www.engclubs.net

بسم الله الرّحمن الرّحیم

برنامه سازی پیشرفته (C)

توسط: دکتر سید سعید آیت

دانشگاه پیام نور

طرح درس:

- درس ٤ واحدی جهت دانشگاه پیام نور
- ٤٠ ساعت (٢٠ جلسه ٢ ساعته در سایت یا (١+١) ساعته در کلاس و سایت.
- توصیه می شود که کلاس در سایت کامپیوتر بر گزار شود, در غیر این صورت پیشنهاد می شود قسمت اول هر جلسه در کلاس و قسمت دوم در سایت, جهت تمرین عملی مطالب تدریس شده, برگزار شود.

جایگاه درس

- این درس یکی از اصلی ترین دروس رشته های مهندسی کامپیوتر و علوم کامپیوتر می باشد.
- این درس پیش نیاز بسیاری از دروس رشته های مذکور می باشد
- از آنجا که یکی از اصلی ترین مهارتهای دانشجویان رشته های مذکور برنامه نویسی می باشد, این درس شایسته دقت و تمرین مناسب می باشد.

فهرست مطالب

- منظور از برنامه نويسي كامپيوتر
- تاريخچه مختصر برنامه نويسي
- مراحل نوشتن یک برنامه
- قالب کلی برنامه ها در زبان C
- انواع خطاهای برنامه نویسی

فهرست مطالب

- نمایش متن, دستور printf و کاراکترهای کنترلی
- آشنایی با مفهوم متغیرها و عملگرها
- چاپ مقدار متغیرها
- دستورات ورودي: scanf, getche, getch
- فرمت بندي خروجي
- تبدیل انواع
- تقدم عملگرها

فهرست مطالب

- تعینن طول میدان در دستورات scanf, printf
- تبدیل انواع داده ای به یکدیگر
- عملگرهای ++, --
- دستورات شرطی:if, if-else, switch-case
- دستور break
- حلقه ها: while, for, do-while
- حلقه های تودرتو

فهرست مطالب

- توابع (مفهوم, طرز تعریف)
- توابع کاربر- توابع کتابخانه ای
- متغیرهای محلی و سراسری
- توابع بازگشتی
- خوانایی برنامه
- آرایه ها
- دستور define
- آرایه های چندبعدی
- رشته ها

برنامه نویسی

- یک برنامه در واقع مجموعه ای از دستورات است که در حافظه ذخیره می شود و سپس کامپیوتر آنها را اجرا می کند.

چگونگی شکل گیری برنامه نویسی

- در کامپیوترهای اولیه برای انجام یک دستور خاص(مثلا جمع) ورودی ها به فرم مبنای ۲ به دستگاه داده می شد و سپس خروجی به صورت مبنای ۲ مشاهده می شد و بعد از آن دستورات بعدی انجام می شد.
- با استفاده ساختار فون نیومن کامپیوترهایی تولید شدند که قادر بودند دستورات را در حافظه ذخیره کنند و سپس آنها به طور خودکار و متوالیا اجرا شوند.
- به یک مجموعه دستورات که توسط کامپیوتر اجرا می شود برنامه گفته می شود.

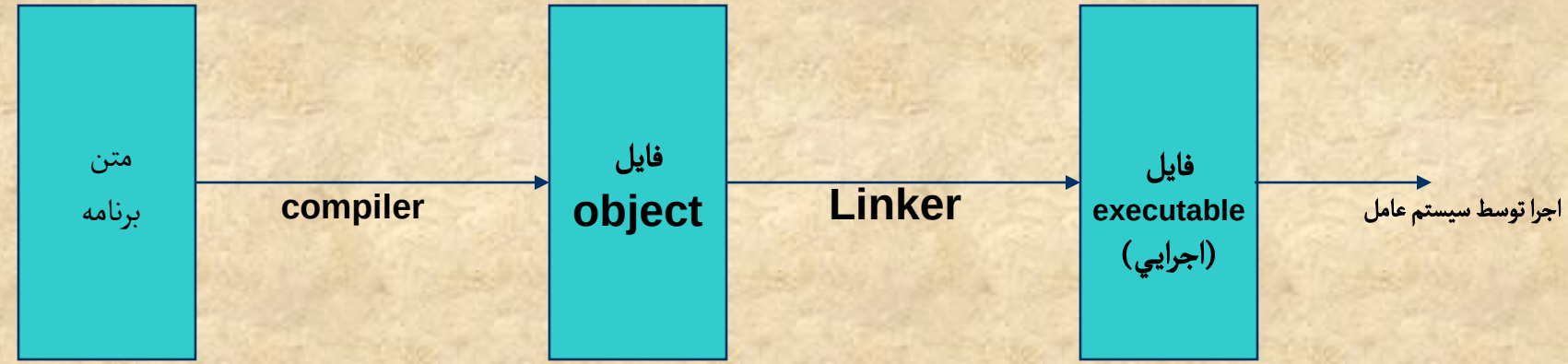
زبان ماشین و اسمبلی

- چون برنامه های اولیه به صورت کدهای دودوئی که مستقیما قابل فهم به زبان ماشین بود نوشته می شدند به این برنامه ها برنامه ها به زبان ماشین (machine language program) گفته می شد و به هر دستور، یک دستور زبان ماشین گفته می شد.
- مشکل نوشتن برنامه به زبان ماشین سختی نوشتن و ناخوانایی آن بود.
- برای رفع این مشکل زبانهای اسمبلی شکل گرفت. در این زبان ها برای هر دستور زبان ماشین یک عبارت تعریف شده است. مثلا برای جمع دو خانه حافظه A و B عبارت ADD A,B . در این حالت برنامه نویس به جای نوشتن یک سری ۰ و۱ ، با این عبارات برنامه خود را می نوشت.
- برنامه ای که برنامه اسمبلی نوشته شده توسط کاربر را به زبان ماشین تبدیل می کند اسمبلر خوانده می شود.

زبان های سطح بالا

- با وجود آنکه زبان های اسمبلی کار برنامه نویسی را آسان می کرد اما باز برنامه ها طولانی و معمولا ناخوانا بودند.
- برای رفع این مشکل زبان های سطح بالا بوجود آمدند. در این زبان ها هر چند دستور زبان ماشین به یک عبارت با معنا که به زبان معمولی نزدیک بود تبدیل می شود.
- دو برنامه کامپایلر(compiler) و linker روی هم کار تبدیل برنامه سطح بالا به زبان ماشین را انجام می دهند.

مراحل ایجاد یک برنامه



مراحل ایجاد برنامه سطح بالا: نوشتن متن برنامه ، کامپایل ، link و اجرا

به مجموع دو عمل compile و link اصطلاحا Build مي گوييم.

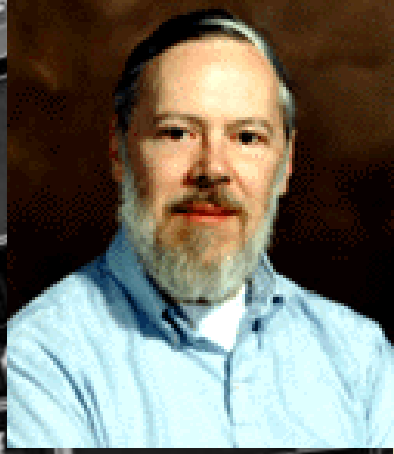
Build=compile+link Builder=compiler+linker

تاریخچه زبان های برنامه نویسی

- اولین زبان برنامه نویسی در ۱۹۴۶ توسط Plankalkül آمد و
- اولین زبان سطح بالا FORTRAN I بود که در ۱۹۵۴ بوجود آمد.
- به مرور زمان برنامه نویسی ساده تر و پیشرفته تر شد:
  - زبان های مجهز به امکانات گرافیکی :زبان های ویژوال : امکان نوشتن برنامه بدون تایپ دستورات: دهه ۱۹۶۰
  - زبان های شیء گرا : Simula(دهه ۱۹۷۰)
  - انجام کارهای شبکه ای و اینترنتی: java (۱۹۹۵)

زبان C

- موضوع درس آشنایی با زبان سطح بالای C است.
- این زبان در سال توسط ۱۹۷۲ توسط دنیس ریچی (Dennis Ritchie) بوجود آمد که بر اساس زبان دیگری به نام B شکل گرفته بود.
- B توسط Ken Thompson در ۱۹۶۹ بوجود آمد.



تصویری از Thompson و Rithchie (۱۹۷۲)

نکته

- در یک زبان عادی برای بیان یک مفهوم می توان از عبارات مختلفی استفاده کرد :
  - علی به خانه رفت.
  - علی خانه رفت.
  - علی رفت خانه.
  - رفت علی خانه.
- در یک زبان برنامه نویسی نیاز دستورات نیاز به بیان دقیق دارند و باید طبق ساختار مشخصی که در زبان مشخص شده استفاده شوند تا کامپایلر قادر به درک آنها باشد.

برنامه نویسی به زبان C

ساده ترین برنامه به زبان C

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

نکات:

1. خط اول در هر برنامه C باید وجود داشته باشد.
2. { : شروع برنامه
3. } : خاتمه برنامه
4. دستورات برنامه در داخل {} نوشته مي شوند.

نمایش مراحل سه گانه نوشتن ، compile ، link و اجرا

1. نوشتن
2. Compile
3. Link
4. اجراء

IDE

مشکل: زمانبر بودن

رفع مشکل: ارائه نرم افزارهايي که امکان ويرايش ، کامپايل ، link و اجرا را در يک محيط فراهم مي کنند.

IDE: Integrated Development Environment

مانند: Borland C ، Visual C و....

- اجراي برنامه در Visual C

نکات ۱–

- فاصله گذاري
- حساس بودن به حالت حروف (case sensitivity)

نكات–۲

- **error**: به خطاهاي برنامه نويسي **error** مي گويند.
- انواع خطاها در برنامه نويسي:
  - خطاهاي زمان **compile**(**compile errors**):
    - مانع كامپايل صحيح برنامه مي شوند.
  - خطاهاي زمان **link** (**Link errors**):
    - براي كامپايل مزاحمتي ايجاد نمي كنند اما مانع **Link** برنامه مي شوند.
  - خطاهاي زمان اجرا: (**Run time errors**):
    - كامپايل و **Link** با موفقيت انجام مي شود ولي اجراي برنامه دچار اشكال مي شود .

error

- حسن سیب را خورد.
- هسن سیب را خورد.
  - متناظر با خطاي كامپايل
- را حسن خورد سیب.
  - متناظر با خطاي Link
- سیب حسن را خورد.
  - متناظر با خطاي زمان اجرا

مثال ۲) نمایش متن بر روی مانیتور

برنامه اي بنويسيد كه پيغام Hello را در مانيتور نشان دهد.

```
#include <stdio.h>
void main()
{
   printf(“Hello”);
}
```

پيغام Hello چاپ مي شود و مكان نما بعد از حرف o قرار مي گيرد.

نکات

- Stdio.h نمونه اي از يک header file است. فايل هاي header جزئيات غير مرتبط با کاربر را از ديد او مخفي مي کند و موجب مي شود برنامه اي خلاصه تر و خواناتر داشته باشيم.

- دستور .... #include يک راهنماي پيش پردازش (preprocessor directive) يا راهنماي کامپايلر (compiler directive) خوانده مي شود. Compiler قبل از شروع کامپايل محتويات اين فايل را به برنامه اضافه مي کند و سپس کامپايل آغاز مي گردد.

- **در انتهای هر دستور زبان C داخل main علامت ; قرار داده می شود.**

نکات (ادامه)

- Printf(“Hello welcome”);

خروجي ؟؟

  - فاصله ها هم نمايش داده مي شوند.

- مي خواهيم خروجي ما به صورت زير باشد:

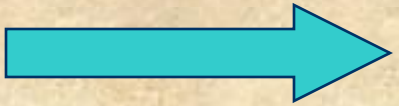
```
#include <stdio.h>
Void main( )
{
   Printf(“Hello
Welcome”);
}
```

خطا

```
#include <stdio.h>
Void main( )
{
   Printf(“Hello\nWelcome”);
}
```

\t

```
#include <stdio.h>
Void main( )
{
    Printf("Hello \tWelcome");
}
```



حرکت به اندازه یک tab

\n و \t اصطلاحا کاراکترهاي کنترلي نامیده مي شوند. کاراکترهاي کنترلي در printf عینا چاپ نمي شوند بلکه اثراتي دیگر از خود بروز مي دهند.

خلاصه مطالب قبل

- تعريف و تاريخچه مختصري از كامپيوتر
- مقدمه اي بر مفهوم برنامه نويسي و چگونگي شكل گيري آن
- نوشتن چند برنامه ساده به زبان C
- كاراكترهاي كنترلي : مانند \n و \t

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

ساده ترین برنامه

```
#include <stdio.h>
void main()
{
  printf("Hello");
}
```

چاپ متن روی مانیتور

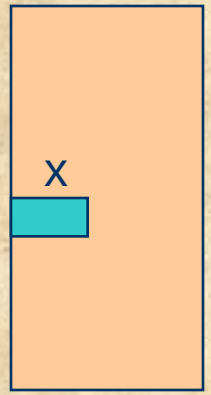
- آشنایی با مفهوم متغیرها، اپراتورها (عملگرها)
- چاپ مقدار متغیرها
- دستورات ورودی
- فرمت بندی خروجی
- تبدیل انواع
- تقدم عملگرها

مثال

- برنامه ای بنویسید که حاصل ۲+۲ را محاسبه کند.

```
void main( )
{
    int x;
    x=2+2;
}
```

نکات–۱

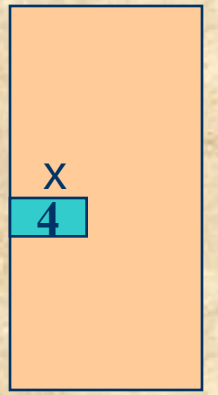


- int x;
  - هنگام اجرای این دستور:
    - یک مکان ۴ بایتی حافظه به برنامه اختصاص داده می شود.
    - این مکان را x می نامد.
    - به برنامه اجازه می دهد که یک عدد صحیح (integer) در x بریزد.
  - اصطلاحا به x ، یک متغیر (Variable) از نوع صحیح (int) گفته می شود.
  - به دستور بالا دستور تعریف متغیر می گوییم.
  - x نام متغیر می باشد.
  - نام متغیر
    - ترکیبی از حروف a تا z و A تا Z ، ارقام 0 تا 9 و _ (Undeline) می باشد.
    - نباید با یک رقم شروع شود.

    مثال : اسامی مجاز: a، asd، ali، A09 ، _A

    اسامی غیر مجاز: 9a، A-s

نکات–۲



- X=2+2;
  - ابتدا 2 با 2 جمع مي شود.
    - به علامت + اصطلاحا یک operator (عملگر) میگوییم.
    - هر یک از اعداد 2 یک operand (عملوند) خوانده میشوند.
  - سپس حاصل جمع در X قرار مي گيرد.

    یا اصطلاحا 4 به X تخصیص مي یابد.
  - به = اصطلاحا یک اپراتور تخصیص مقدار و به دستور بالا یک دستور تخصیص مقدار (Assignment statement) میگوییم.
  - درسمت چپ یک دستور تخصیص مقدار فقط و فقط نام یک متغیر باید باشد. سمت راست مي تواند ترکيبي از نام متغیر و مقادیر باشد.

بعضی انواع متغیرها

	مصرف حافظه (بایت)	نوع	محدوده
int	4	صحیح	$2^{31}-1$ تا -2^{31}
float	4	اعشاري	$\pm(10^{-37}-10^{38})$ ماکزیمم 6رقم اعشار دقت-
double	8	اعشاري با دقت بالا	$\pm(10^{-307}-10^{308})$ ماکزیمم 15 رقم اعشار دقت -
long (long int)	8	صحیح بزرگ	$2^{63}-1$ -2^{63}
short (short int)	2	صحیح کوچک	32,767تا -32,768
char	1	کاراکتر	127 تا -128

```
float x;
X=2.3;

short ys;
ys=123;
ys=0x23;
char c;
c='a‘;
c=97;
```

??

```
int x;
x=2.3;
```

```
short x;
x=3243243255556;
```

```
float x;
x=3;
```

بعضی انواع اپراتورها

- محاسباتی:

  + – * / % (باقیمانده) ++ --

- تخصیص مقدار:

  =

- مقایسه ای:

  > < == !=

صورت های دیگر نوشتن برنامه

```
Void main ()
{
    int x;
    int y;
    y=2;
    x=y+2;
}
```

```
Void main ()
{
    int x,y;
    y=2;
    x=y+2;
}
```

```
Void main ()
{
    int y=2;
    int x;
    x=y+2;
}
```

```
Void main ()
{
    int y=2,x;
    x=y+2;
}
```

```
void main( )
{
    int x;
    x=2+2;
}
```

```
Void main ()
{
    int z=2,y=2,x;
    x=z+y;
}
```

چرا متغیر

```
Void main ()
{
    int x=4;
    int y=5;
    x=2;
    y=x+2;
}
```

نمایش مقادیر در مانیتور

- مسئله: مي خواهيم مقدار يک متغير را روي مانيتور چاپ کنيم.

```
#include <stdio.h>
Void main ()
{
    int x=4;
    int y=5;
    x=2;
    y=x+2;
    printf("Result=%d",y);
}
```

%d يعني يک عدد صحيح را روي مانيتور نشان بده
اين عدد صحيح y خواهد بود.

