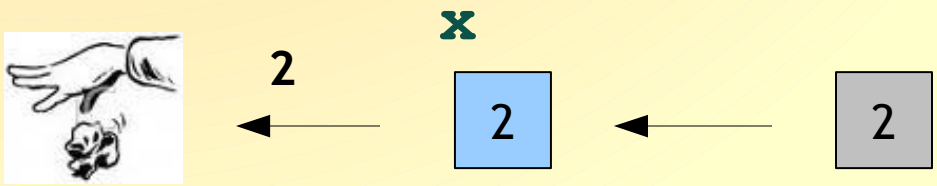
بکارگیری مقدار بازگشتی در برنامه‌ی فراخواننده

- با اینکه عدم استفاده از مقدار بازگشتی در بسیاری از موارد، غیرمنطقی به نظر می‌رسد اما برنامه‌نویس در بسیاری از مواقع از این ویژگی زبان C در عمل سود می‌برد.
- در چنین مواردی، هدف برنامه‌نویس از فراخوانی تابع (یا عملگر) به اجرا درآوردن عملیات آن تابع (یا عملگر) است و مقداری که به عنوان نتیجه‌ی عملیات باز گردانده می‌شود برای برنامه مهم نیست.

بکارگیری مقدار بازگشتی در برنامه‌ی فراخواننده

- برای درک موضوع باید بدانید که به عنوان مثال، عملگری همچون = (نسبت‌دهی)، پس از کپی کردن مقدار عبارت سمت راست در متغیر سمت چپ، مقدار کپی شده را به عنوان مقدار بازگشتی به برنامه‌ی فراخواننده باز می‌گرداند.

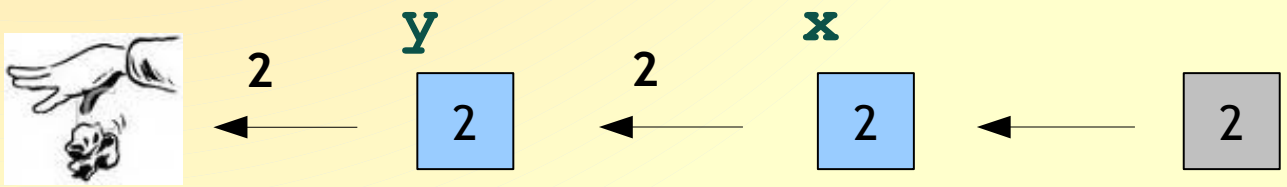
```
x = 2;
```



بکارگیری مقدار بازگشتی در برنامه‌ی فراخواننده

- ویژگی زبان C به ما اجازه می‌دهد از مقدار بازگشتی عملگر نسبت‌دهی استفاده نکنیم آنچنانکه معمولاً همین کار را می‌کنیم.
- این بدان معنی نیست که حتماً باید مقدار بازگشتی این عملگر را بدون استفاده رها کنیم. به عنوان مثال می‌توانید از مقدار بازگشتی عملگر نسبت‌دهی به منظور نسبت‌دهی‌های سریالی استفاده کنید.

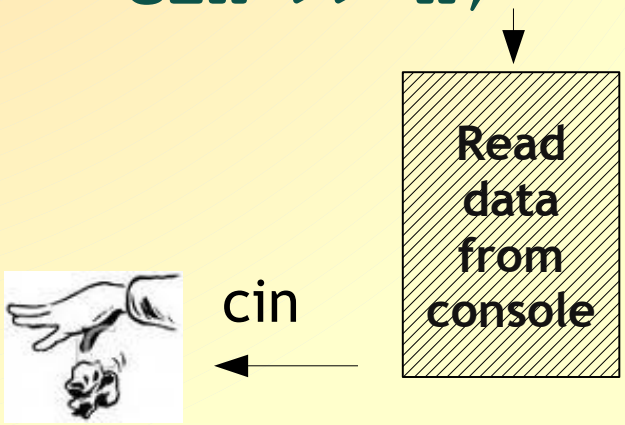
```
y = x = 2;
```



بکارگیری مقدار بازگشتی در برنامه‌ی فراخواننده

- به عنوان مثالی دیگر، عملگرهای << و >> را در نظر بگیرید.
- هر دو عملگر، پس از انجام عملیات ورودی/خروجی مورد درخواست، عملوند سمت چپ را به عنوان مقدار بازگشتی به برنامه‌ی فراخواننده باز می‌گردانند که معمولاً مورد استفاده قرار نمی‌گیرد.

