دانشگاه بین المللی امام خمینی قزوین

دانشکدهٔ فنی و مهندسی

مبانی کامپیوتر و برنامه سازی

(دومین ترم کرونا!)

فصل یازدهم: توابع در C

نستوه طاهری جوان

nastooh@aut.ac.ir

مقدمه

- ✓ تابع چیست؟
  - زیربرنامه ای که با هدف خاصی نوشته شده است! و می تواند بارها فراخوانی شود.
- ✓ انواع تابع در زبان C
  - توابع کتابخانه ای از پیش تعریف شده
    - پیاده سازی اعمالِ متداول در برنامه نویسی جهت سهولت کار برنامه نویسان
    - جلوگیری از اختراع دوبارهٔ چرخ!!!
    - مانند printf() و strcat() و tan()
  - توابع تعریف شده توسط برنامه نویس
    - موضوع این فصل از درس

nastooh@aut.ac.ir

مقدمه

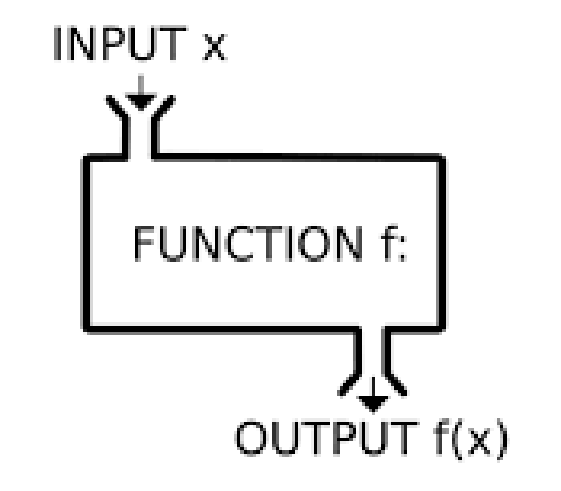
- ✓ مزیت استفاده از توابع نوشته شده توسط برنامه نویس
  - شکستن برنامه به قطعات کوچک
    - حل راحت تر مسائل پیچیدهٔ دنیای واقعی
    - خوانایی بیشتر برنامه های نوشته شده
  - توسعه تیمی برنامه ها
    - برنامه نویسی گروهی!
  - استفاده مجدد از کد
    - به حالتی فکر کنید که یک محاسبهٔ خاص، بارها باید صورت پذیرد.
  - تجرد
    - آیا بپرسیم یک کد چگونه کار می کند؟ یا یک کد چه کار می کند؟؟

nastooh@aut.ac.ir

مقدمه

- مثال زیر را در نظر بگیرید:
  - پزشک خانوادگی، پس از معاینه اگر نیاز به رادیولوژی داشته باشد، فقط بیمار را نزد رادیولوژیست می فرستند و انتظار دارد یک گزارش از رادیولوژیست دریافت کند.

- بیان شماتیک یک تابع ساده:



nastooh@aut.ac.ir

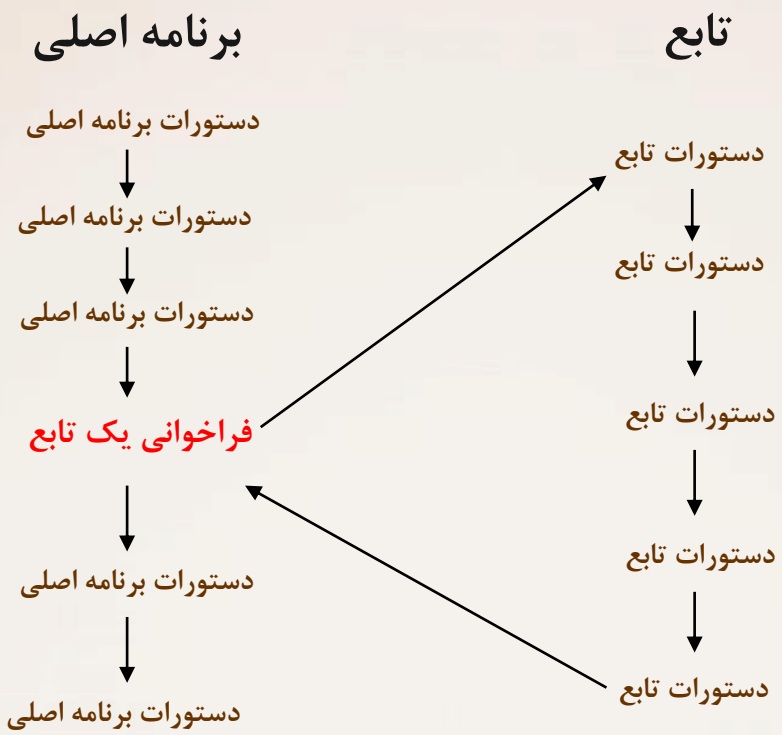
مقدمه

- ✓ توابع را باید ابتدا تعریف کرد
  - یعنی نحوهٔ عملکرد آنها را مشخص کرد.
- ✓ سپس می توان آنها را فراخوانی کرد (صدا زد).
  - فراخوانی تابع همواره با نام آن صورت می پذیرد.
  - معمولاً (نه لزوما)، توابع محاسباتی را انجام داده و مقداری را برمیگردانند.

nastooh@aut.ac.ir

طرز کار توابع

- روال اجرای برنامه هنگام فراخوانی توابع



nastooh@aut.ac.ir

تعریف توابع

✓ نحوهٔ تعریف یک تابع

```
نوع دادهٔ بازگشتی تابع   نام تابع ( نام و نوع پارامتر اول , نام و نوع پارامتر دوم , ...)
{
    دستورات بدنهٔ تابع
    return مقدار بازگشتی تابع ;
}
```

- نکته یک: هر برنامه می تواند هنگام فراخوانی، مقادیری برای تابع ارسال کند. این مقادیر همان "پارامترها" هستند. در واقع پارامترها وسیله ای برای تبادل داده بین تابع فراخوانی کننده و تابع فراخوانی شده هستند.
- نکته دو: هنگام فراخوانی باید با روشی مقادیر پارامترها را به تابع منتقل کرد.
- نکته سه: در واقع مقدار خروجی یک تابع در نام تابع ذخیره می شود.

nastooh@aut.ac.ir

تعریف توابع

- نحوهٔ تعریف یک تابع
  - مثال:

این تابع سه مقدار ورودی را از تابع فراخوانی کننده دریافت می کند، اولی int، دومی int و سومی float.

خروجی این تابع از نوع int خواهد بود.

```
int func1(int a, int b, float c)
{
    دستورات تابع
    دستورات تابع
    return x;
}
```

خروجی این تابع مقدار x خواهد بود

nastooh@aut.ac.ir

تعریف توابع

- نحوهٔ تعریف یک تابع
  - توابع را باید خارج از تابع main() تعریف کرد.
  - مثال:

```
#include<stdio.h>

int  area (int  a, int  b)
{
    .....
    return  x;
}

void main()
{
    ...
}
```

تعریف عملکرد تابع **area**

حال می توان در داخل تابع **main** تابع **area** را فراخوانی کرد

nastooh@aut.ac.ir

تعریف توابع

- ✓ نحوهٔ تعریف یک تابع
  - هنگام تعریف تابع باید با دقت مشخص کنیم چه داده هایی باید به تابع ارسال شود (از طریق پارامترها) و چه مقداری باید برگشت داده شود.
  - در داخل تابع می توان متغیرهایی برای انجام محاسبات تعریف کرد.
  - هنگام فراخوانی یک تابع، باید به دقت مقادیری که باید به تابع ارسال شود را مشخص کرد.
  - دقت کنید خروجی یک تابع در نام آن ذخیره می شود. پس در تابع فراخوانی کننده باید نام تابع را به درستی استفاده کرد.
  - به مقادیری که هنگام فراخوانی به تابع ارسال می شوند، آرگومان گویند.

nastooh@aut.ac.ir

فراخوانی توابع

✓ برای فراخوانی توابع، از نام آنها استفاده می کنیم.

```
#include<stdio.h>

int  area (int  a, int  b)
{
     int  c;
     c = a * b;
     return  c;
}

void main()
{
     int  x, y, z;
     scanf (“%d%d”, &x, &y);
     z = area (x, y);
     printf (“%d”, z);
}
```

- مثال: برنامه ای که مساحت مستطیل را به کمک یک تابع محاسبه می کند.

هنگام فراخوانیِ تابع، مقادیر آرگومانهای x و y به ترتیب در پارامترهای a و b کپی می شوند.