%d نيز يک کاراکتر کنترلي مي باشد.

نکات

- چاپ بیش از یک مقدار

```
Void main ()
{
    int x=2,y=2,z;
    z=x+y;
    printf(“%d + %d =%d”,x,y,z);

}
```

كاراكترهاي كنترلي مشابه %d

%d	اعداد صحيح دهدهي	
%f	اعداد اعشاري	float x=23.2; printf(“%f”,x);
%c	كاراكتر	char ch=‘B’; char ch=66; printf(“%c”,ch);
%x	اعداد مبناي 16 صحيح	int a=0x234; printf(“%x”,a);

چند مثال

- printf(“%d”,23);
  - 23 **چاپ می شود.**
- printf(“%c”,’a’)
  - **کاراکتر** a **روی مانیتور چاپ می شود.**
- printf(“%d”,’a’);
  - **کد اسکی کاراکتر** a **یعنی ۹۷ چاپ می شود.**
- printf(“%c”,97)
  - **کاراکتری که کد اسکی آن ۹۷ است یعنی** a **چاپ می شود.**

دستورات ورودي

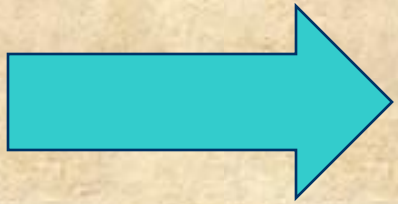
- با استفاده از دستورات ورودي مي توانيم مقدار متغيرها را از طريق keyboard وارد كنيم.

دستور scanf

برنامه اي بنويسيد كه عددي صحيح را از keyboard دريافت كند و مربع آن را چاپ كند.

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

علامت آدرس



يک عدد صحيح را از كاربر دريافت و آن را در متغير x مي ريزد.

نكات

- به طرزي مشابه **printf**
  - براي خواندن اعداد اعشاري: %f
  - خواندن كاركترها:%c
  - خواندن اعداد مبناي 16: %x
- براي استفاده از **scanf** هم بايد از **stdio.h** استفاده كرد.
- بعد از وارد كردن مقدار بايد **ENTER** زده شود.

مثال

- برنامه اي بنويسيد كه دو عدد را از صفحه كليد دريافت كرده و حاصل ضرب آنها را چاپ كند.

```
#include <stdio.h>
void main()
{
      int x,y;
   printf(“\nEnter two numbers”);
      scanf(“%d%d”,&x,&y);
      printf(“The sum=%d”,(x*y));
}
```

هنگام وارد كردن اعداد بايد بين آنها لااقل يک فاصله قرار داده شود يا ENTER زده شود.
بعد از وارد كردن عدد آخر بايد ENTER زده شود.
دستور scanf بالا را مي توان به صورت زير هم نوشت:

```
scanf(“%d”,&x);
scanf(“%d”,&y);
```

مي توان بيش از دو مقدار هم با دستور scanf دريافت كرد.

یک مشکل در گرفتن کاراکتر با scanf

- برنامه ای بنویسید که دو کاراکتر بگیرد و سپس آنها را در کنار هم چاپ کند.

ظاهرا این برنامه باید به صورت زیر باشد:

```
#include <stdio.h>
void main()
{
   char c1,c2;
   scanf("%c%c",&c1,&c2);
   printf("%c%c",c1,c2);
}
```

اما هنگام اجرای برنامه ملاحظه می کنیم که تنها یکی از کاراکترهای وارد شده چاپ می شود.

بافر ورودی

- **علت رخ دادن این مشکل:**
  - داده هایی که در هنگام اجرای scanf وارد می کنیم در داخل محلی از حافظه که بافر دستور scanf خوانده می شود ذخیره می شود. دستور printf بسته به آنچه برایش مشخص شده است از این بافر عنصر بر می دارد و روی مانیتور چاپ می کند.
  - در این مثال هنگام اجرای scanf ما ابتدا یک کاراکتر (مثلا a) ، سپس فاصله یا ENTER و بعد کاراکتربعد (مثلا b) را وارد کرده ایم. با توجه به اینکه فاصله یا ENTER در واقع خود یک کاراکتر هستند در واقع ما سه کاراکتر وارد کرده ایم که هر ۳ در بافر scanf قرار می گیرند.
  - هنگام اجرای printf دو تا کاراکتر اول یعنی a و فاصله (یا ENTER) چاپ می شود و کاراکتر مورد نظر (b) چاپ نمی شود.
  - برای رفع مشکل می توان نوشت :

```
scanf("%c%c%c",&c1,&c2,&c3);
```

- به طور کلی استفاده از scanf برای گرفتن کاراکتر مناسب نیست. دو دستور که مخصوص گرفتن کاراکتر هستند و مشکل بالا را ندارند getche و getch می باشند.

دستور getche

- تنها براي دريافت كاراكتر از keyboard به كار مي رود.
- تفاوت آن با scanf اين است كه در اينجا نيازي به زدن Enter بعد از وارد كردن مقدار نيست.
- نياز به header file با نام conio.h دارد.

```
#include <conio.h>
#include <stdio.h>
void main()
{
char ch;
ch=getche();
printf(”\n%c",ch);
}
```

دستور getch

- كاملا مشابه getche مي باشد. تنها تفاوت آن اين است كه ورودي را نمي بينيم.

فرمت بندي خروجي

```
#include <stdio.h>
Void main()
{
    float num=3.4;
    printf(“%f”,num);
}
```

- روي مانيتور ، 3.400000 چاپ خواهد شد.
- %f به طور پيش فرض عدد را با 6 رقم اعشار نشان مي دهد.
- با فرمت بندي خروجي مي توانيم مشخص كنيم اعداد چگونه نمايش داده شوند.

فرمت بندي خروجي (ادامه)

- در فرمت بندي خروجي دو چيز مشخص مي شود:
  - تعداد ارقام اعشار
  - تعداد مكان هاي در نظر گرفته شده براي نمايش عدد (طول ميدان)

مثالها:

```
float x=2.3;
printf("%.3f",x);
```

مقدار چاپ شده : 2.300

```
float x=2.3;
printf("%.7f",x);
```

مقدار چاپ شده : 2.3000000

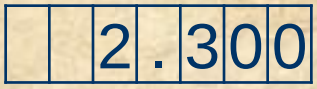
رقم بعد از نقطه تعداد ارقام اعشاري را مشخص مي كند.

تعیین طول میدان

```
int x=23;
printf(“%5d”,x);
```


```
int x=23;
printf(“%-5d”,x);
```


```
float x=2.3;
printf(“%7.3f”,x);
```



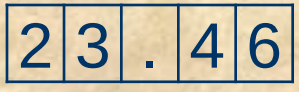

```
float x=2.3;
printf(“%-7.3f”,x);
```

چند مثال

```
int x=123456;
printf(“%3d”,x);
```

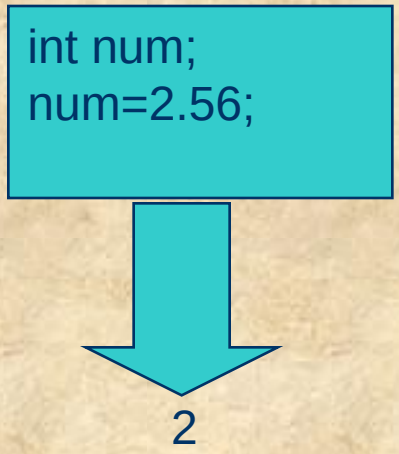

```
float  x=23.456;
printf(“%.2f”,x);
```


```
float  x=23.456;
printf(“%2.2f”,x);
```



تبديل انواع

- زماني بروز مي كند كه مي خواهيم عددي از يک نوع را در متغيري از نوع ديگر بريزيم.

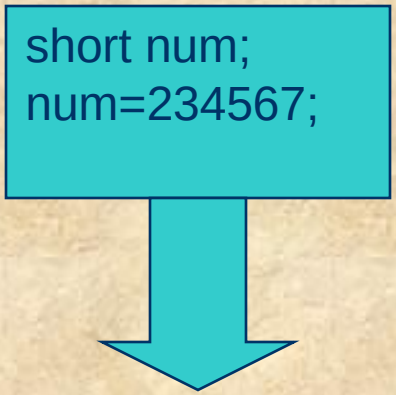


قانون كلي

مقدار اعشاري=متغير از نوع صحيح (int، long، short، char)

بخش اعشاري حذف مي شود و بخش صحيح در متغير قرار مي گيرد.

تخصیص مقدار خارج از محدوده عدد



short عددي در محدوده

بين 32768- تا 32767

استفاده از كاراكتركنترلي نامناسب

```
float x=23.5;
printf(“%d”,x);
```

مقدار نامرتبط

```
int x=23;
printf(“%f”,x);
```

صفر

در استفاده از كاراكترهاي كنترلي **%** در **printf** بايد دقت كرد كه متناظر با نوع متغير باشد.

مثال هاي بالا نمونه هايي از خطاهاي زمان اجرا مي باشد.

اپراتورهاي ++ و --

X= X+1

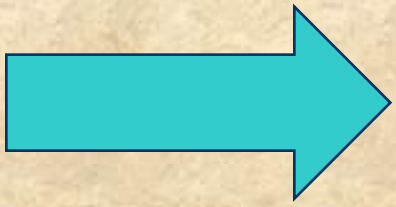
≡ X++ يا ++X

X=X-1

≡ X-- يا --X

تفاوت j++ و ++j یا j-- و --j

```
#include <stdio.h>
void main()
{
   int i=0;
   printf("%d",++i);
}
```



خروجي: 1

=+ و =-

X=X+23;

≡ X+=23;

X=X-23;

≡ X-=23;

X*=z;

X/=Y;

X%=z;

X=X+Y;

≡ X+=Y;

X=X-Y;

≡ X-=Y;

تقدم (اولویت) عملگرها

X=2+4*5%2;

?

مثال

2+4%2=2+0=2

(2+4)%2=(6)%2=0

4+6%3/2*4=4+0/2*4

=4+0*4=4+0=4

i=0;

j=3;

++i+4/2+ -- j=1+4/2+-- j

=1+4/2+2=1+2+2=3+2=5

()
++ --
/ * %
+ -
> >= < <=
== !=
= += -= *= /= %=

در یک عبارت ،تقدم اپراتورهايي که در یک سطح اولویت قرار دارند از چپ به راست مي باشد.

خلاصه مطالب گذشته

- آشنایی با دستور خروجی printf برای چاپ متن یا مقدار متغیرها
- دستورات ورودی: scanf ، getche و getch
- فرمت بندی خروجی: تعیین تعداد ارقام اعشار و طول میدان
- اپراتورها، انواع آنها و اولویت

چند مثال برنامه نویسي

مثال ۱: برنامه ای بنویسید که زاویه بر حسب درجه را دریافت کرده و معادل رادیان آن را روی مانیتور نشان دهد.

1. **قبل از نوشتن یک برنامه قدم اول پیدا کردن راه حل مسئله است.**
   - **در این جا باید رابطه ای بین درجه و رادیان پیدا کرد.**

$$\frac{d}{180} = \frac{R}{\pi} \Rightarrow R = \frac{\pi}{180} d$$

۲. سپس آنچه در برنامه از ما خواسته شده را به صورت مجموعه ای از مراحل می نویسیم.

- **گرفتن درجه: d**
- **تبدیل درجه به رادیان با استفاده از رابطه بالا: به دست آمدن R**
- **چاپ R**

- **به بیان بالا اصطلاحا یک الگوریتم گفته می شود. الگوریتم مجموعه ای از مراحل است که دنبال کردن آنها ما را به حل مسئله می رساند.**
  - **نوشتن الگوریتم مستقل از یک زبان برنامه نویسی خاص است.**

۳. در مرحله آخر با نوشتن دستورات مناسب الگوریتم را به برنامه واقعی تبدیل می کنیم.

ادامه چند مثال برنامه نویسي

- **گرفتن d:**
  - scanf(“%f”,&d);
- **تبدیل d به R:**
  - R=(3.14/180)*d;
- **چاپ R:**
  - printf(“%f”,R);

ادامه چند مثال برنامه نويسي

```
#include <stdio.h>
void main()
{
    float d,R;
    printf("Enter a angle in degree:");
    scanf("%f",&d);
    R=(3.14/180)*d;
    printf("It is %.2f  radian",R);
}
```

ادامه چند مثال برنامه نویسي

- **مثال ۲) برنامه ای بنویسید که مقدار $\sin\frac{\pi}{2}$ را با استفاده از ۳ جمله اول سری $\sin(x)=\sum_{n=0}^{\infty}(-1)^n\frac{x^{(2n+1)}}{(2n+1)!}$ محاسبه کند. همچنین خطای حاصل را هم حساب کند.**

$$\sin(x)\approx\sum_{n=0}^{3}(-1)^n\frac{x^{(2n+1)}}{(2n+1)!}=x-\frac{x^3}{3!}+\frac{x^5}{5!}=A(x)$$

$$error=\sin\frac{\pi}{2}-A(\frac{\pi}{2})$$

- **الگوریتم:**
  - **محاسبه** $A(\frac{\pi}{2})$
  - **محاسبه خطا**
  - **چاپ دو مقدار محاسبه شده**

ادامه چند مثال برنامه نويسي

```
#include <stdio.h>
void main()
{
        float x=3.141592/2;
        float A=x-(x*x*x)/(1*2*3)+(x*x*x*x*x)/(1*2*3*4*5);
        float E=1-A;
        printf("The sin(pi/2) by use of 3 terms=%f",A);
        printf("\nThe error=%f",E);
}
```

ادامه چند مثال برنامه نویسي

- **مثال ۳) برنامه ای بنویسید که عددی را بگیرد اگر عدد بر ۲ بخش پذیر بود ۰ و درغیر این صورت ۱ چاپ کند.**

- **روش حل مسئله: اگر عدد** a **بر دو بخش پذیر باشد باقیمانده تقسیم آن بر دو صفر می شود و اگر بخش پذیر نباشد یک .**
- **الگوریتم:**
  - a **را بگیر**
  - a%2 **را چاپ کن**

ادامه چند مثال برنامه نويسي

```
#include <stdio.h>
void main()
{
    int a;
    printf("Enter a number");
    printf("\nThe program will print 0 if it is dividable by 2");
    printf("\nand 1 if not:");
    scanf("%d",&a);
    printf("%d",a%2);
}
```

دستورات شرطی

- **دستورات شرطی**(Conditional statement) **:**
  - **مقدمه: تعریف عبارت** (expression) **در زبان** C **و انواع عبارات**
  - **دستور** if
  - **دستور** if-else
  - **دستور** switch-case

مقدمه – expression (عبارت)

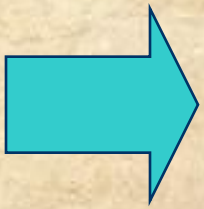
- **اپراتورها:**
  - **محاسباتی:** + – * / % ++ – –
  - **مقایسه ای:** < > == =< >= =! !=
  - **منطقی:** && (و) ||(یا) ! (نقیض)
- Expression **(عبارت) : ترکیبی است از مقادیر، متغیرها و اپراتورها.**
- **انواع عبارتها:**
  - **محاسباتی : از اپراتورهای محاسباتی استفاده می شود:** 3+5 3*4+3 x*2+y
  - **منطقی: عبارتی است که می توان به آن ارزش درست یا نادرست نسبت داد. مانند**
    x<=0 3!=4 (x!=3 && z>=3) x==3 4==y !(3==4)
  - **معمولا این گونه عبارات با استفاده از اپراتورهای مقایسه ای و منطقی ساخته می شوند.**

مقدار (ارزش) عبارات

- **هر عبارت (محاسباتی یا مقایسه ای) دارای مقدار (ارزش) است:**
  - **مقدار (ارزش) عبارت** 3+4 **برابر** 7 **است.**
- **در زبان** c **به عبارت مقایسه ای درست ارزش** 1 **و به عبارت مقایسه ای نادرست ارزش** 0 **نسبت داده می شود.**
  - **ارزش** 4=<3 ، 0 **است.**
  - **ارزش** (3>2 || 5=<4) ، 1 **است.**

مثال

```
#include <stdio.h>
Void main()
{
    printf(“%d”,(3<= 4));
}
```



خروجی عدد ۱ خواهد بود.
خروجی عدد ۰ خواهد بود.

دستورات شرطی

در بسیاری اوقات می خواهیم یک دستور تنها زمانی اجرا شود که شرط خاصی برقرار باشد.