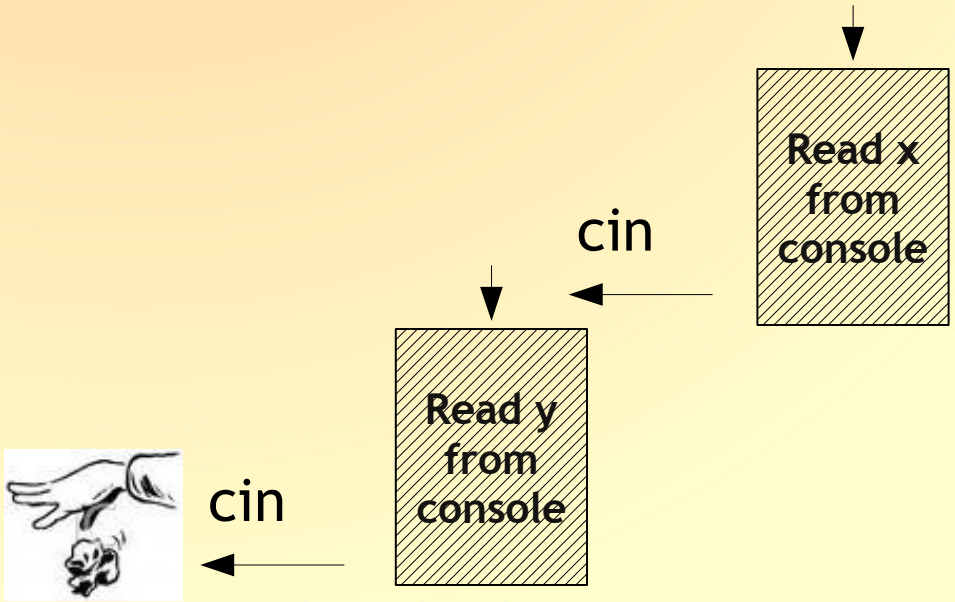
```
cin >> x;
```



بکارگیری مقدار بازگشتی در برنامه‌ی فراخواننده

- وقتی عملگرهای >> و << را به طور سریالی فراخوانی می‌کنیم درواقع از مقدار بازگشتی آخرین فراخوانی به عنوان عملوند فراخوانی بعدی استفاده می‌کنیم.

```
cin >> x >> y;
```



برخی اشتباهات رایج در برنامه‌نویسی مرتبط با موضوع مقدار بازگشتی عملگرها

- می‌دانیم که برای ارزیابی تساوی دو مقدار در زبان C باید از عملگر == استفاده کنیم. اما اگر در برنامه‌ای به اشتباه دستوری مانند دستور زیر را بکار برید با خطای نگارشی (syntax error) مواجه نمی‌شوید!

```
if ( x = y ) ....
```

- اما عمل‌کرد دستور اساساً با منظور برنامه‌نویس تفاوت دارد. در این دستور، ابتدا مقدار متغیر y در متغیر x نوشته می‌شود و این مقدار به عنوان مقدار بازگشتی (نتیجه‌ی محاسبه) به دستور if داده می‌شود که اگر صفر باشد به عنوان false در نظر گرفته می‌شود و اگر غیر صفر باشد به عنوان true تلقی می‌شود.

برخی اشتباهات رایج در برنامه‌نویسی مرتبط با موضوع مقدار بازگشتی عملگرها

```
int x = 0;
int y = 0;
if ( x = y )
  cout << "Equal" << endl;
else
  cout << "Different" << endl;
```

```
int x = 99;
int y = 100;
if ( x = y )
  cout << "Equal" << endl;
else
  cout << "Different" << endl;
```

برخی اشتباهات رایج در برنامه‌نویسی مرتبط با موضوع مقدار بازگشتی عملگرها

- در مقابل، در مثال زیر استفاده از == به عنوان عملگر نسبت‌دهی (به اشتباه) باعث بروز خطای نگارشی نمی‌شود!

```
x == y;
```

- در عمل، بین مقادیر دو متغیر x و y مقایسه صورت می‌گیرد و نتیجه‌ی مقایسه (true یا false) هر چه باشد بدون استفاده رها می‌شود و در عمل، هیچ مقداری از y به x کپی نمی‌شود. بدین ترتیب آنچه خواست برنامه‌نویس بوده است به انجام نمی‌رسد.