پس از اتمام اجرای تابع، مقدار خروجی آن در نام تابع ذخیره شده است که ما آن را به z نسبت داده ایم.

nastooh@aut.ac.ir

تعریف توابع

- می توان جهت خوانایی بیشتر، تعریف توابع را بعد از main قرار داد، اما در این صورت باید حتما prototype تابع را فقط قبل از تابع main قرار داد.

```
#include<stdio.h>
int  func1 (int  a, int  b);
char  func2 (char ch,  float f);

void main()
{
      دستورات برنامه اصلی
}

int  func1 (int  a, int  b)
{
    دستورات تابع اول
}
char func2 (char ch,  float f)
{
    دستورات تابع دوم
}
```

در واقع پیش تعریف دقیقا همانند خط اول تعریف تابع است.

انتهای پیش تعریف باید حتما سمی کالن گذاشت!

nastooh@aut.ac.ir

استفاده از توابع

✓ مثال: محاسبهٔ فاکتوریل توسط تابع:

```
#include<stdio.h>
double fact (int  a);
void main()
{
    int  num;
    double result;
    printf (“please enter a number:”);
    scanf(“%d”, &num);
    result = fact (num);
    printf(“%f”, result);
}
double fact (int a)
{
    double  r = 0;
    int  i;
    for (i = 1 ; i <= a ; i++)
        r *= i;
    return  r;
}
```


nastooh@aut.ac.ir

استفاده از توابع

✓ نکته مهم

- توابع می توانند مقدار خروجی نداشته باشند.
  - یعنی فقط محاسباتی را انجام دهند، و یا نتایج کار را خودشان چاپ کنند.
  - در این صورت نوع تابع void انتخاب می شود.
  - در این حالت انتهای تابع نیازی به return نیست.
  - هنگام فراخوانی تایع نیز، نیازی نیست آن را به متغیری نسبت دهیم.

```
void circle (float  r)
{
    float   s;
    s = r * r * 3.14;
    printf (“%f”,  s);
}

void main()
{
     circle (5);
}
```

چون تابع خروجی ندارد، در انتهای تعریف تابع، return نداریم.

چون تابع خروجی ندارد، هنگام فراخوانی به متغیری نسبت داده نشده است!

nastooh@aut.ac.ir

استفاده از توابع

✓ نکته مهم

- توابع می توانند مقدار ورودی هم نداشته باشند.
  - در این صورت پارامترها void انتخاب می شود.
  - داشتن یا نداشتن ورودی در توابع، ارتباطی به داشتن یا نداشتن خروجی ندارد!

```
float  circle (void)
{
    float   s, r;
    scanf (“%f”, &r);
    s = r * r * 3.14;
    return  s;
}
```

```
void  circle (void)
{
    float   s, r;
    scanf (“%f”, &r);
    s = r * r * 3.14;
    printf (“%f”,  s);
}
```


nastooh@aut.ac.ir

استفاده از توابع

- مثال: با دقت قطعه کد زیر را تحلیل کنید.

```
#include<stdio.h>
void circle (void);
void main()
{
    ....
    circle();
    ....
}

void circle (void)
{
    float  r,  s;
    scanf (“%f”, &r);
    s = r * r * 3.14;
    printf (“%f”,  s);
}
```


nastooh@aut.ac.ir

ارسال پارامتر به تابع

- روش های ارسال پارامتر به تابع
  - روش اول: call by value
    - در این روش هنگام فراخوانی، یک کپی از مقدار آرگومان ها در پارامتر ها قرار می گیرد.
    - تغییر در مقدار پارامترها درون تابع، هیچ تاثیری بر آرگومان ها ندارد.

```
#include<stdio.h>
void func (int  a);
void main()
{
    int  x = 10;
    func (x);
    printf (“%d”, x);
}

void func (int  a)
{
    a = 20;
    printf (“%d”,  a);
}
```

خروجی: **2010**

nastooh@aut.ac.ir

ارسال پارامتر به تابع

- روش های ارسال پارامتر به تابع
  - روش دوم: call by reference
    - به این روش فراخوانی با آدرس هم گویند.
    - در واقع آدرس آرگومان به تابع ارسال می شود.
    - ***به نحوهٔ تعریف و استفاده از پارامتر ها دقت کنید.***

```
#include<stdio.h>
void func (int  *a);
void main()
{
    int  x = 10;
    func (&x);
    printf (“%d”, x);
}

void func (int  *a)
{
    *a = 20;
    printf (“%d”,  *a);
}
```