مثال : برنامه ای بنویسید که عددی را از ورودی دریافت کرده و قدر مطلق آن را چاپ کند.

$$y = |x| = \begin{cases} x & x \geq 0 \\ -x & x \leq 0 \end{cases}$$

```
#include <stdio.h>
void main()
{
float x;
scanf("%f",&x);
??
}
```

دستورات شرطي در C

- **سه نوع دستور شرطی در C وجود دارد:** if، if-else و switch-case

• عبارت منطقی حتما باید در داخل پرانتز باشد.

•در حالتی که بخش های if یا else بیش از یک دستور باشند باید بین { و } قرار گیرند.

```
if (عبارت منطقی)
   یک یا چند دستور
else
   یک یا چند دستور

if (x<0)
  y=-x;
else
  y=x;

if (x<0)
{
  y=-x;
  z=y+1;
}
else
   y=x;
```

```
if (عبارت منطقی)
   یک یا چند دستور

if (x<0)
   y=-x;

if (x<0)
{
   y=-x;
   z=y+1;
}
```

مثال ۱

محاسبه قدر مطلق:

```
#include <stdio.h>
void main( )
{
float x;
printf(“Enter the number:”);
scanf(“%f”,&x);
if (x<0)
  printf(“%f”,-x);
else
  printf(“%f”,x);
}
```

مثال ۲

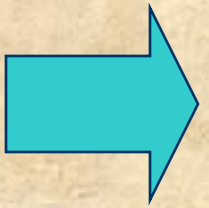
- مثال ۲) برنامه ای برای محاسبه sign(x):

```
include <stdio.h>
void main()
{
float x;
int y;
printf("Enter a number:");
scanf("%f",&x);
if (x>0)
{
        y=1;
        printf("its sign=%d",y);
}
else
 if (x<0)
{
        y=-1;
        printf("its sign=%d",y);
}

else
printf("its sign undefined");

}
```

$$sign(x)=\begin{cases}1 & x>0\\ -1 & x<0\\ undefine & x==0\end{cases}$$



در داخل بخش if یاelse می تواند دستور if یا if-else دیگری قرار گیرد. (nested if—تودرتو if-else)

نکته: یک if یا if-else به همراه مجموعه دستورات درونی شان در مجموع یک دستور فرض می شوند . به همین علت در برنامه روبرو برای else اول از {} استفاده نشده است.

Nested-if

- Nested if معمولا موجب ناخوانایی برنامه و دشواری درک آن می شود. در درک نحوه عملکرد برنامه استفاده از قانون زیر می تواند راه گشا باشد:
  - هر else به نزدیکترین if قبل از خود وابسته است. مشروط بر آنکه :

    در داخل } و { محصور نشده باشد

مثال ۱

- **خروجی قطعه برنامه زیر را به دست آورید.**

```
x=-3;
if (x>=0)
if (x>=12)
printf(“X is greater than or equal 12”);
else
printf(“X is less than 12 and greater than 0”);
else
printf(“X is less than 0”);
```

خروجی x is less than 0

مثال ۲

```
x=-5;
if (x>=0)
{
if (x>=12)
printf(“x is greater or equal than 12”);
}
else
printf(“x is less than 0”);
```

عبارت x is less than 0 **چاپ خواهد شد.**

دستور switch-case

- **فرم کلی این دستور به صورت زیر است:**

```
switch (متغیر از نوع صحیح  یا کاراکتر)
{
case   عدد صحیح یا کاراکتر:
     دستور(ات)
      break;
case   عدد صحیح یا کاراکتر:
     دستور(ات)
      break;
.....
……
……..
 default:
     دستور(ات)
}
```

- **متغیر یا مقدار صحیح با مقادیر موجود در** case **ها مقایسه می شود. اگر با مقدار یکی از** case **ها مطابقت داشت دستورات داخل آن اجرا می شود. در صورت عدم تطابق با هیچ یک از** case **ها دستورات بخش** default **اجرا می شود.**
- **آوردن بخش** default **اختیاری است .**
- **نیازی به قرار دادن** {} **ما بین دستورات** case **ها نیست.**
- **نوشتن** break **در انتهای دستورات هر** case **الزامی است.**

مثال ۱

```
x=1;
switch(x)
{
case 1:
      printf(“*”);
      break;
case 2:
      printf(“**”);
      break;
case 3:
       printf(“***”);
      break;
default:
       printf(“Error”);
}
```

دستور break **موجب خروج از** switch **می شود.**
اگر در انتهای case **ای** break **نوشته نشود بدنه** case **بعدی اجرا می شود.**

مثال ۲

```
char x=‘b;
switch(x)
{
case ‘a’:
      printf(“%d”,’a’);
     break;
case ‘b’:
      printf(“%d”,’b’);
      break;
}
```

- **نکته مهم: دستور switch فقط برای مقادیر صحیح یا کاراکتر به کار می رود.**

نکته در مورد break

- **نکته: دستور** break **موجب خروج از دستور** switch **می شود.** break **بر دستورات** if **و** if-else **تأثیری ندارد و موجب خروج از آنها نمی شود.**
- **مثال:**

در بعد از ظهر ۱۵ ژانویه ۱۹۹۰ سرویس های تلفنی شرکت AT&T **آمریکا (بزرگترین شرکت مخابراتی آمریکا) برای اولین بار در ۱۱۴ سال دچار اختلالی گسترده شد که ۹ ساعت به طول انجامید.**

- مهندسان دریافتند که منبع اشکال در بخشی از نرم افزار نوشته شده برای کنترل خطوط است:
- نویسندگان نرم افزار اشتباها تصور کرده بودند که دستور break در if (y==1) موجب خروج از بخش شرط if می شود. در حالی که موجب خروج از switch می شود . در نتیجه بخشی که به صورت _____ نشان داده شده اجرا نمی شود و موجب بروز مشکل شده بود.

```
switch (line)
{
   case 1:
        ........
        break;
   case 2:
        if (x == 0)
     {
            ...........
            ...........
            if (y == 1)
              break;
            ............
            ............
        }
        _________
        _________
        break;
   default:
        .........
       ........
 }
 ........
.........
...........
```

دستورات تکرار (حلقه ها)

- با استفاده از دستورات تکرار (حلقه ها) می توانیم اجرای بخشی از برنامه را چند بار تکرار کنیم.

- سه دستور تکرار در زبان C
  - for
  - while
  - do-while

ضرورت وجود حلقه ها در برنامه

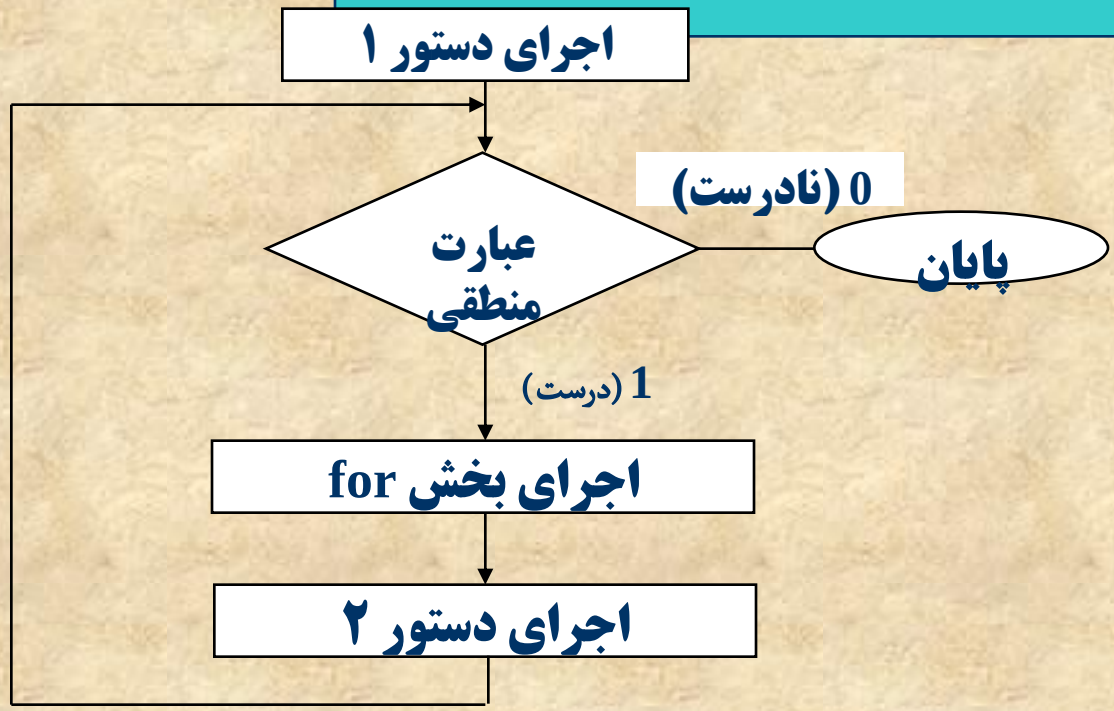
- **مثال ۱) برنامه ای بنویسید که اعداد ۱ تا ۱۰۰ را چاپ کند.**

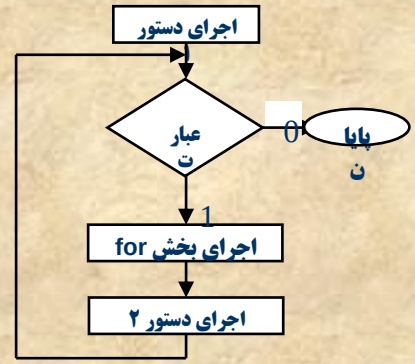
```
#include <stdio.h>
void main()
{
  int i;
   i=0;
   i++;
   printf(“%d”,i);
   i++;
   printf(“%d”,i);
   .....
   .....
   ...
}
```

دستور for

for (دستور ۱ ;عبارت منطقی ;دستور ۲)
دستور(ات)

بخش for: بخشی که اجرای آن تکرار خواهد شد.
برای بیش از یک دستور در بخش for استفاده از { } لازم است.





مثال

- مثال ۲) عملکرد قطعه برنامه زیر را توضیح دهید:

```
for (i=0;i<2;i++)
    printf("\n%d" ,i);
```

1. i=0
2. چون i<2 است دستور printf اجراء می شود و عدد 0 در مانیتور چاپ می شود.
3. i++ اجرا می شود. در نتیجه i=1
4. i=1<2 در نتیجه مجددا دستور printf اجراء می شود و این بار عدد 1 در مانیتور چاپ می شود.
5. i++ اجرا می شود. در نتیجه i=2
6. این بار i=2<2 برقرار نیست. در نتیجه از حلقه خارج می شویم.

دستور printf دو بار اجرا شد.

- مثال ۳) مثال ۱ با استفاده از حلقه ها

```
#include <stdio.h>
void main()
{
    int i=0;
    for (i=1;i<=100;++i)
        printf("\n%d",i);
}
```

چند مثال دیگر

```
for (i=5;i<=8;i++)
        printf("\n%d" ,i);
```

```
for (x=5;x>8;x++)
        printf("\n%d" ,x);
```

حلقه اصلا اجرا نمی شود

```
for (k=8;k>5;k--)
        printf("\n%d" ,k);
```

```
for (num=3;num>0;num++)
    printf("\n%d" ,num);
```

حلقه پایان ناپذیر (بی نهایت)

```
for (i=8;i>1;i/=2)
    printf("\n%d" ,i);
```

مثال

- عملکرد قطعه برنامه زیر:

```
scanf(“%d”,&i);
if (i>0)
    for(j=0;j<i;++j)
        printf(“\n%d”,j);
```

مشابه if ، for و دستورات داخل حلقه مجموعا یک دستور فرض می شوند و در نتیجه نیازی به { } برای if نیست.

مثال

- برنامه ای بنویسید که ۱۰۰ عدد را از کاربر بگیرد و حاصل جمع آنها را چاپ کند.

```
#include <stdio.h>
void main()
{
   int i;
   float sum;
   float x;
   sum=0;
   for (i=0;i<100; i++)
   {
     scanf("%f", &x);
     sum+=x;
   }
   printf("The result=%f",sum);
}
```

حلقه های تودرتو(nested for)

- **عملکرد برنامه زیر را توضیح دهید.**

```
for (i=0;i<3;++i)
    for (j=0;j<2;++j)
        printf(“*”);
```

حلقه ای که در درون آن حلقه ای دیگر قرار گرفته است.

for اول ۳ بار اجرا می شود و در هر بار اجرای آن for دوم، دو بار اجرا می شود. در نتیجه ۶ علامت ستاره در خروجی چاپ خواهد شد.

مثال

- عملکرد قطعه برنامه زیر:

```
for (i=0;i<3;++i)
for (j=0;j<2;++j)
printf(“*”);
printf(“!”);
```

شش * و تنها یک علامت ! چاپ خواهد شد.

- عملکرد قطعه برنامه زیر:

```
for (i=0;i<3;++i)
    for (j=0;j<i;++j)
        printf(“*”);
```

مثال

- عملکرد برنامه زیر:

```
#include <stdio.h>
void main()
{
int i,j;
for (i=1;i<=10;++i)
{
    printf("\n");
    for (j=1;j<=10;++j)
        printf("%d\t",(i*j));
}
}
```

چاپ جدول ضرب ۱۰*۱۰

```
1       2       3       4       5       6       7       8       9       10
2       4       6       8       10      12      14      16      18      20
3       6       9       12      15      18      21      24      27      30
4       8       12      16      20      24      28      32      36      40
5       10      15      20      25      30      35      40      45      50
6       12      18      24      30      36      42      48      54      60
7       14      21      28      35      42      49      56      63      70
8       16      24      32      40      48      56      64      72      80
9       18      27      36      45      54      63      72      81      90
10      20      30      40      50      60      70      80      90      100
Press any key to continue
```

نکاتی در مورد دستور for

نکته ۱: در بخش های دستور ۱ و دستور ۲ هر دستوری می تواند باشد.
مثال

```
#include <stdio.h>
#include <conio.h>
void main()
{
	char ch;
	float x,sum=0;
	for (ch='y';ch=='y';ch=getche())
	{
		printf("\nEnter a number:");
		scanf("%f",&x);
		sum+=x;
		printf("To continue press y: ");
	}
	printf("\nThe sum=%f",sum);
}
```

برنامه تا زمانی که کاربر y را فشار می دهد عدد می خواند و زمانی که کاربر کلیدی غیر از y را فشار دهد مجموع اعداد گرفته شده را چاپ می کند.

نکاتی در مورد دستور for(ادامه)

نکته ۲) هر یک از بخش های دستور ۱ ، دستور ۲ یا عبارت منطقی را می توان حذف کرد

مثال ۱

```
#include <stdio.h>
void main()
{
   int i=3;
   for (;i<7;++i)
  {
      printf("%d\t",i);
  }
}
```

مثال ۲

```
#include <stdio.h>
void main()
{
   int i;
   for (i=3;i<7; )
  {
      printf("%d\t",i);
      i++;
  }
}
```

مثال ۳

```
#include <stdio.h>
void main()
{
   int i;
   for (i=3;;i++ )
  {
      printf("%d\t",i);
}
}
```

نکته ۳) در بخش های دستور ۱ و ۲ می توان بیش از یک دستور قرار داد. دستورات با استفاده از کاما (,) از هم جدا می شوند.

مثال:

```
#include <stdio.h>
void main()
{
    int m,n;
    for (m=1,n=8; m<n;m++,n--)
          printf("m=%d , n=%d\n",m,n);
}
```

دستور while

while (عبارت مقایسه ای)
دستور(ات)

- عملکرد: اگر عبارت مقایسه ای درست باشد دستور(ات) اجرا می شود. اگر عبارت مقایسه ای نادرست باشد از حلقه خارج می شویم.
- استفاده از {} در صورت وجود بیش از یک دستور در حلقه

مثال

```
i=0;
while (i<4)
{
    printf(“%d”,i);
    i++;
}
```

عبارت 0123 روی مانیتور نشان داده می شود

مثال

```
#include <stdio.h>
void main()
{
   char ch;
   int num,sum;
   ch='y';
   sum=0;
   while (ch=='y')
   {
      printf("\nEnter a number: ");
      scanf("%d",&num);
      sum+=num;
      printf("\ncontinue(enter y/n):");
      scanf("\n%c",&ch);
   }
   printf("The result= %d",sum);
}
```

تا زمانی که کاربر y را انتخاب کند اعداد گرفته می شود
با انتخاب هر کلیدی غیر از y از حلقه خارج می شویم
و حاصل جمع مقادیر چاپ می شود.

نکته–تفاوت مهم for و while

- با استفاده از دستور while می توان حلقه هایی نوشت که در آنها تعداد تکرار از قبل مشخص نیست. برخلاف for که معمولا در آن تعداد تکرار از قبل مشخص است.

