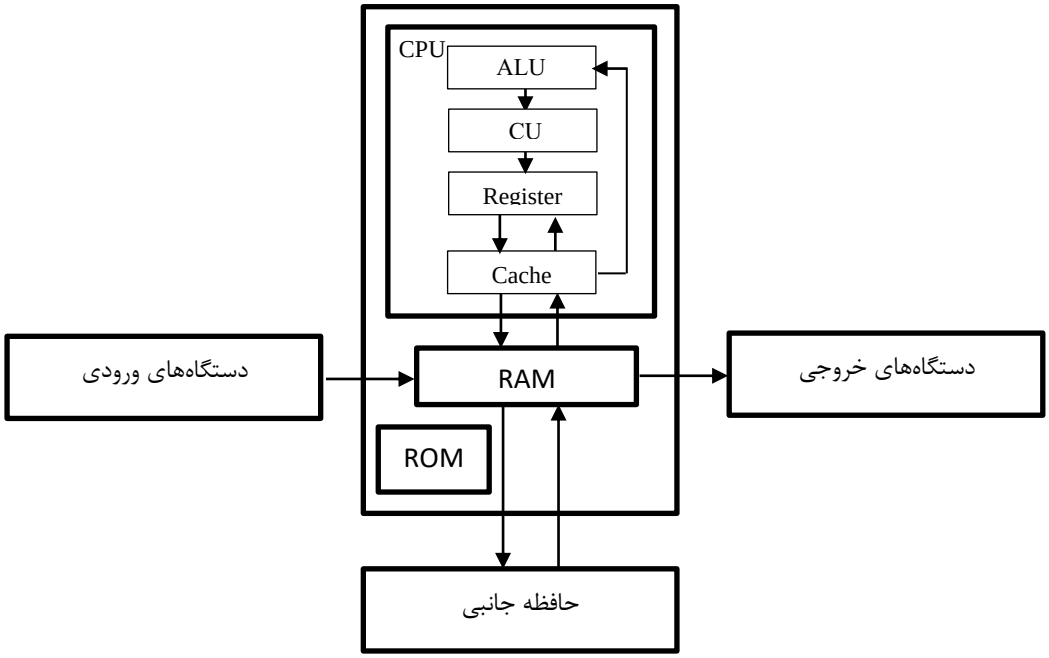
مباحث کلاسی

برنامه نویسی کامپیوتر

مدرس : مهدی مجد

طرح کلی سیستم‌های کامپیوتری

اجزای اصلی سیستم های پردازشی با توجه به کامپیوترهای شخصی (PC) مورد بررسی قرار می‌گیرد. کلیه سیستم‌های پردازشی ساختار مشابهی دارند اما با مقدار کمی تفاوت.



در سیستم پردازشی چند جزء مهم وجود دارد.

۱- CPU : واحد مرکزی پردازش – Central Processing Unit
۲- RAM : حافظه با دسترسی تصادفی – Random Access Memory
۳- ROM : حافظه فقط خواندنی - Read Only Memory

CPU : **تراشه ای** است که مستقیماً به بُرد اصلی متّصل می شود و در حکم «مغز» برای سیستم است . وظایف ریزپردازنده عبارت اند از : کنترل و اجرای دستورالعمل ها، ایجاد هماهنگی بین فعّالیّت‌های اجزای مختلف سیستم، تشخیص نوع عملیّات و ترتیب اجرای آنها، آوردن اطّلاعات مورد نیاز از حافظه به داخل ریزپردازنده و ذخیره‌ی نتیجه‌ی عملیات در حافظه. ریزپردازنده شامل چند جزء مهم می‌باشد.

۱- Cache : حافظه میانی که یک نوع حافظه‌ی بسیار سریع تر از RAM است که به دلیل اختلاف سرعت CPUوRAM به عنوان حافظه‌ی واسط بین این دو به کار می رود .در این حافظه، آدرس و محتوای خانه هایی از حافظه‌ی RAM که بیشتر مورد استفاده اند، ذخیره می شود. ریزپردازنده، هنگام نیاز به اطّلاعات، ابتدا به حافظه‌ی cache مراجعه می کند و اگر اطّلاعات مورد نیاز را پیدا نکرد، به آدرس اطّلاعات در RAM مراجعه می کند. بدین ترتیب، سرعت اجرای برنامه ها به طور قابل توجهی افزایش می یابد. این نوع حافظه درون CPU یا بر روی برد اصلی قرار می گیرد.

۲- رجیستر : حافظه موقتی که محاسبات بر روی آن اجرا می‌شوند.

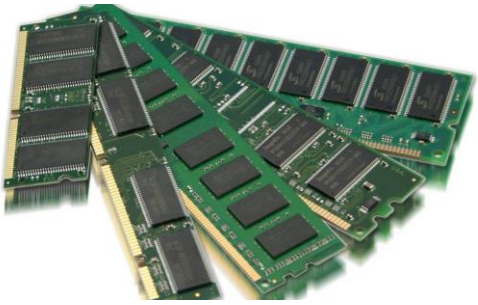
۳- ALU : واحد حساب و منطق یا ALU واحد حساب و منطق عملیات محاسباتی از قبیل :جمع، تفریق، ضرب و تقسیم را انجام می دهد .این واحد هم چنین عملیات منطقی برای مقایسه‌ی دو داده (مانند بزرگ تر، کوچک تر و یا مساوی)را انجام می دهد .سایر عملیات، با ترکیب این عملیات ساده انجام می شوند.