خروجی: **2020**

nastooh@aut.ac.ir

استفاده از تابع

- تمرین:
  - برنامه ای بنویسید که طول و عرض مستطیل را در تابع main() خوانده و محاسبات مربوط به محیط و مساحت مستطیل را در دو تابع جداگانه انجام دهد و کماکان نتیجهٔ محیط و مساحت را در تابع main() نمایش دهد.

nastooh@aut.ac.ir

ارسال پارامتر به تابع

- ارسال آرایه به تابع

  - به نظر شما برای ارسال یک آرایه به یک تابع، از کدام روش استفاده کنیم بهتر است؟

ارسال با ارجاع یا ارسال با مقدار؟؟

nastooh@aut.ac.ir

ارسال پارامتر به تابع

- ارسال آرایه به تابع
  - ارسال آرایه به تابع با ارجاع انجام می شود.
    - پس هر تغییری بر روی آرایه اصلی اعمال می شود.
  - دو روش ارسال آرایه به تابع
    - با طول مشخص
    - با طول نامشخص

```
void  f1 ( int b [ ]);
void  f2 ( int a [40]);

void  main (   )
{
   int   x [40];

    f1(x);
    f2(x);
}
```


nastooh@aut.ac.ir

ارسال پارامتر به تابع

- ارسال آرایه به تابع
  - با توجه به ماهیت آرایه ها، مرسوم است که سایز آرایه را نیز به صورت جداگانه به تابع ارسال کنند.

```
void  f1 ( int b[ ],  int size)
{
......
}

void  main (   )
{
   int   x [20];
   int  s = 20;
....

    f1(x, s);
}
```


nastooh@aut.ac.ir

ارسال پارامتر به تابع

- ارسال آرایه به تابع
  - از آنجا که نام آرایه خود از جنس اشاره گر است، می توان به صورت زیر نیز آرایه را ارسال کرد.

```
void  f1 (int *b,  int size)
{
....
   for (i=0; i<size ;  i++)
   {
      &b = 0;
      b++;
   }
}
void  main (   )
{
    int   x [20];
    f1(x, s);
}
```


nastooh@aut.ac.ir

ارسال پارامتر به تابع

- ارسال آرایه دوبعدی به تابع

- تمرین:
  - **در مورد ارسال آرایه با ابعاد بیشتر به تابع تحقیق کرده و برای خود مثال بزنید.**

nastooh@aut.ac.ir

متغیرهای سراسری و محلی

- متغیرهای محلی (Local)
  - متغیرهایی که در داخل توابع تعریف شوند، محلی نام دارند.
  - این متغیرها فقط در داخل همان تابع قابل استفاده هستند.
    - حتی متغیرهای تابع main().
- متغیرهای سراسری (Global)
  - متغیرهایی که خارج از توابع و قبل از main() تعریف شوند، سراسری هستند.
  - این متغیرها در کل برنامه و تابع ها قابل دسترسی هستند.

nastooh@aut.ac.ir

متغیرهای سراسری و محلی

✓ مثال: قطعه کد زیر را تحلیل کنید.

```
#include<stdio.h>
int  x;
void func (void);
void main()
{
    x = 10;
    printf (“%d”, x);
    func ();
    printf (“%d”, x);
}

void func (void)
{
    x = 20;
    printf (“%d”,  x);
}
```

خروجی: **102020**

nastooh@aut.ac.ir

متغیرهای سراسری و محلی

- ✓ نکته یک: مقدار اولیه متغیرهای سراسری همواره صفر است، حتی اگر مشخص نشود.
- ✓ نکته دو: مقدار اولیه متغیرهای محلی نامشخص است، مگر اینکه مشخص شود.
- ✓ نکته سه: می توان در داخل یک تابع، متغیر محلی ای همنام با متغیر سراسری تعریف کرد.
  - در این صورت آن متغیر برای آن تابع، **محلی** است.