دستور do-while

do

دستور(ات)

while (**عبارت مقایسه ای**);

عملکردی کاملا مشابه while **دارد. با این تفاوت که در** while **عبارت مقایسه ای در ابتدا و قبل از اجرای دستورات چک می شود، اما در** do-while **این کار در انتها و بعد از اجرای دستورات انجام می شود.**

بخش حلقه لااقل یک بار اجرا خواهد شد.

مثال

```
#include <stdio.h>
void main()
{
   char ch;
   int num,sum;
   sum=0;
   do
   {
      printf("\nEnter a number: ");
      scanf("%d",&num);
      sum+=num;
      printf("\ncontinue(enter y/n):");
      scanf("\n%c",&ch);
    }
   while (ch=='y');
  printf("The result= %d",sum);
}
```

دستور break

- موجب خروج از حلقه می شود.
- مثال :

```
#include <stdio.h>
void main()
{
    int t;
for (t=0;t<100;t++)
{
        printf(“%d”,t);
        if (t==5)
                break;
}
}
```

- چند مثال برنامه نویسی
- توابع : یکی از مهم ترین مباحث زبان C

چند مثال برنامه نویسی

- **برنامه ای بنویسید که ضرایب یک معادله درجه ۲ را بگیرد و ریشه های آن را محاسبه کند.**

$$ax^2 + bx + c = 0$$

$$\Delta = b^2 - 4ac$$

$$\begin{cases} x_{1,2} = \dfrac{-b \pm \sqrt{\Delta}}{2a} & \Delta > 0 \\ x_1 = \dfrac{-b}{2a} & \Delta = 0 \\ no \quad root & \Delta < 0 \end{cases}$$

1. **حل مسئله:**
2. **الگوریتم:**
   1. **گرفتن ضرایب** a، b و c
   2. **محاسبه دلتا**
   3. **محاسبه ریشه ها بر اساس مقدار دلتا**

نکات:

- **دستور sqrt برای محاسبه ریشه دوم به کار می رود.**
- **header file آن ، math.h می باشد.**

```
#include <stdio.h>
#include <math.h>
void main()
{
    float a,b,c,x1,x2,delta;
    printf("Enter the coefficients");
    printf("\na=");
    scanf("%f",&a);
    printf("b=");
    scanf("%f",&b);
    printf("c=");
    scanf("%f",&c);
    delta=b*b-4*a*c;
    if (delta>0)
    {
        x1=(-b+sqrt(delta))/(2*a);
        x2=(-b-sqrt(delta))/(2*a);
        printf("x1=%.2f\tx2=%.2f",x1,x2);
    }
    else
        if (delta==0)
        {
            x1=-b/(2*a);
            printf("x1=%.2f",x1);
        }
        else
            printf("No root");
}
```

ادامه چند مثال برنامه نویسي

- **مثال ۲) تبدیل مختصات دکارتی (x,y) به مختصات قطبی** (r,θ) $\theta \in [-\pi, \pi)$
  - **محاسبه r به سهولت قابل انجام است:** $r = \sqrt{x^2 + y^2}$
  - **برای محاسبه وضعیت های مختلف زیر را خواهیم داشت:**
    - **(y>=0 و x>0) یا (y<0 و x>0)** $\theta = \arctan(\frac{y}{x})$
    - **(y>0 و x<0)** $\theta = \pi + \arctan(\frac{y}{x})$
    - **(y<=0 و x<0)** $\theta = -\pi + \arctan(\frac{y}{x})$
    - **(y>0 و x=0):** $\theta = \frac{\pi}{2}$
    - **(y<0 و x=0):** $\theta = -\frac{\pi}{2}$

الگوریتم:

1. **گرفتن (x,y)**
2. **محاسبه r**
3. **محاسبه θ با توجه به محل نقطه**

```
#include <stdio.h>
#include <math.h>
void main()
{
        float x,y,r,teta;
        printf("Enter (x,y)\n");
        printf("x=");
        scanf("%f",&x);
        printf("y=");
        scanf("%f",&y);
        r=sqrt(x*x+y*y);
        if ((x>0 && y>=0) || (x>0 && y<0))
                teta=atan(y/x);
        if (x<0 && y>0)
                teta=atan(y/x)+3.141592;
        if (x<0 && y<=0)
                teta=atan(y/x)-3.141592;
        if (x==0 && y>0)
                teta=3.141592/2;
        if (x==0 && y<0)
                teta=-3.141592/2;
        printf("polar cordinate:\n");
        printf("r=%.2f",r);
        printf("\nteta=%.2f radian\n",teta);
}
```

نکته: atan ، براي محاسبه arc tan به کار مي رود. header file آن math.h است.

ادامه چند مثال برنامه نويسي

- **برنامه ای بنویسید که دو عدد را بگیرد و ب م م آنها را محاسبه کند.**

ب م م:Greatest Common Devisor (GCD)

ادامه چند مثال برنامه نويسي

1. **روش حل: محاسبه ب م م با روش پلکانی:**

	8	1	2	
52	6	4	2	0
48	4	4		

2. **الگوریتم :**

الف) گرفتن عدد اول (a) **و عدد دوم** (b) **. فرض کنید** a>b

ب) تا زمانی که a%b=0 **نشده است.**

c=a%b

a=b **و بعد** b=c

ج) b **را چاپ کن**

ادامه چند مثال برنامه نويسي

```
#include <stdio.h>
void main()
{
    int a,b,c;
    printf("Enter 2 number. at first enter the larger\n");
    printf("Number 1:");
    scanf("%d",&a);
    printf("Number 2:");
    scanf("%d",&b);
    while (a%b !=0)
    {
      c=a%b;
      a=b;
      b=c;
    }
    printf("The GCD=%d",b);
}
```

ادامه چند مثال برنامه نویسي

ساخت یک ساعت کامپیوتری (h:m:s)

الگوریتم:

1. s=0، m=0، h=0
2. **نمایش زمان**
3. s=s+1
4. **اگر** s=60 **آنگاه** s=0 **و** m=m+1
5. **اگر** m=60 **آنگاه** m=0 **و** h=h+1
6. **اگر** h=24 **آنگاه** h=0
7. **یک ثانیه انتظار**
8. **انجام ۲ تا ۷ به تعداد بی نهایت**

```
#include <stdio.h>
#include <windows.h>
void main()
{
   int hour=0,minute=0,second=0;
          int i;
          for (;;)
          {
              printf("%d:%d:%d   ",hour,minute,second);
              second++;
              if (second==60)
              {
                  second=0; minute++;
              }
              if (minute==60)
             {
                  minute=0; hour++;
              }
              if (hour==24)
                  hour=0;
              for (i=0;i<11;++i)
                  printf("\b");
             Sleep(1000);
          }
}
```

نکات:

1. for(;;) یک حلقه پایان ناپذیر است.
2. کاراکتر کنترلی \b موجب می شود که مکان نما ۱ واحد به عقب برود. استفاده از آن موجب می شود تا زمان بعدی روی قبلی چاپ شود و نه در محل مجزا.

کد اسکی \b ، ۸ است. (کد اسکی \n و \t به ترتیب

۳. تابع Sleep برای ایجاد تاخیر به کار می رود . (S حرف بزرگ است).

Sleep(n) به اندازه n میلی ثانیه تاخیر ایجاد می کند.

header file آن windows.h است.

(در borland و Turbo ، dos.h می باشد).

ادامه چند مثال برنامه نويسي

- **افزودن قابلیت زنگ زدن در یک زمان مشخص**

نکته : کاراکتر کنترلی \a **یک صدای بوق تولید می کند.**

کافی است دستوراتی برای گرفتن زمان بوق زدن اضافه شود و همچنین یک دستور شرطی که رسیدن زمان را چک کند.

```
#include <stdio.h>
#include <windows.h>
void main()
{
   int hour=0,minute=0,second=0;
  int i;
  int h,m,s;
  printf("Set time to ring\n");
  printf("Hour=");
  scanf("%d",&h);
  printf("Minute=");
  scanf("%d",&m);
  printf("Second=");
 scanf("%d",&s);
 for (;;)
 {
  printf("%d:%d:%d   ",hour,minute,second);
    if (hour==h && minute==m && second==s)
      printf("\a\a");
     second++;
 if (second==60)
 {
     second=0; minute++;
 }
    if (minute==60)
   {
       minute=0; hour++;
    }
    if (hour==23)
       hour=0;
    for (i=0;i<11;++i)
        printf("\b");
   Sleep(1000);
 }
}
```

ادامه چند مثال برنامه نويسي

- **بازی حدس: توسط کامپیوتر عدد صحیح تصادفی بین ۰ تا ۱۰۰۰تولید می شود (لکن به کاربر نشان داده نمی شود) سپس از کاربر خواسته می شود عددی را به عنوان حدس وارد کند . اگر عدد وارد شده مساوی عدد مطلوب باشد بازی با برنده شدن فرد خاتمه می یابد و اگر کوچکتر یا بزرگتر باشد به کاربر اعلام می شود. کاربر می تواند تا حداکثر ۱۳ حدس داشته باشد. در صورتی که تمام حدس ها اشتباه باشد کاربر بازنده می شود**

ادامه چند مثال برنامه نويسي

- **الگوریتم:**
1. **تولید یک عدد بین ۰ تا ۱۰۰۰**
2. **دریافت حدس کاربر**
3. **اگر عدد کاربر برابر عدد مطلوب بود اعلام موفقیت**
4. **و گرنه اعلام کوچکتر یا بزرگتر بودن به کاربر و سپس بازگشت به ۲**
5. **انجام ۲ تا ۴ به تعداد ۱۳ بار**

rand() اعداد تصادفی بین ۰ تا ۳۲۷۶۷ تولید می کند. در نتیجه rand()%1001 اعداد تصادفی بین ۰ تا ۱۰۰۰ تولید می کند.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{ int x,y,i;
 x=rand()%1001;
 for(i=0;i<13;i++)
{
   printf("\n Guess %dth number:",(i+1));
   scanf("%d",&y);   if (x==y)
  {
    printf("\nyou win");
    break;
   }
  if(x>y)
     printf("\nthe number is greater than you entered");
 if(x<y)
    printf("\nthe number is less than you entered");
 }
if(i==13)
   printf("you loss");
}
```

تابع

- با پاسخ به ۳ سؤال زیر می توان با این مفهوم آشنا شد:

1. تابع چیست؟
2. چگونه ساخته می شود و در کجای برنامه قرار می گیرد؟
3. چگونه استفاده می شود؟

در این جلسه به این سؤال ها پاسخ داده می شود.

۱) تابع چیست؟؟

- **مقدمه– تابع در ریاضیات**

$$y = f(x) = x^2$$
$$x \in R \quad y \in R$$

- **می توان گفت در هر تابع در ریاضیات عناصر زیر وجود دارد:**
  - **نام: (در مثال بالا f)**
  - **ورودی (در مثال بالا x) که متعلق به مجموعه ای به نام دامنه است.**
  - **مقدار برگشتی (y) که متعلق به مجموعه ای به نام برد است.**
  - **ضابطه تابع (x^2) که ارتباط بین مقدار برگشتی و ورودی را مشخص می کند.**

تابع مقدار x را می گیرد و بر اساس ضابطه موجود، y را محاسبه کرده و برمی گرداند.

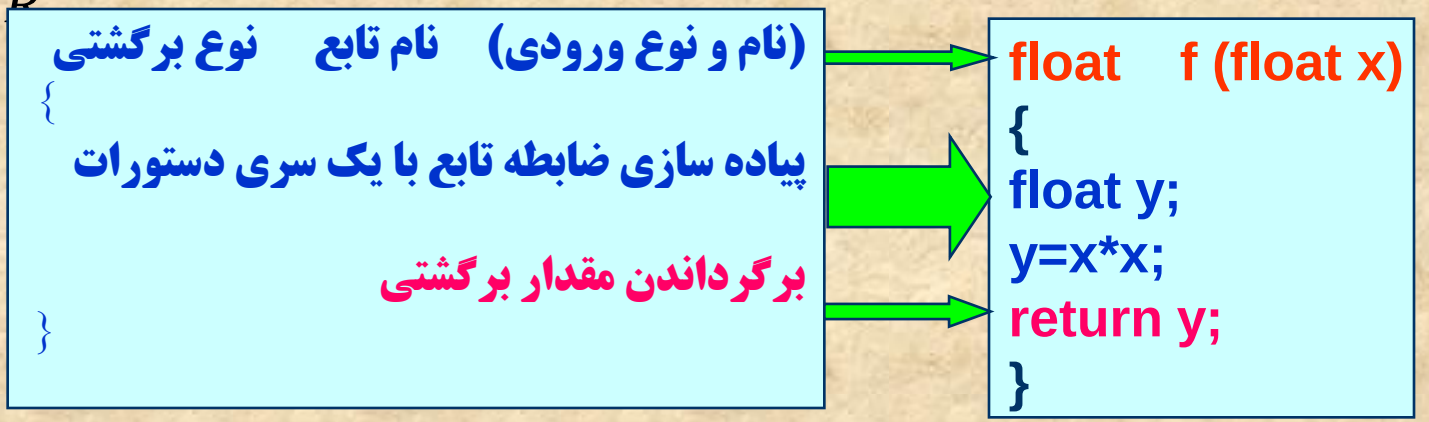
تابع چیست؟؟ (ادامه)

- مفهوم تابع در زبان C مشابهت زیادی با مفهوم تابع در ریاضیات دارد و در واقع مدل کننده تابع در ریاضیات است.

۲) ساخت تابع

$$y = f(x) = x^2$$

$$x \in R \quad y \in R$$



دستور return مقدار محاسبه شده را برمی گرداند.

در مثال بالا تابعی نوشتیم که یک ورودی اعشاری را می گیرد و مجذور آن را برمی گرداند.

به x ورودی یا پارامتر ورودی می گویند.

y هم مقدار برگشتی می باشد.

(نام و نوع ورودی) نام تابع نوع برگشتی
{
پیاده سازی ضابطه تابع با یک سری دستورات
برگرداندن مقدار برگشتی
}

ساخت تابع (ادامه)

- تابعی بنویسید که یک ورودی اعشاری بگیرد و قدر مطلق آن را محاسبه کند.

```
float  func (float num)
{
   float result;
   if (num < 0)
       result=-num;
  else
       result=num;
  return result;
}
```

ساخت تابع (ادامه)

- فرم دیگری از تابع مثال قبل

```
float  func (float num)
{
    if (num < 0)
       return –num;
    else
       return num;
}
```

ساخت تابع (ادامه)

- **تابع** sign(x)

$$sign(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$$

محل قرار گرفتن تابع

```
#include …..
#include ….
……
void main()
{
…..
…..
}
```

```
float    f (float x)
{
float y;
y=x*x;
return y;
}
```

قبل از main **و بعد از دستورات** include **قرار می گیرد.**

۳) استفاده از تابع

- **استفاده از تابع در ریاضیات در واقع همان احضار تابع است.**

$y=f(x)=3x+1$

احضار تابع: $f(2)=7$

عدد 2 **به جای** x **قرار می گیرد و** $y=7$ **به دست می آید.**

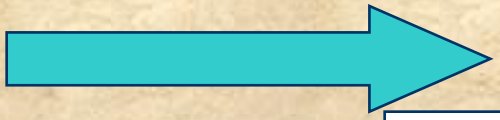
در زبان C **هم می توان تابع را احضار کرد (مثال بعد)**

استفاده از تابع

- مثالی از احضار تابع

```
#include <stdio.h>
float   f (float x)
{
float y;
y=x*x;
return y;
}
 void main()
{
float z;
z=f(3.5);
printf("%f",z);
}
```

در این مثال هنگام اجرای برنامه زمانی که به دستور احضار تابع می رسیم تابع f به ازای ورودی x=3.5 محاسبه خواهد شد.
مقدار برگشتی تابع (یعنی y) در z ریخته می شود



دستور احضار تابع

به 3.5 آرگومان ورودی گفته می شود

- مثال قبل یک برنامه کامل و قابل اجرای C است.
- در این مثال :
  - تابعی به نام f نوشته ایم که مجذور یک عدد را محاسبه می کند.
  - در main این تابع را احضار کرده ایم.

- مثال عملکرد تابع زیر را توضیح دهید.

```
float  average (int n)
{
   int i;
   float x,sum=0;
   for (i=0;i<n;++i)
   {
      scanf(“%f”,&x);
      sum+=x;
   }
return (sum/n);
}
```

تابع بالا n عدد مي گيرد و ميانگين آن را برمي گرداند

مثال

```
#include <stdio.h>
float  average (int n)
{
   int i;
   float x,sum=0;
   for (i=0;i<n;++i)
  {
     scanf("%f",&x);
     sum+=x;
   }
```

ادامه برنامه

```
return (sum/n);
}
void main()
{
printf("%f",average(3));
}
```

برنامه میانگین سه عدد را چاپ می کند

مثال

```
#include <stdio.h>
float  average (int n)
{
   int i;
   float x,sum=0;
   for (i=0;i<n;++i)
  {
     scanf(“%f”,&x);
     sum+=x;
   }
```

ادامه برنامه

```
return (sum/n);
}
void main()
{
int p;
scanf(“%d”,p)
printf(“%f”,average(p));
}
```

مشابه مثال قبل . با این تفاوت که در اینجا تعداد مقادیر از کاربر گرفته می شود.

خلاصه مطالب قبل

- آشنایی با مفهوم تابع
- طرز ساخت تابع
- چند مثال

مثال –تابع با بیش از یک ورودی

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

تابع ماکزیمم دو عدد را برمی گرداند.
این تابع دارای دو ورودی می باشد(مشابه تابع دو متغیره در ریاضیات)

ادامه برنامه

```
#include <stdio.h>
int max (int x, int y)
{
    if x>=y
       return x;
    else
       return y;
}
```

```
void main()
{
   int z,s;
   scanf(“%d%d”,&z,&s);
   printf(“%d”,max(z,s));
}
```

نکته: تعداد و نوع آرگومانها باید با تعداد و نوع پارامترها یکسان باشد.

تعریف چند تابع در برنامه

```
#include <stdio.h>
long fact(int n)
{
int i;
long result=1;
for (i=1;i<=n;++i)
  result*=i;
return result;
}
```

ادامه (۱)

```
float pow(float x, int n)
{
int j;
float ret_val=1;
for (j=0;j<n;++j)
  ret_val=ret_val*x;
return ret_val;
}
```

ادامه (۲)

```
float exp(float x, int m)
{
  int k;
float res=1;
for (k=1;k<=m;++k)
res+=pow(x,k)/fact(k);
return res;
}
void main()
{
  printf("%f",exp(2,10));
}
```

توضیح برنامه

- در این برنامه از سه تابع fact، pow و exp استفاده شده است. این نشان می دهد می توان در یک برنامه از چند تابع استفاده کرد.

توضیح برنامه را با بررسی عملکرد این سه تابع آغاز می کنیم.

تابع fact

- این تابع عدد n را می گیرد و !n را حساب می کند.
- نکته: به ازای n های منفی خطایی گرفته نمی شود اما نتیجه غلط 1 برگشت داده می شود.

```
long fact(int n)
{
int i;
long result=1;
for (i=1;i<=n;++i)
  result*=i;
return result;
}
```

تابع pow

```
float pow(float x, int n)
{
int j;
float ret_val=1;
for (j=0;j<n;++j)
   ret_val=ret_val*x;
return ret_val;
}
```

تابع pow عدد حقیقی x و عدد صحیح n را می گیرد و x^n را محاسبه می کند.

تابع exp

```
float exp(float x, int m)
{
   int k;
float res=1;
for (k=1;k<=m;++k)
res+=pow(x,k)/fact(k);
return res;
}
```

$$e^x = \frac{x0}{0!} + \frac{x1}{1!} + \frac{x2}{2!} + \frac{x3}{3!} + \ldots + \frac{x^m}{m!}$$

تابع exp ، عدد حقیقی x و عدد صحیح m را می گیرد و رابطه بالا را محاسبه می کند.

عملکرد برنامه

- **عملکرد برنامه:**
  1. **اجرای برنامه از main شروع می شود.**
  2. **وجود exp(2,10) موجب می شود تابع exp شروع به اجرا کند.**
  3. **در تابع exp توابع pow و fact با پارامترهای مختلف احضار می شوند و محاسبات خود را انجام می دهند. مثلا به ازای k=1 در تابع exp ، fact(1) و pow(2,1) احضار می شوند.**
  4. **نتیجه احضار تابع exp(2,10) محاسبه عبارت زیر است:**

$$e^x = \frac{x0}{0!} + \frac{x1}{1!} + \frac{x2}{2!} + \frac{x3}{3!} + \ldots + \frac{x^{10}}{10!}$$

تقسیم بندی توابع

- می توان از جنبه های مختلف توابع را تقسیم بندی کرد:
  - ورودی: با ورودی –بی ورودی
  - مقدار برگشتی: با برگشتی– بی برگشتی
  - نوشته شده توسط کاربر–کتابخانه ای

تقسیم بندی تابع از نظر ورودی

- یک تابع می تواند دارای ورودی باشد(مانند مثالهایی که تاکنون گذشت) و می تواند بدون ورودی باشد (مثال صفحه بعد)

```
#include <stdio.h>
#include <conio.h>
int check_password ( )    یا   int check_password (void)
{
   char c1, c2,c3;
   c1=getche();
   c2=getche();
   c3=getche();
   if (c1=='A' && c2=='B'  && c3=='C')
         return 1;
   else  return 0;
}
void main()
{
  int s=check_password();
  if (s==1)
    printf("Correct password");
  else
    printf("Incorrect password");
}
```

- برای تابع بی ورودی، ( ) گذاشته می شود. در داخل پرانتز یا چیزی نمی نویسیم و یا کلمه void نوشته می شود.
- هنگام احضار تابع نیز باید () بگذاریم.

تقسیم بندی از نظر مقدار برگشتی

یک تابع می تواند یک مقدار برگرداند(مانند مثالهایی که تا کنون گذشت) یا هیچ مقداری را برنگرداند (مثال زیر)

تابع بدون برگشتی return **ندارد.**
در بخش نوع برگشتی void **نوشته می شود.**

```
#include <stdio.h>
void  ave  (int n)
{
      float x,sum=0;
      int i;
      for (i=0;i<n;++i)
      {
            scanf("%f",&x);
            sum+=x;
      }
      printf("%f",sum/n);
}
void main ( )
{
   ave(3);
}
```

مثال ۲ –تابع بی ورودی –بی برگشتی

```
#include <stdio.h>
void header ( )  یا void header(void)
{
   printf(“This program is written by Rouhollah Dianat”);
   printf(“\nAs a part of computer programming course”);
   printf(“\nfall 2006”);
}
void main()
{
  header();
  .....
  ......
}
```

مثال ۳) تابع بی ورودی – بی برگشتی

تابع بی ورودی بی برگشتی بنویسید که میانگین چند عدد را محاسبه کند.

```
#include <stdio.h>
void ave ( )
{
   int i,n;
   float x,sum=0;
   printf("\nEnter a number:");
   scanf("%d",&n);
   for (i=0;i<n;++i)
   {
            scanf("%f",&x);
            sum+=x;
   }
   printf("%f",sum/n);

}
```

```
void main ()
{
    ave( );
}
```

main تابع است

- نکته :

با نگاهی به بخش main درمی یابیم که در واقع main خود یک تابع است. تابعی بی ورودی بی برگشتی

به بیان دقیق تر:

یک برنامه در زبان C از چند تابع تشکیل می شود. نام یکی از این توابع باید main باشد. خاصیت این تابع این است که اجرای برنامه از آن شروع می شود.

- **نکته : می توان** main **را به صورت تابعی با مقدار برگشتی تعریف کرد:**

```
int main( )
{
.....
return 0;
}
```

مقدار برگشتی main **به سیستم عامل برگشت داده می شود.**

- **نکته: تابع** main **حتی می تواند ورودی داشته باشد.(تحقیق)**

- **نکته : اگر تابع** main **در برنامه وجود نداشته باشد:**
  - **یک خطای** link **اعلام می شود. در واقع یکی از وظایف** linker **پیوند دادن اجزای مختلف برنامه برای رسیدن به یک برنامه قابل اجرا می باشد. زمانی که** main **وجود نداشته باشد** linker **نمی تواند محل شروع اجرای برنامه را تشخیص دهد و در نتیجه خطا اعلام می کند.**

توابع نوشته شده توسط کاربر –توابع کتابخانه ای

- توابع نوشته شده توسط کاربر همان توابعی است که تاکنون نوشته ایم.

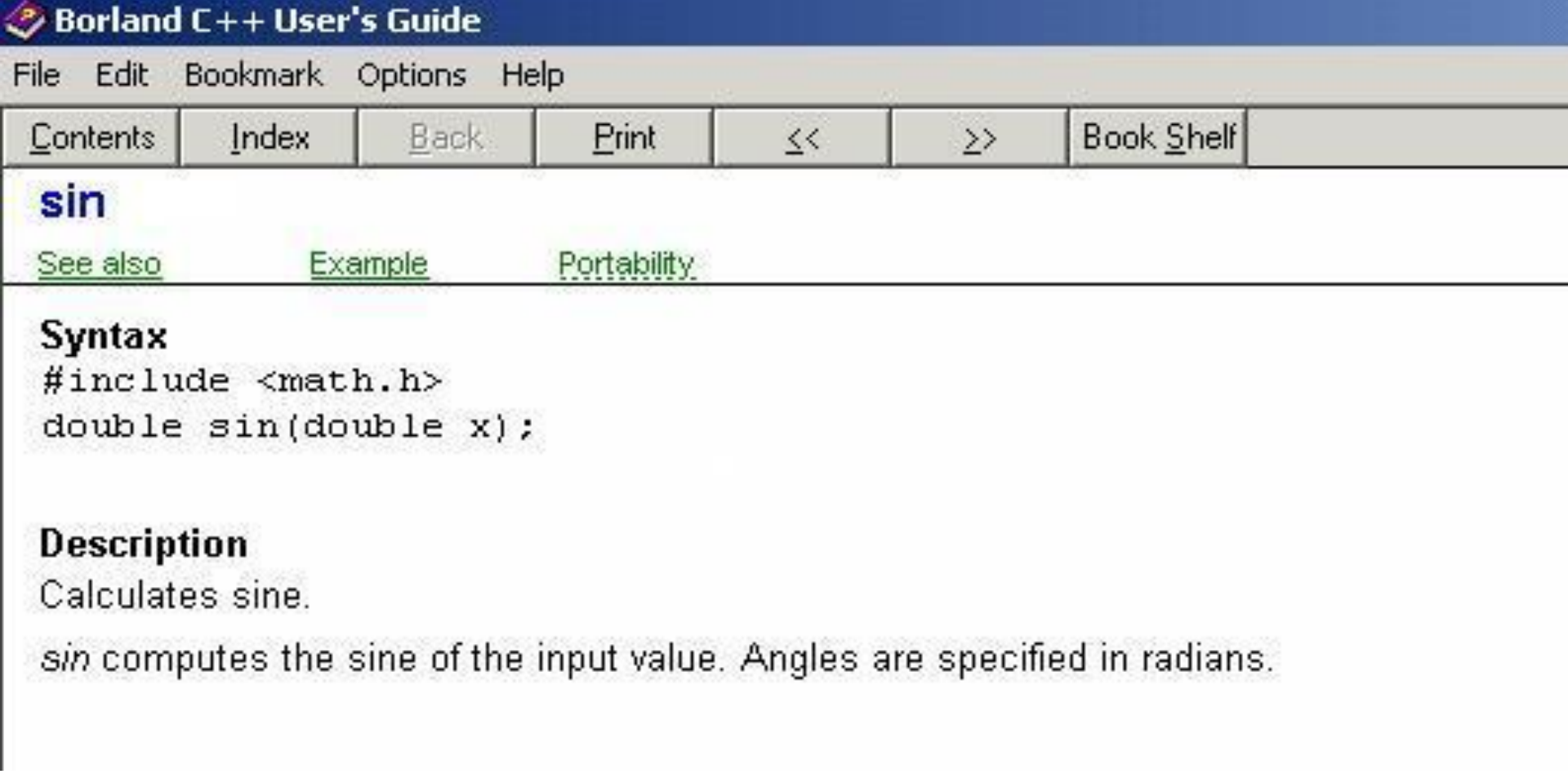
- توابع کتابخانه ای: در زبان C برای سهولت کار برنامه نویس در زبان C توابع زیادی از قبل تهیه شده اند و برنامه نویس می تواند از آنها استفاده کند.
- جزئیات این توابع در فایل های header متعدد قرار گرفته است.

توابع کتابخانه ای

- **انواع توابع کتابخانه ای:**
  - **توابع ورودی–خروجی: printf، scanf، getche،getch و...**
    - **header file: اغلب stdio.h ، conio.h**
  - **توابع ریاضی: abs (برای محاسبه قدر مطلق) ،pow (برای محاسبه توان) ، sqrt (محاسبه رادیکال) sin، cos، tan، sinh، cosh،tanh، asin، acos، atan، exp (محاسبه e^x) ، log (لگاریتم در مبنای e) ، log10 (log در مبنای ۱۰)**
    math.h :header file
  - **توابع برای کار با فایل ها: fread، fwrite، fopen و ...(در بخش آشنایی با فایل ها توضیح داده خواهند شد).**
    stdlib.h :header file
  - **کارهای گرافیکی، کاربا پورت ها، نوشتن برنامه اسمبلی و.....**

توابع کتابخانه ای

نکته: در کتاب های برنامه نویسی C و همچنین help محیط های نرم افزاری در مورد ورودی و مقدار برگشتی توابع و نوع آنها و همچنین عملکرد آنها توضیح داده شده است.



در شکل help محیط borland c برای تابع sin نشان داده شده است.
ملاحظه می شود که نوع مقدار برگشتی double است. تابع یک ورودی از نوع double دارد و برای محاسبه sin به کار می رود.
header file این تابع هم math.h است.

ادامه توابع کتابخانه ای

- **مثال:**

```
#include <stdio.h>
#include <math.h>
void main ()
{
printf("%f",sin(3.141));
}
```

مثال هایی از کاربرد توابع

مثال ۱

- **می خواهیم برنامه ای بنویسیم که مشتق تابع دلخواه f(x) را در نقطه x_0 محاسبه کند.**

روش حل:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

به ازای h کوچک می توان نوشت:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}$$

اکنون برنامه ای می نویسیم که عبارت بالا را محاسبه کند (صفحه بعد)

محاسبه مشتق x^2+1 در x=1

```
#include <stdio.h>
#include <math.h>
float f(float x)
{
    return (pow(x,2)+1);
}

float f_prime(float x0)
{
    float res;
    float h=0.0001;
    res=(f(x0+h)-f(x0))/h;
    return res;
}
```

```
void main()
{
  printf("%f",f_prime(1));
}
```

مشتق دوم

$$f''(x_0) = \lim_{h \to 0} \frac{f'(x_0 + h) - f'(x_0)}{h}$$

$$\approx \frac{f'(x_0 + h) - f'(x_0)}{h}$$

محاسبه مشتق دوم تابع xsinx+1 در نقطه x=3

```
#include <stdio.h>
#include <math.h>
float f(float x)
{
      return (x*sin(x)+1);
}

float f_prime(float x0)
{
      float res;
      float h=0.0001;
      res=(f(x0+h)-f(x0))/h;
      return res;
}
```