nastooh@aut.ac.ir

متغیرهای سراسری و محلی

✓ مثال مهم: قطعه کد زیر را تحلیل کنید.

```
#include<stdio.h>
int  x;
void func (void);
void main()
{
    x = 10;
    printf (“%d”, x);
    func ();
    printf (“%d”, x);
}

void func (void)
{
    int  x;
    x = 20;
    printf (“%d”,  x);
}
```

خروجی: **102010**

nastooh@aut.ac.ir

توابع بازگشتی

- سوال: آیا یک تابع می تواند خود را فراخوانی کند؟؟؟
  - **خوب فکر کنید...** آیا دور بینهایت از فراخوانی ها شکل نمی گیرد؟

```
#include<stdio.h>
void func (void);

void main()
{
    func ();
}

void func (void)
{
    printf("A");
    func();
}
```


nastooh@aut.ac.ir

توابع بازگشتی

- ✓ تعریف توابع بازگشتی (Recursive)
  - توابعی هستند که خود را فراخوانی می کنند.
  - باید با شرایطی تابع را نوشت که از تشکیل دور بی نهایت از فراخوانی ها جلو گیری شود.
    - در عمل فراخوانی تابع را باید مشروط کرد.
  - برای حل مسائلی که خاصیت استقرائی دارند، مناسب هستند.
    - به عنوان مثال، مسائلی که پاسخ یک مرحلهٔ آنها وابسته به پاسخ مراحل قبلی باشد.

nastooh@aut.ac.ir

توابع بازگشتی

✓ مثال: تابع فاکتوریل

```
int  fact (int  n)
{
    if (n <= 1)
        return  1;
    else
        return (n * fact(n-1));
}
```

○ فراخوانی با مقدار ۴:

fact (4) = 4 * fact (3)

3 * fact (2)

2 * fact (1)

1

nastooh@aut.ac.ir

توابع بازگشتی

- نکته مهم: دقت کنید در توابع بازگشتی با هر بار فراخوانی، متغیرهای محلی از نو ساخته می شوند.
  - در واقع یکی از ایرادهای توابع بازگشتی، مصرف بالای حافظه آنهاست.

- می توان تمام توابع بازگشتی را با حلقه نیز پیاده سازی کرد.

nastooh@aut.ac.ir

توابع بازگشتی

- ✓ مثال: تابع فیبوناچی
  - عنصر nام سری فیبوناچی را برمیگرداند.

```
int  bibo (int  n)
{
    if ((n == 1) || (n == 2))
        return  1;
    else
        return (fibo (n-1) + fibo (n-2));
}
```


nastooh@aut.ac.ir

توابع بازگشتی

- سوال مهم:

در صورتی که هر دو راه حل ممکن باشد، شما ترجیح میدهید از توابع بازگشتی استفاده کنید، یا حلقه ها؟

بحث کنید!

nastooh@aut.ac.ir

کلاس های حافظه

- کلاس های حافظه دو مورد را در مورد متغیرها مشخص می کنند:
  - Scope: حوزه متغیر
    - متغیر در چه جاهایی از برنامه قابل دسترسی است.
  - Life time: طول عمر
    - متغیر چه زمانی ایجاد شده و چه زمانی از حافظه بیرون می رود.
- چهار کلاس حافظه در زبان C
  - static
  - extern
  - auto
  - register
- هنگام تعریف متغیر، می توان یکی از این چهار مورد را جلوی نوع آن قید کرد.

nastooh@aut.ac.ir

کلاس های حافظه

- ✓ کلاس حافظهٔ static
  - ○ در حالت محلی
    - فقط و فقط در همان تابعی که تعریف می شوند قابل استفاده هستند.
    - مقدار پیش فرض آنها صفر است.
    - فقط و فقط یک بار مقدار دهی اولیه می شوند!
    - هنگام خروج از تابع، آخرین مقدار خود را حفظ می کنند!!

```
#include <stdio.h>
void  f (void)
{
    int  a = 5;
    static  int  b = 5;
    a++;  b++;
    printf (“%d%d”, a , b)
}
void main()
{
    f();
    f();
    f();
}
```

خروجی:

666768

nastooh@aut.ac.ir

کلاس های حافظه

- کلاس حافظهٔ static
  - در حالت محلی
    - می توان گفت از نظر حوزه متغیر مانند متغیرهای محلی هستند، اما از نظر طول عمر مانند متغیرهای سراسری هستند.
    - هنگامی که تابعی قرار است چندین و چند بار صدا زده شود و یک متغیر باید مقدار خود را حفظ کند، باید از این کلاس استفاده کرد.