```
float f_second(float x0)
{
    float res;
    float h=0.0001;
    res=(f_prime(x0+h)-f_prime(x0))/h;
    return res;
}

void main()
{
   printf("%f",f_second(3));
}
```

نکته : روش های تحلیلی و عددی

- معمولا محاسبه مشتق ، با استفاده از تعریف مشتق و یک سری قضایا انجام می شود. به عبارت دیگر مشتق با استفاده از استدلال و اثبات محاسبه می شود. به این گونه روش ها روش های حل تحلیلی گفته می شود.

- در روشی که در این برنامه استفاده شد مشتق را با جایگذاری مقدار و انجام یک سری محاسبات روی آنها به دست آوردیم. اصطلاحا به این روش ها، روش های حل عددی گفته می شود.

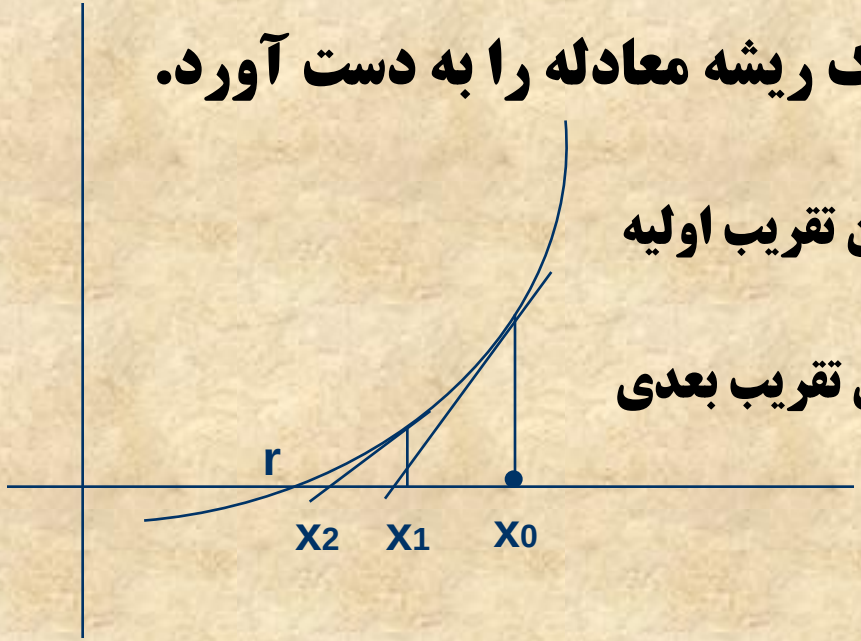
- روش های عددی به آسانی قابل پیاده سازی توسط کامپیوتر هستند.

مثال ۲) به دست آوردن ریشه معادله

- تابع f(x) مفروض است. می خواهیم ریشه های معادله f(x)=0 را تعیین کنیم.

- روش های تحلیلی محاسبه ریشه تنها برای دسته محدودی از توابع ارائه شده اند.
- در مقابل روش های عددی قادرند ریشه معادله را تا هر دقتی محاسبه کنند.
- یکی از این روش ها نیوتن–رافسون است که در این مثال به پیاده سازی این روش می پردازیم.

روش نیوتن–رافسون



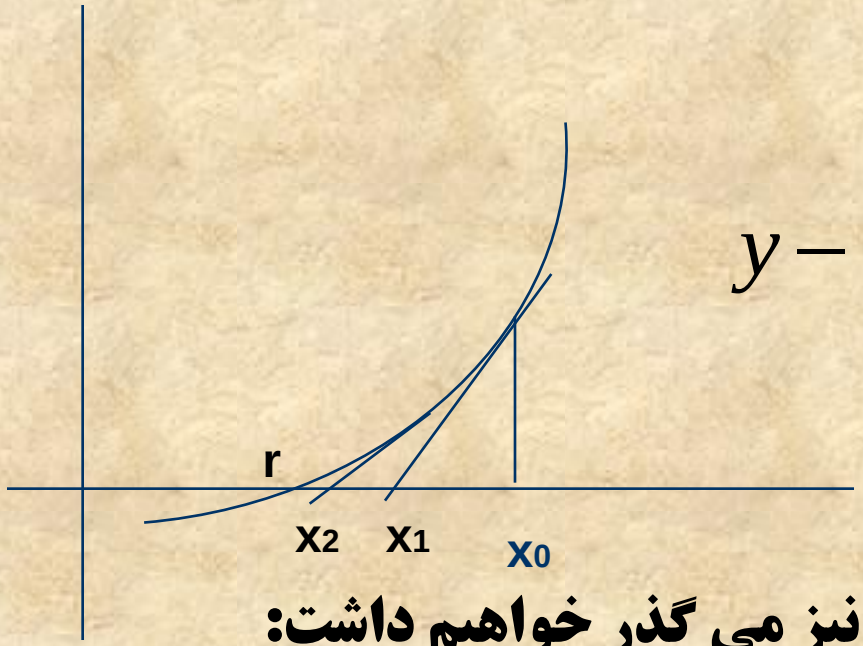
- با استفاده از این روش می توان یک ریشه معادله را به دست آورد.
  مراحل روش به این صورت است:
  - انتخاب یک نقطه دلخواه x0 به عنوان تقریب اولیه
  - رسم خط مماس بر نقطه (x0,f(x0))
  - x1 نقطه تقاطع خط و محور به عنوان تقریب بعدی
  - با رسم خط مماس بر (x1,f(x1))

  به تقریب x2 می رسیم

  .....

  می توان ثابت کرد در صورتی تابع f شرایط خاصی را ارضا کند این تقریب ها به سمت ریشه همگرا می شود.

ادامه



معادله خط مماس بر (x0,f(x0))**:**

$$y - f(x_0) = f'(x_0)(x - x_0)$$

با توجه به اینکه این خط از نقطه (x1,0) **نیز می گذر خواهیم داشت:**

$$0 - f(x_0) = f'(x_0)(x_1 - x_0)$$

$$\Rightarrow x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

به همین ترتیب برای x2 خواهیم داشت:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

و در حالت کلی:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

با افزایش n به تقریب بهتری دست می یابیم.

ادامه

- برنامه ای بنویسید که ریشه معادله f(x)=xsinx-1 را تا ۳ رقم اعشار محاسبه کند.

(نکته : دقت ۳ رقم اعشار: $|f(x_n)| \le 0.5*10^{-3}$

```
#include <stdio.h>
#include <math.h>
float f(float x)
{
float y=x*sin(x)-1;
return y;
}
float f_prime(float x0)
{
float res;
float h=0.001;
res=(f(x0+h)-f(x0))/h;
return res;
}
```

```
float root()
{
  float x0=1,x1;
  float d=fabs(x0);
  while (fabs(f(x0))>=0.5e-3)
  {
     x1=x0-f(x0)/f_prime(x0);
     x0=x1;
  }
  return x0;}
void main()
{
   printf("%f",root());}
```

نکات مثال قبل

- fabs: **محاسبه قدر مطلق اعداد حقیقی**

تابع abs **برای محاسبه قدر مطلق اعداد صحیح به کار می رود.**

- 0.5e-3 **اصطلاحا نماد علمی نمایش اعداد خوانده می شود و برابر است با :** $0.5*10^{-3}$

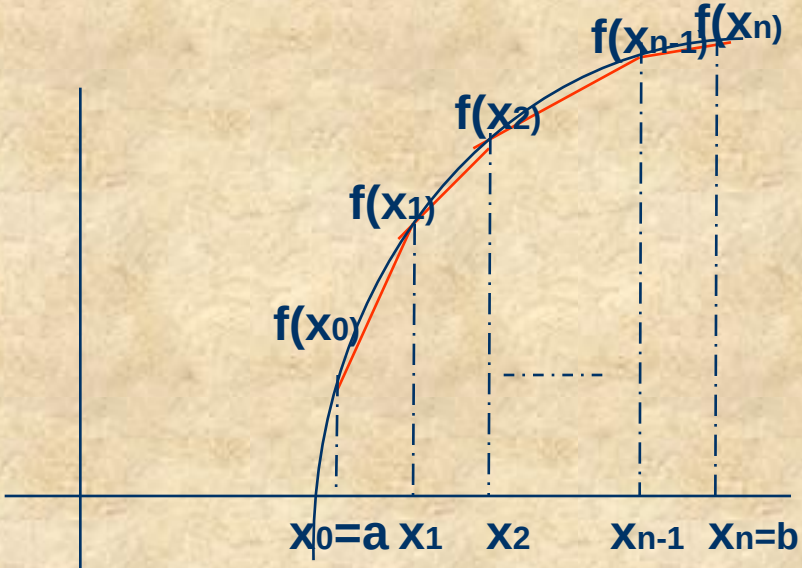
مثال ۳–محاسبه انتگرال

برنامه ای بنویسد که حاصل $\int_a^b f(x)dx$ **را محاسبه کند.**

بیان یک روش عددی:

بازه [a,b] **را به نقاط متساوی الفاصله** xi **تقسیم می کنیم.**

مجموعه مساحت ذوزنقه های تشکیل شده در شکل مقابل می تواند تقریبی از انتگرال بالا باشد:



$$I \approx \frac{h}{2}(f(x_0)+f(x_1))+\frac{h}{2}(f(x_1)+f(x_2))+\ldots\frac{h}{2}(f(x_{n-2})+f(x_{n-1}))+\frac{h}{2}(f(x_{n-1})+f(x_n))=$$

$$\frac{h}{2}(f(x_0)+2f(x_1)+2f(x_2)+\ldots+2f(x_{n-1})+f(x_n))$$

$$\int_{1}^{3}(x\sin x+1)dx$$

محاسبه

```
#include <stdio.h>
#include <math.h>
float f (float x)
{
  return  x*sinx+1;
}

float integral(float a, float b,int n)
{
     int i;
     float p=a;
     float sum=f(p);
     float h=(b-a)/n;
```

```
     for (i=1;i<n;++i)
     {
          p=p+h;
          sum=sum+2*f(p);
     }
     sum=sum+f(b);
     return(h/2*sum);
}

void main()
{
 printf("%f",integral(1,3,10));
}
```

نکته: متغیرهای محلی و سراسری

برنامه زیر را در نظر بگیرید.

```
#include <stdio.h>
void func (int z)
{
    int y;
    y=z*z;
}
void main ()
{
int y;
y=5;
func (6);
printf("%d",y);
}
```

سوال: مقدار چاپ شده بر روی مانیتور چقدر است:

عدد ۵ یعنی مقدار y تعریف شده درتابع main چاپ می شود نه ۳۶ که مقدار y تعریف شده در func است.

ادامه

- مثال ۲) برنامه زیر را در نظر بگیرید:

```
#include <stdio.h>
 void func (int z)
{      int y;
     y=z*z;
}

void main ()
{func(3);
printf("%d",y);
}
```

این برنامه مواجه با خطای کامپایلی زیر خواهد شد:

y: undeclared identifier

یعنی y تعریف نشده است. به عبارت دیگر main متغیری به نام y را نمی شناسد.

ادامه

با یاد گرفتن بحث متغیرهای محلی و سراسری می توان به سؤالاتی از این قبیل پاسخ داد.

متغیرهای محلی و سراسری

- در زبان C دو دسته متغیر وجود دارد.
  - محلی (local) .
  - عمومی–سراسری (global).

متغیرهای local

- **متغیرهای local متغیرهایی هستند که در داخل توابع تعریف شده اند.**
- **این متغیرها دارای ویژگی های زیر هستند.**
  - **تنها در داخل تابعی که تعریف شده اند معتبرند.**
  - **هنگام اجرای تابع ایجاد می شوند و هنگام خاتمه اجرا از بین می روند.**
- **همه متغیرهایی که تاکنون در مثال ها دیده ایم متغیرهای local هستند.**

مثال

```
#include <stdio.h>
void func (int z)
{
int y;
y=z*z;
}
void main ()
{
func(3);
printf("%d",y);
}
```

در تابع func ، y محلی است. در نتیجه تنها زمانی که func در حال اجراست وجود دارد. و وقتی که اجرای func تمام می شود از بین می رود.
به همین علت دستور printf(“%d”,y) اشتباه است چون y ای وجود ندارد که printf آن را چاپ کند.

ادامه

در این مثال مقداری که چاپ می شود مقدار متغیر Y تعریف شده در main است و نه y تعریف شده در تابع func. چون y تابع func فقط در محدوده خود func شناخته شده است و برای main شناخته شده نیست.

```
#include <stdio.h>
void func (int z)
{
    int y;
    y=z*z;
}
void main ()
{
int y;
y=5;
func (6);
printf("%d",y);
}
```

متغیرهای سراسری (global)

تعریف: به متغیرهایی که خارج از توابع تعریف می شوند متغیرهای سراسری یا global می گویند.

```
#include <stdio.h>
int num=4;
int  func1  (int x)
{
      num+=x;
     return num;
}
int  func2  (int x)
{
      num-=x;
       return num;
}
void main ( )
{
   printf("%d", func1(3));
   printf("\n%d", func2(3));
}
```

دو ویژگی متغیرهای global:

- **برخلاف متغیرهای local، از نقطه ای که تعریف می شوند تا انتهای برنامه معتبرند.**
- **اگر هنگام نعریف مقداری به آنها داده نشود مقدارشان صفر در نظر گرفته می شوند.**

در برنامه روبرو، متغیر num global است. چون در خارج توابع تعریف شده. این متغیر در func1، func2 و main اعتبار دارد.

عملکرد برنامه:

- **ابتدا num=4 می شود.**
- **func1(3) اجرا می شود. مقدار num برابر 7 شده و چاپ می شود.**
- **سپس func2(3) اجرا می شود. مقدار num برابر 4 شده و چاپ می شود.**
- **اگر به جای int num=4، تنها int num نوشته می شد، مقدار num صفر می شد.**

تابع بازگشتی (Recursive function)

- تابعی است که خودش را احضار کند.
- در همه مثال هایی که تاکنون داشتیم احضار یک تابع توسط تابعی دیگر صورت می گیرد. در زبان C این امکان وجود دارد که یک تابع توسط خودش احضار شود(مثال بعد).

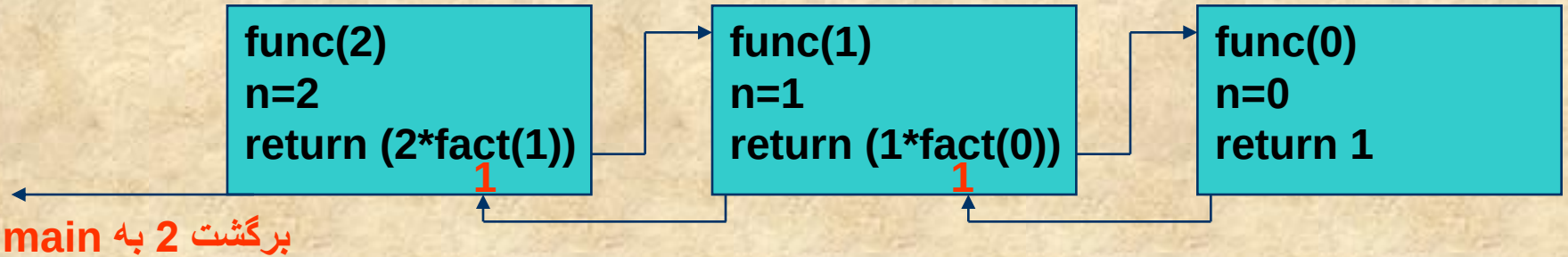
مثال ۱

```
#include <stdio.h>
long fact(int n)
{
  if (n==0)
     return 1;
  else
      return (n*fact(n-1));
}
void main()
{
   printf("%d",fact(2));
}
```

تابع func، یک تابع بازگشتی است.
چون خودش را احضار کرده است.

توضیح عملکرد برنامه

```
#include <stdio.h>
long fact(int n)
{
   if (n==0)
       return 1;
   else
         return (n*fact(n-1));
}
void main()
{
      printf("%d",fact(2));
}
```



نکته

- **نکته :ایده ای که ما را به برنامه بازگشتی رساند فرمول زیر بود:**

$$n! = n * (n-1)!$$

یا:

fact(n)=n*fact(n-1)

فرمول بالا به روشنی مفهوم بازگشت را درخود دارد. (برای محاسبه fact **به ازای** n **باید همان تابع** fact **به ازای** n-1 **محاسبه شود).**

نکته

- **در هر تابع بازگشتی دو رکن اصلی زیر وجود دارد:**
  - **قانون (فرمول) بازگشت: چگونگی بازگشت را مشخص می کند. در مثال قبل قانون بازگشت** n!=n*(n-1)! **بود.**
  - **شرط خاتمه: مشخص می کند بازگشت چه زمانی متوقف می شود. در مثال قبل** n=0 **شرط خاتمه بود.**

$$fact(n) = \begin{cases} n * fact(n-1) & n > 0 \\ 1 & n = 0 \end{cases}$$

- **برای بتوانیم یک برنامه بازگشتی بنویسیم باید ابتدا این دو رکن را به دست آوریم. بعد از انجام این کار نوشتن برنامه های بازگشتی، به سادگی امکان پذیر است.**

مثال ۲

- **برنامه ای بنویسید که x^n را به صورت بازگشتی محاسبه کند.**
n عدد صحیح غیر منفی و x اعشاری است.

فرض کنید می خواهیم تابعی به نام pow بدین منظور بنویسیم :
ابتدا دو رکن بازگشت را مشخص می کنیم:

- **قانون بازگشت:** $x^n = x * x^{n-1}$
- **شرط خاتمه :** n=0

$$pow(x,n) = \begin{cases} x * pow(x, n-1) & n > 0 \\ 1 & n = 0 \end{cases}$$

برنامه مثال ۲

```
#include <stdio.h>
float pow(float x, int n)
{
    if (n==0)
        return 1;
    else
        return (x*pow(x,n-1));
}

void main()
{
    printf("%f",pow(2.5,2));
}
```

مثال ۳

- **برنامه بازگشتی بنویسید که حاصل عبارت زیر را محاسبه کند:**

$$\sqrt{a+\sqrt{a+\sqrt{a+.....}}}$$

(تعداد n تا a داریم)

فرض کنید می خواهیم تابعی بازگشتی به نام rad(a,n) **بنویسیم که این کاررا انجام دهد.**

- **قانون بازگشت و شرط خاتمه :**

$$rad(a,n)=\begin{cases}\sqrt{a+rad(a,n-1)} & n>1\\ \sqrt{a} & n=1\end{cases}$$

برنامه مثال ۳

```
#include <stdio.h>
#include <math.h>
float rad(float a , int n)
{
    if (n==1)
        return sqrt(a);
    else
        return sqrt(a+rad(a,n-1));
}

void main()
{
    printf("%f",rad(2,3));
}
```

نکته: تابع sqrt از توابع کتابخانه ای است و header file آن math.h است. این تابع برای محاسبه رادیکال استفاده می شود.

مثال ۴

- **برنامه ای بازگشتی بنویسید که ب.م.م دو عدد را محاسبه کند.**

ب. م .م : Greatest Common Divisor (GCD)

می خواهیم تابعی به نام gcd(a,b) **بنویسیم که ب.م.م دو عدد** a **و** b **را محاسبه کند.**

روش معمول محاسبه ب.م.م، روش پلکانی است.

در این مثال:

	3	1	2	
33	9	6	3	0
27	6	6		

ب.م.م (3,0)=3 **ب.م.م** =(6,3) **ب.م.م** =(9,6) **ب.م.م** =(33,9)

مثال ۴–ادامه

- **با دقت در رابطه صفحه قبل می توان گفت:**

gcd(33,9)=gcd(9,33%9)

(%: باقی مانده)

به این ترتیب می توان قانون بازگشت و شرط خاتمه را به صورت زیر به دست آورد:

$$\gcd(a,b)=\begin{cases}\gcd(b,a\%b) & b\neq 0\\ a & b=0\end{cases}$$

برنامه مثال ۴

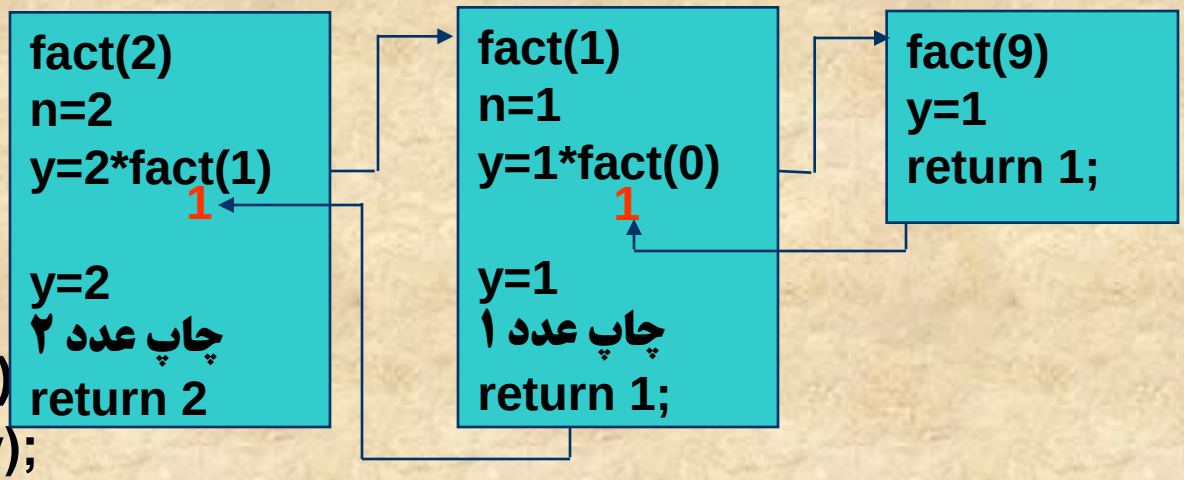
```
#include <stdio.h>
int gcd(int a , int b)
{
    if (b==0)
        return a;
    else
        return gcd(b,a%b);
}

void main()
{
    printf("%d",gcd(33,9));
}
```

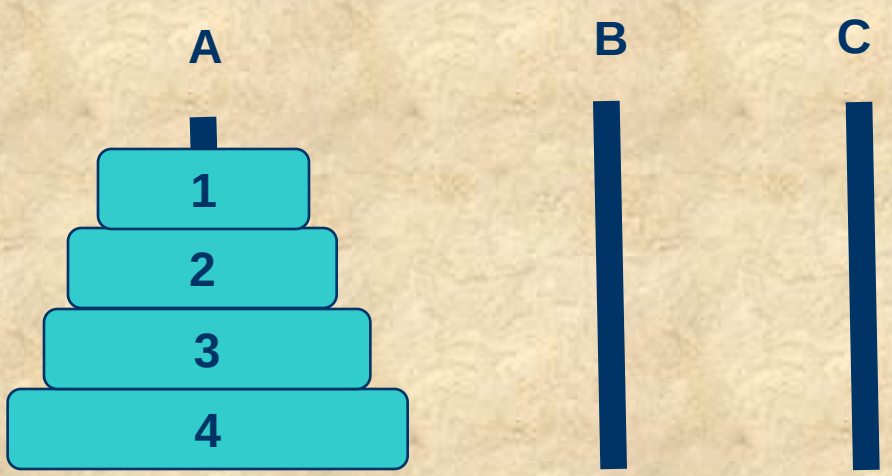
مثال ۵

عملکرد برنامه زیر را توضیح دهید.

```
#include <stdio.h>
long fact(int n)
{     long y;
     if (n==0)
        y=1;
   else   {
   y= n*fact(n-1);
   printf(“%d”,y);
   }
    return y; }
void main()
{printf("%d",fact(2));}
```



مثال ۶–برج های هانوی (Towers of hanoi)



بازی برج های هانوی:

سه میله A، B و C و تعدادی دیسک روی میله A داریم. می خواهیم این دیسک ها را از A و با کمک میله B به C انتقال دهیم . به نحوی که هیچ گاه دیسک بزگتر بر روی کوچکتر قرار نگیرد.

در اینجا می خواهیم برنامه بازگشتی بنویسیم که این بازی را پیاده کند.

برج های هانوی–ادامه

- هدف: جابجایی n دیسک از میله مبدأ (A) به میله مقصد (C) با کمک گرفتن از میله کمکی (B).
- می خواهیم برنامه ما یک سری پیغام به صورت زیر چاپ کند:
  - move disk from A to C
  - move disk from A to B
  - .....
- پیاده سازی بازی را با تابعی بازگشتی به نام move انجام دهیم:

```
move (int n,char source,char destination, char spare )
```

- دیسک ها را از ۱ تا n شماره گذاری می کنیم. (1 کوچکترین)

ادامه

- به دست آوردن قانون بازگشت:

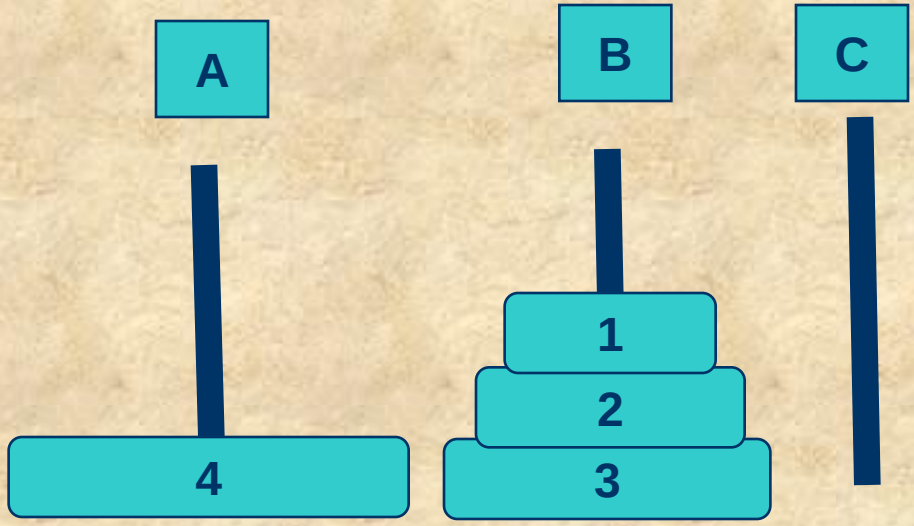
فرض کنید بازی را شروع کرده اید و بعد از انجام یک سری جابجایی به وضعیت زیر رسیده ایم:

یعنی n-1 دیسک را به B جابجا کرده باشیم.

با این فرض بقیه مراحل به صورت زیر است:

دیسک n ام را به C منتقل کن.

دیسک میله را از B به C جابجا کن.



قانون بازگشت و شرط خاتمه

در نتیجه:

- **جابجایی n دیسک از A به C با کمک B:**
  - **جابجایی n-1 دیسک از A به B با کمک C**
  - **انتقال دیسک n ام از A به C**
  - **جابجایی n-1 دیسک از B به C با کمک A**
- **به عبارت بهتر قانون بازگشت به صورت زیر**

- move (n,source,destination,spare)=
  - move(n-1,source,spare,destination)
  - move(1,source,destination,spare)
  - move(n-1,spare,destination,source)
- **شرط خاتمه: اگر n=1 باشد دیسک را از source به destination منتقل کن.**

برنامه برج های هانوی

```
#include <stdio.h>
void move(int n ,char source,char destination,char spare )
{
    if (n==1)
        printf("\nMove top disk from %c to %c",source,destination);
    else
    {
        move(n-1,source,spare,destination);
        printf("\nMove top disk from %c to %c",source,destination);
        move(n-1,spare,destination,source);
    }
}
void main()
{
    move(3,'A','C','B');
}
```

مزایا و معایب برنامه های بازگشتی

- **مزایا:**
  - **خواناتر بودن**
  - **اگر قانون بازگشت به دست آورده شود نوشتن یک برنامه به صورت بازگشتی بسیار ساده تر از معادل غیر بازگشتی آن است.مثلا معادل غیربازگشتی برنامه برج های هانوی برنامه ای طولانی می باشد در حالی که برنامه بازگشتی آن با دو–سه خط نوشته می شود.**
- **معایب:**
  - **مصرف حافظه بیشتری دارند و سرعت اجرای آنها هم معمولا کمتر است. (بررسی به عنوان تحقیق)**

محل قرار گرفتن توابع

- در زمانی که تنها یک تابع داشتیم محل آن قبل از main و بعد از include ها بود.
- هنگامی که چند تابع وجود داشته باشند ؟
- ضابطه کلی:
  - یک تابع قبل آنکه احضار شود بایستی تعریف شده باشد.

مثال ۱

```
#include <stdio.h>
long fact(int n)
{
int i;
long result=1;
for (i=1;i<=n;++i)
  result*=i;
return result;
}
```

```
float pow(float x, int n)
{
int j;
float ret_val=1;
for (j=0;j<n;++j)
  ret_val=ret_val*x;
return ret_val;
}
```

```
float exp(float x, int m)
{
  int k;
float res=1;
for (k=1;k<=m;++k)
res+=pow(x,k)/fact(k);
return res;
}

void main()
{
  printf("%f",exp(2,10));
}
```

در این برنامه ضابطه گفته شده در صفحه قبل رعایت شده .
مثلا تابع pow **قبل از احضار در** exp **، تعریف شده است.**
یا تابع exp **قبل از احضار در** main **تعریف شده است.**

مثال ۲

```
#include <stdio.h>

long fact(int n)
{int i;
long result=1;
for (i=1;i<=n;++i)
  result*=i;
return result;}
```

```
float exp(float x, int m)
{  int k;
float res=1;
for (k=1;k<=m;++k)
res+=pow(x,k)/fact(k);
return res;}
```

```
float pow(float x, int n)
{int j;
float ret_val=1;
for (j=0;j<n;++j)
  ret_val=ret_val*x;
return ret_val;}
void main()
{
  printf("%f",exp(2,10));
}
```

در این برنامه ضابطه گفته شده رعایت نشده .
تابع pow در exp احضار شده در حالی که قبل از آن تعریف نشده است.
در این گونه موارد در اکثر کامپایلر ها با خطای کامپایلی زیر روبرو می شویم:
Call to undefined function pow
یعنی تابعی به نام pow، احضار (call) شده بدون اینکه تعریف شود.

ادامه

- برای سهولت بیشتر در نوشتن برنامه ها، در زبان C اجازه داده شده است که تابع بعد از محل احضار تعریف شود. مشروط بر اینکه خط اول تابع را قبل از احضار ذکر کنیم.

به این ترتیب complier می فهمد که چنین تابعی در برنامه وجود دارد وهنگامی که در جایی تابع احضار شد می داند چنین تابعی در برنامه وجود دارد و در نتیجه پیغام خطایی نمی دهد.

به این خط اصطلاحا عنوان تابع گفته می شود.

بعد از ذکر عنوان تابع گذاشتن ; الزامی است.

مثال

```
#include <stdio.h>
long fact(int n);
float pow(float x, int n);

float exp(float x, int m)
{   int k;
float res=1;
for (k=1;k<=m;++k)
res+=pow(x,k)/fact(k);
return res;}
```

```
long fact(int n)
{int i;
long result=1;
for (i=1;i<=n;++i)
  result*=i;
return result;}
```

```
float pow(float x, int n)
{int j;
float ret_val=1;
for (j=0;j<n;++j)
  ret_val=ret_val*x;
return ret_val;}
```

```
void main()
{
  printf("%f",exp(2,10));
}
```

در این مثال توابع pow و fact بعد از exp تعریف شده اند اما چون آنها را قبل از exp اعلان کرده ایم مشکلی پیش نمی آید.

ادامه

- تعریف تابع(Function definition): نوشتن کامل تابع
- اعلان تابع (Function declaration):نوشتن خط اول تابع

- هر تابع قبل از احضار می بایست یا تعریف شود یا اعلان شده باشد.

مزایای استفاده از توابع

- نوشتن برنامه های خواناتر : تقسیم برنامه به چند تابع سبب می شود که فهم برنامه آسان تر شود.

- امکان نوشتن پروژه های بزرگ به صورت ساده تر و سریع تر.
  - تقسیم پروژه به چند بخش.
  - انجام هر بخش توسط یک تیم مجزا . مثلا حاصل کار هر تیم، می تواند یک تابع باشد.
  - در کنار هم قرار دادن توابع و استفاده از آنها .

توصیه هایی جهت نوشتن برنامه های خوانا

۱. استفاده از توابع در نوشتن برنامه

۲. انتخاب نام های مناسب و با مسمی برای متغیرها و توابع
- مثلا برای تابعی که فاکتوریل را حساب می کند fact یک نام مناسب است.
- برای متغیری که حاصل جمع چند عدد را نگاه می دارد نام sum مناسب است.

ادامه

۳. دندانه ای کردن :رعایت یک ترتیب خاص در نوشتن خطوط برنامه
   - مثلا دستورات داخل تابع با کمی فاصله نسبت به خط اول تابع قرار گیرند.

```
void main()
{
   int x;
   x=4;
}
```

   - دستورات داخل شرط و حلقه ها با کمی فاصله نسبت به خود دستورات شرط و حلقه قرار گیرند.

```
if (x==0)
    return 1;
```

ادامه

۴. نوشتن توضیح (comment) برای دستورات

اگر در خطی از برنامه از // استفاده شود، کامپایلر از بعد از آن تا انتهای خط را هنگام کامپایل در نظر نمی گیرد. به این ترتیب می توانیم برای خطوط مختلف برنامه توضیحاتی به زبان معمولی خود بنویسیم.

comment نوشتن

```
#include <stdio.h>
long fact(int n) //This function compute factorial of n
{
      if (n==0)
       //stop condition
            return 1;
      else
            return n*fact(n-1);  //Recursion formula
}
void main()
{
      printf(“%d”,fact(2));
}
```

نوشتن comment

اگر توضیحات ما بیش از یک خط باشد از */ و /* استفاده می کنیم.