nastooh@aut.ac.ir

کلاس های حافظه

- ✓ کلاس حافظهٔ static
  - در حالت سراسری
    - این متغیرها فقط در توابعی که بعد از آنها تعریف شده اند، قابل استفاده هستند.

```
#include<stdio.h>
void main()
{
   ....
}
static  int  a;
int  func1 (int  a, int  b)
{
   ....
}
static int  b;
char func2 (char ch,  float f)
{
    .....
}
```


nastooh@aut.ac.ir

کلاس های حافظه

- ✓ کلاس حافظهٔ register:
  - برنامه نویس با این کلاس حافظه به سیستم پیشنهاد می دهد در صورت امکان این متغیر را در ثبات های پردازنده جانمایی کند.

- در این حالت سرعت دستیابی به این متغیر به شدت افزایش می یابد.
- متغیرهای خاص و پر کاربرد، مانند شمارنده های حلقه ها، را می توان از این نوع کلاس تعریف کرد.
- دقت کنید در این صورت آدرس این متغیرها معنی ندارد.
- دقت کنید ثبات های پردازنده بسیار کم و انگشت شمار هستند و عموما ثباتِ آزاد عنصری کمیاب محسوب می شود.
- امروزه دیگر از این کلاس حافظه استفاده نمی شود.

nastooh@aut.ac.ir

کلاس های حافظه

- کلاس حافظهٔ auto:
  - متغیرهای محلی درون توابع به صورت پیش فرض از این نوع هستند.
    - نیازی به استفاده از کلمهٔ auto قبل از متغیرهای محلی نیست.
    - حوزه دسترسی به این متغیرها، همان تابع است.
    - این متغیرها در ابتدای اجرای تابع در حافظه ساخته می شوند و پس از اتمام اجرای تابع، از بین می روند.

کلاس های حافظه

- تمرین:
  - در مورد کلاس حافظهٔ extern تحقیق کنید.

nastooh@aut.ac.ir

تاملی در باب تابع main()

- main() نیز یک تابع عادی است که در واقع سیستم عامل آن را صدا می زند.
- تابع main() نیز می تواند یه مقدار را به سیستم عامل برگرداند.
  - در این صورت باید نوع مقدار را مشخص کرد و در انتهای main از return مناسب استفاده کرد.
- در کاربردهای عادی، تابع main() دادهٔ معنا داری به سیستم عامل برنمیگرداند.
  - کاربردهایی هست که تابع main() باید یک مقدار به سیستم عامل برگرداند.
- برخی از برنامه نویسان عادت دارند نوع تابع main() را int تعریف کرده و در انتها یک صفر به سیستم عامل برگردانند.
  - در این حالت اگر برنامه به درستی و تا انتها اجرا شود، صفر به سیستم عامل برمیگردد.

```
#include<stdio.h>
int  main()
{
    ....
    ....
    return  0;
}
```


nastooh@aut.ac.ir

تاملی در باب تابع main()

- تابع main() می تواند حتی ورودیهایی از سیستم عامل دریافت کند.
  - این ورودی ها را باید هنگام اجرای برنامهٔ نوشته شده در کنسول (خط فرمان)، به تابع main() پاس داد.
  - نحوهٔ انجام کار به این صورت است:

```
int  main(int  argc,  char *argv[ ])
{
    .....
    return 0;
}
```

  - که argc تعداد آرگومانهای ورودی حین اجرا را به خود میگیرد و مقدار آرگومانهای وارد شده به ترتیب در آرایهٔ رشته ای argv قرار میگیرند.
  - این آرگومانها هنگام اجرای برنامه در خط فرمان، جلوی نام برنامه باید تایپ شوند.

nastooh@aut.ac.ir

منابع

[1] P. Deitel , H. Deitel, **C How to Program**, 8th ed., 2016. (Download Link)

[2] B. W. Kernighan, D. M. Ritchie, **The C Programming Language**, 2nd ed., 1988. (Download Link)

برای دانلود کتاب ها، اسلایدها و نمونه پروژه های درسی به سایت www.nastooh.com بخش دانشجویان مراجعه کنید.

پایان

nastooh@aut.ac.ir