```
/*this program
  is written by .....
  Date: 1384/9/13 */
```

کامپایلر خطوطی را که بین */ و /* قرار گرفته اند را در نظر نمی گیرد.

آرایه ها

- آرایه ها
  - ضرورت وجود آرایه ها
  - تعریف آرایه
  - مثال هایی از کاربرد آرایه ها
  - آرایه ها به عنوان ورودی و برگشتی تابع
  - ثابت (constant)

ضرورت وجود آرایه ها

- **مثال: برنامه ای بنویسید که ۵ عدد را بگیرد و میانگین و واریانس آنها را محاسبه کند.**

نکته: میانگین و واریانس n نقطه x_0، x_1 و....و x_{n-1} از روابط زیر به دست می آیند:

$$\mu = \frac{\sum_{i=0}^{n-1} x_i}{n} \qquad \sigma^2 = \frac{\sum_{i=0}^{n-1} (x_i - \mu)^2}{n}$$

اگر تنها هدف به دست آوردن میانگین بود به راحتی می توانستیم آن را انجام دهیم:

```
float x;
sum=0;
for (i=0;i<5;++i)
{
    scanf("%f",&x);
    sum+=x;
}
ave=sum/5;
```

ادامه

- **در اینجا برای به دست آوردن واریانس نیاز به همه x_i ها داریم. اما در کد نوشته شده قبل تنها آخرین عدد را داریم و بقیه مقادیر از بین رفته اند.**

در نتیجه ناچاریم کد خود را به صورتی مثل این بنویسیم:

```
float x0,x1,x2,x3,x4;
float ave,var;
scanf(“%f%f%f%f%f”,&x0,&x1,&x2,&x3,&x4);
ave=(x0+x1+x2+x3+x4)/5;
var=(pow(x0-ave,2)+pow(x1-ave,2)+pow(x2-ave,2)+pow(x3-
    ave,2)+pow(x4-ave,2))/5;
```

ادامه

- به این ترتیب کد نامناسبی خواهیم داشت. این موضوع به ازای داده های زیادتر (مثلا ۱۰۰۰ عدد) بیشتر نمود پیدا می کند.
- در این جلسه نشان می دهیم با تعریف آرایه ها می توان چنین برنامه هایی را خیلی ساده و خلاصه پیاده کرد.
- به طور کلی آرایه ها در مواردی به کار برده می شوند که با مجموعه ای از داده های هم نوع سروکار داشته باشیم.

تعریف آرایه

- **یادآوری: با دستور تعریف متغیر مثل** int x; **مکانی در حافظه به نام** x **به برنامه تخصیص داده می شد.**

- **در زبان** C **اگر بنویسیم** int x[5]; **، پنج مکان مجاور هم در حافظه به برنامه تخصیص داده می شود. این مکان ها به صورت** x[0]**،** x[1]**،** x[2] **و** x[4] **نامیده می شوند.**
- **اصطلاحا می گوییم آرایه ای به نام** x **، از نوع** int **و با ۵ عنصر تعریف کرده ایم.**
- **به** 0**،** 1 **،**2**،** 3 **و** 4 **اصطلاحا اندیس آرایه گفته می شود.**
- **به همین ترتیب می توان آرایه هایی ازانواع دیگر مثل** float**،** char**،** short **و ....داشت.**

مثال ۱

```
Void main( )
{
   float num[3];
     num[0]=1.2;
     num [1]=2;
     num[2]=3.4;
   printf(“%f”,num[1])
}
```

```
Void main( )
{
   char c[4];
   c[2]=‘A’;
   c[0]=‘d’;
   c[1]=‘D’;
   c[3]=getche();
}
```

مثال ۲

```
#include <stdio.h>
void main( )
{
  int i;
  int x[4];
  for (i=0;i<4;++i)
      scanf(“%f”,&x[i]);
  for (i=0;i<4;++i)
     printf(“\t%f”,x[i]);
}
```

حلقه اول مقادیر را دریافت می کند و در آرایه x قرار می دهد و حلقه دوم این مقادیر را چاپ می کند.

ادامه

مثال ۳) محاسبه میانگین و واریانس ۵ عدد

```
#include <stdio.h>
#include <math.h>
void main( )
{
  float x[5];
  float sum=0 , ave;
  int I;
  for (i=0;i<5;++i)
  {
     scanf(“%f”,&x[i]);
     sum+=x[i];
  }
ave=sum/5;
var=0;
for (i=0;i<5;++i)
  var+=pow (x[i]-ave,2)/5;
printf(“The average=%f”,ave);
Printf(“The variance=%f”,var)
}
```

ادامه

مثال ۴) برنامه ای بنویسید که ۴۰ مقدار را از کاربر دریافت کرده و آنها در یک آرایه بریزد و سپس ماکزیمم مقادیر و اندیس عنصر ماکزیمم را محاسبه کند.

```
#include <stdio.h>
void main()
{       int i;
        float x[۴0];
        float max;
        int max_index;
        for (i=0;i<۴0;++i)
        {       printf("Enter  number %dth:",(i+1));
                scanf("%f",&x[i]);
        }
        max=x[0];
        max_index=0;
        for (i=1;i<۴0;++i)
                if (x[i]>max)
                {
                                max=x[i];
                                max_index=i;
                }
        printf("\nThe maximum=%f",max);
        printf("\nAnd its index=%d",max_index);
}
```

ادامه

مثال ۵) برنامه ای بنویسید که ۱۰ عدد از کاربر بگیرد آنها را به صورت صعودی مرتب کرده و چاپ کند.

برنامه از دو بخش تشکیل شده است:

دریافت اعداد

مرتب کردن آنها

1. **دریافت اعداد را می توان با تعریف یک آرایه و نوشتن قطعه برنامه ای به صورت زیر انجام داد:**

```
float x[10];
int i;
for (i=0;i<10;++i)
   scanf(“%f”,&x[i]);
```

ادامه

۲. به مسئله مرتب سازی اصطلاحا sorting **گفته می شود.**
برای مرتب سازی روش های متعددی وجود دارد. مانند :
bubble sort**(مرتب سازی حبابی)،** quick sort، merge sort **و.....**
در این مثال از bubble sort **که یکی از ساده ترین آنهاست استفاده می شود.**

Bubble sort برای مرتب سازی صعودی

- ابتدا دو عنصر اول و دوم آرایه با هم مقایسه می شوند.اگر عنصر اول از دوم بزرگتر بود جای دو عنصر عوض می شود. سپس عناصر دوم و سوم مقایسه می شوند و مشابه قبل تعویض مکان در صورت لزوم انجام می گیرد. همین کارها برای عناصر ۳ و ۴، ۴ و ۵، و....انجام می شود تا به انتهای آرایه برسیم.
- زمانی که به انتهای آرایه برسیم ماکزیمم مقادیر در آخرین مکان آرایه قرار گرفته است و به عبارت دیگر

زمانی که یک باراز ابتدا تا انتهای آرایه پیمایش شود عنصر آخر می شود.

- در مرحله بعد این پیمایش و اعمال گفته شده روی عناصر اول تا یکی مانده به آخر انجام می شود و در نتیجه :

با دو بار پیمایش ۲ عنصر مرتب می شوند.

- در مرحله بعد پیمایش از ابتدا تا عنصر دوتا مانده به آخر آرایه انجام می شود و در نتیجه:

با سه بار پیمایش سه عنصر مرتب می شود.

- **به این ترتیب اگر آرایه ، n عنصری باشد با n-1 پیمایش ، کل آرایه مرتب می شود.**

تعویض دو مقدار

- **فرض کنید دو متغیر** x=23 **و** y=34 **داشته باشیم و بخواهیم مقدار آنها را با هم تعویض کنیم.**
- **با تعریف متغیری به نام** temp **و نوشتن قطعه کد زیر می توان این کار را انجام داد:**

```
float temp;
temp=x;
x=y;
y=temp;
```

برنامه مثال ۵

```
#include <stdio.h>
void main()
{    int i,j;
     float A[10];
     float temp;
     for (i=0;i<10;++i)
          scanf("%f",&A[i]);
     for (i=0;i<9;++i)
          for (j=0;j<9-i;++j)
               if (A[j]>A[j+1])
               {
                    temp=A[j];
                    A[j]=A[j+1];
                    A[j+1]=temp;
               }
     for (i=0;i<10;++i)
          printf("\n%f",A[i]);
}
```

مقدار دهی به آرایه ها

- **سه روش برای مقدار دهی به آرایه ها وجود دارد:**
  - **با دستورات ورودی مثل** scanf **یا** getche**:**

```
ch[0]=getche();
scanf(“%d”,&x[2]);
```

  - **مقدار دهی مستقیم:**

```
ch[0]=‘a’;
x[2]=23;
```

  - **مقداردهی هنگام تعریف: (صفحه بعد)**

مقداردهی هنگام تعریف

- **می توان آرایه ها را (تنها)هنگام تعریف به صورت زیر مقداردهی کرد:**

```
int x[3]={2,3,-5};
```

در اینجا ۲ در x[0]، ۳ در x[1] و ۵– در x[3] قرار می گیرد.

مثال های دیگر:

```
float f[2]={0.34,4.33};
char ch[4]={‘a’,’d’,’x’,’v’};
```

چند سوال

- **اگر در مقداردهی هنگام تعریف، تعداد مقادیر نوشته شده داخل { } از اندازه مشخص شده برای آرایه بیشتر باشد خطا اعلام می شود.**

```
int x[2]={3,4,6,7,8}
```

- **اگر تعداد مقادیر کمتر باشد، بقیه صفر در نظر گرفته می شوند:**

```
int x[3]={3,4}
```

در اینجا x[2]=0 **می شود.**

مثال دیگر:

```
int x[100]={0}
```

همه مقادیر صفر می شوند. (این روشی برای صفر کردن همه عناصر آرایه است).

- **اگر بنویسیم:** int x[ ]={2,3} **اندازه آرایه ۲ در نظر گرفته می شود.**

یک نکته

```
Int i=8;
Int x[i];
```

اندازه آرایه باید یک عدد ثابت باشد. بنابراین نوشتن به صورت بالا نادرست است.

آرایه ها به عنوان ورودی و برگشتی توابع

- آرایه ها مانند سایر متغیرها می توانند ورودی یا برگشتی تابع باشند.

آرایه به عنوان ورودی تابع

مثال ۶: تابعی بنویسید که یک آرایه را بگیرد و ماکزیمم آن را برگرداند:
ورودی: آرایه برگشتی: یک عدد

```
float max(float x[10])
{
    int i;
    float  max=x[0];
    for (i=1;i<10;++i)
          if (x[i]>max)
                    max=x[i];
    return max;
}
```

برنامه کامل مثال ۶

```
#include <stdio.h>
float max(float x[10])
{
	int i;
	float  max=x[0];
	for (i=1;i<10;++i)
		if (x[i]>max)
				max=x[i];
	return max;
}

void main()
{
	int i;
	float A[10];
	for (i=0;i<10;++i)
		scanf("%f",&A[i]);
	printf("\nThe maximum=%f",max(A));
}
```

ثابت (Constant)

- در بسیاری مواقع نیاز به کار با مقادیر ثابت در برنامه، پیش می آید و این مقادیر ممکن است چند بار در برنامه تکرار شوند. مانند عدد ۱۰ در مثال قبل
- اگر در مثال قبل بخواهیم ماکزیمم فرضا ۲۰ عدد را حساب کنیم باید تمام اعداد ۱۰ را به ۲۰ تبدیل کنیم. این امر دو مشکل دارد: وقت گیر بودن و فراموش کردن تعویض همه مقادیر.
- دستور define در زبان C این مشکل را مرتفع می سازد.

دستور define

مثال ۷: همان مثال قبل با تغییرات جزئی: (با رنگ قرمز)

```
#include <stdio.h>
#define SIZE 10
float max(float x[SIZE])
{
        int i;
        float  max=x[0];
        for (i=1;i<SIZE;++i)
                if (x[i]>max)
                   max=x[i];
        return max;
}

void main()
{
        int i;
        float A[SIZE];
        for (i=0;i<SIZE;++i)
                scanf("%f",&A[i]);
        printf("\nThe maximum=%f",max(A));
}
```

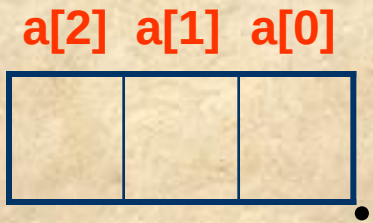
- دستور define، مانند include یک دستور راهنمای پیش پردازنده است.
- کامپایلر در هر جای برنامه که با SIZE مواجه شود به جای آن عدد ۱۰ قرار می دهد.
- SIZE اصطلاحا یک ثابت (constant) خوانده می شود.
- اسم ثابت، دلخواه است. معمولا ثابت ها را با نام های با حروف بزرگ تعریف می کنند.
- مقدار ثابت در برنامه غیرقابل تغییر است. مثلا اگر دستوری مثل SIZE=23 در برنامه بنویسیم خطا داده می شود.
- اگر مثلا خواستیم با ۲۰ عدد کار کنیم تنها کافی است عدد ۱۰ را در define به ۲۰ تبدیل کنیم.
- محل دستور define در بخش include هاست. (ترتیب نوشتن include ها و define مهم نیست و می توانند قبل یا بعد هم قرار گیرند)

ادامه آرایه

- تعریف آرایه:

```
int a[3];
```

- آرایه ای به نام a از نوع int و با اندازه ۳ عنصر تعریف کرده ایم.



- به عناصر آرایه می توان از طریق اندیسشان دسترسی داشت.

```
a[0]=4;
scanf(“%d”,a[1]);
for (i=0;i<3;++i)
    scanf(“%d”,&a[i]);
```

ادامه آرایه

مقداردهی اولیه:

```
float a[4]={4.34,9.45,3.23,1.02};
```

اگر تعداد مقادیر از اندازه مشخص شده برای آرایه بیشتر باشد خطا رخ می دهد:

```
float a[3]={7.34,1.2,3.4,1.44};
```

اگر تعداد مقادیر از اندازه کمتر باشد، بقیه مقادیر صفر در نظر گرفته می شوند:

```
float a[3]={2.34,3.23};
```

نحوه تعریف رشته ها

- **در C، رشته ها به صورت آرایه ای از کاراکترها پیاده می شوند.**
  - **با نوشتن** char st [20]; **می توانیم رشته ای به نام** st **با ماکزیمم طول ۲۰ تعریف کرده ایم.**

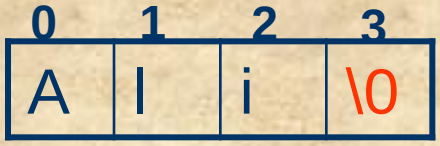
نحوه مقداردهی

- **نحوه مقداردهی: به دو صورت می توان به یک رشته مقدار داد:**
  1. **با استفاده از دستورات ورودی: مثل scanf با استفاده از کارکتر کنترلی %s**
     - **scanf(“%s”,st) یا scanf(“%s”,&st);**
  2. **به صورت مستقیم هنگام تعریف رشته**

```
char st[20]=“Ali”;
```

- **نحوه قرار گرفتن رشته در حافظه به این صورت است:**

- **C در انتهای هر رشته به طور خودکار کاراکتر NULL (پوچ–تهی) (که با علامت \0 نشان داده می شود) را قرار می دهد.**
- **در نتیجه طول واقعی یک رشته یکی بیشتر از تعداد کاراکترهایش است.**



ادامه

نکته:کد اسکی کاراکتر NULL **، صفر است.**

printf(“%c”,’\0’);**. کاری انجام نمی شود**

printf(“%d”,’\0’); **صفر چاپ می شود.**

نحوه چاپ یک رشته: با دستور printf **و استفاده از کاراکتر کنترلی** %s

ادامه

- **مثال ۱)**

```
#include <stdio.h>
void main()
{
    char st[20]="Hello";
    printf("%s",s);
}
```

رشته ای به نام st **با مقدار** Hello **تعریف شده .خروجی این برنامه پیغام** Hello **است.**

می توان این برنامه را با یک دستور زیر نوشت:

```
printf(“%s”,”Hello”);
```

ادامه

مثال ۲) رشته ای را از کاربر دریافت کرده و آن را چاپ می کند.

```
#include <stdio.h>
void main()
{
    char ch[4];
    scanf("%s",ch);
    printf("%s",ch);
}
```

ادامه

مثال ۳)

```
#include <stdio.h>
void main()
{
    char str[23]="Hello";
    printf("%c",str[2]);
}
```

با استفاده از اندیس می توان به کاراکترهای داخل رشته دسترسی داشت.

ادامه

نکته: چند وضعیت مختلف در مقداردهی هنگام تعریف

۱. رشته از اندازه آرایه کوچکتر است.

```
char  s[8]="abc";
```

بقیه عناصر را NULL **در نظر می گیرد.**

۲. رشته از اندازه آرایه بزرگتر است

```
char s[3]="aghugii";
```

خطا

۳. رشته مساوی اندازه آرایه است.

```
char s[3]="ali";
```

خطا نیست. در اینجا کاراکتر NULL **جزو رشته حساب نمی شود.**

ادامه

مثال ۴) برنامه ای بنویسید که طول یک رشته را حساب کند (کاراکتر NULL در طول رشته در نظر گرفته نشود).

```
#include <stdio.h>
int strlen(char str[20])
{
    int i=0;
    while (str[i] != '\0')
        i++;
   return i;
}
void main()
{
    char str[20];
    scanf("%s",str);
    printf("%d",strlen(str));
}
```

ادامه

- مثال ۴) برنامه ای بنویسید که یک رشته از کاربر بگیرد و آن را به رشته ای که تمام حروفش بزرگ است تبدیل کند. (مثلا "abAF" به "ABAF" تبدیل شود.

```
#include <stdio.h>
void main()
{
	char str[20];
	int i;
	printf("Enter a string:");
	scanf("%s",str);
	i=0;
	while (str[i] != '\0')
	{
		if (str[i]>=97 && str[i]<=122)
			printf("%c",str[i++]-32);
	}
}
```

چند تابع کتابخانه ای برای کار با رشته ها

strlen: ورودی آن یک رشته و برگشتی آن طول رشته است.

```
x=strlen(“ali”);
printf(“%d”,x);
```

puts: برای چاپ رشته:

```
puts(“ali is a student”);
char ch[20]=“ali;
puts(ch);
```

عملکردی مشابه printf دارد؛ با دو تفاوت:

- فقط برای نمایش رشته به کار می رود و نمی تواند مقدار متغیر ها را نشان دهد.
- بعد از چاپ رشته به خط بعد می رود.

gets: برای دریافت رشته از کاربر

```
char ss[20];
gets(ss);
```

ادامه

strcat : برای الحاق دو رشته به یکدیگر به کار می رود:strcat(s1,s2) رشته s2 را به انتهای s1 اضافه می کند.

```
#include <string.h>
#include <stdio.h>
void main()
{
    char s1[20]="Ali";
    char s2[20]="Mehdi";
   strcat(s1,s2);
   strcat(s2,”\nhassan”);
    puts(s1);
     puts(s2);
}
```

www.engclubs.net
t.me/engclubs