۴- واحد کنترل یا CU : همان طور که از اسم آن پیداست این واحد وظیفه‌ی کنترل و ایجاد هماهنگی بین قسمت های مختلف سیستم را بر عهده دارد .کنترل ورود داده ها از طریق واحد ورودی، ذخیره-ی آنها در حافظه، انتقال اطّلاعات از حافظه به واحد حساب و منطق و برعکس، رمزگشایی دستورالعمل ها و در نهایت ارسال اطّلاعات به واحد خروجی از وظایف این واحدند.

RAM : **حافظهٔ RAM** ، یک حافظه‌ی موقّتی برای نگهداری برنامه و داده ای است که واحد پردازنده‌ی مرکزی در حال پردازش آن است. برنامه ها برای اجرا باید در حافظه‌ی RAM قرار گیرند؛ به همین دلیل گاهی به آن **حافظهٔ کاری** می گویند .چون دست یابی به داده‌ی موجود در حافظه‌ی RAM به محلّ قرارگرفتن آن بستگی ندارد، به آن **حافظه با دست یابی تصادفی** گفته می شود .هم چنین به دلیل این که با خاموش شدن رایانه و یا قطع برق تمام محتوای حافظه‌ی RAM از بین می رود، به آن **حافظهٔ موقّتی** می گویند. به همین دلیل هنگام کار روی برنامه و یا داده‌ی خاص، لازم است هر چند دقیقه یک بار نتایج کار را در حافظه‌ی جانبی(مثلاً یک دیسک نرم و یا سخت)ذخیره کرد تا اگر به دلیلی مثل قطع برق، اطّلاعات حافظه‌ی RAM از بین رفت، یک نسخه از آن اطّلاعات وجود داشته باشد .در ضمن حافظه‌ی RAMاز نوع **خواندنی نوشتنی** است؛ یعنی هم می توان از آن اطّلاعات را خواند و هم روی آن نوشت .حافظه های RAM با ظرفیت های متفاوتی وجود دارند و درون شیارهای مربوط بر روی بُرد اصلی قرار می گیرند. ظرفیت حافظه‌ی RAM یکی از عوامل تعیین کننده‌ی

سرعت اجرای برنامه است. ظرفیت های متفاوتی وجود دارند و درون شیارهای مربوط بر روی بُرد اصلی قرار می-گیرند ظرفیت حافظه‌ی RAM یکی از عوامل تعیین کننده‌ی سرعت اجرای برنامه است .

پردازنده مرکزی (CPU)

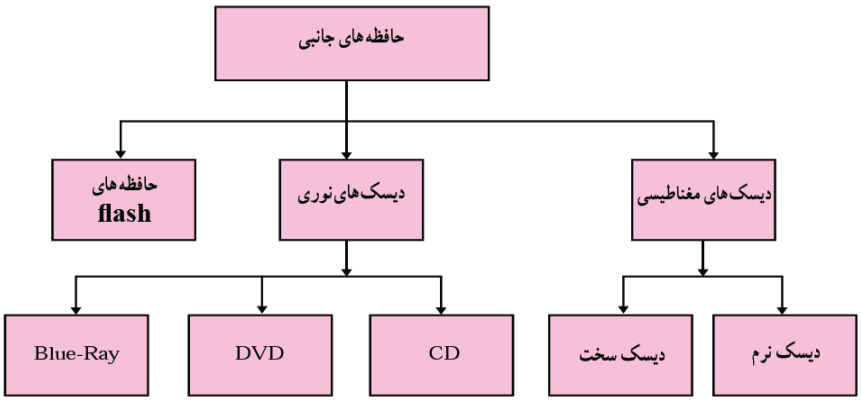


حافظه RAM

ROM : حافظه‌ی ROM ، حافظه ای است **فقط خواندنی** که محتوای آن **ثابت** و **غیرقابل تغییر** است. این حافظه برای ذخیرهی **دائمی** اطّلاعات اوّلیه درباره‌ی مشخّصات سیستم به کار می رود؛ برای مثال، حافظه‌ی ROM شامل برنامه ها و دستورالعمل هایی است که برای راه اندازی رایانه مورد نیازند .سازندگان برد اصلی، داده‌ی لازم را در تراشه های این حافظه ذخیره می کنند و کاربر نمی تواند اطّلاعات موجود در آن را تغییر دهد. بدیهی است خاموش کردن دستگاه رایانه و هم چنین قطع برق هیچ تأثیری بر اطّلاعات وجود در حافظه‌ی ROM ندارد.

حافظه جانبی : حافظه‌ی اصلی در سیستم رایانه یکی از منابع محدود و در عین حال پرکاربرد است .این حافظه، آن قدر بزرگ نیست که بتواند. تمام برنامه ها و داده های مورد نیاز شما را در خود جای دهد؛ قسمتی از آن توسّط سازندگان آن قابل نوشتن است .) ROM (ازطرف دیگر، قسمت عمده‌ی حافظه‌ی اصلی) RAM (موقّتی است و نمی تواند اطّلاعات را به طور دائم در خود نگه دارد؛ لذا برای ذخیره‌ی دائمی برنامه ها و داده ها از **حافظهٔ جانبی** استفاده می کنیم و هرگاه که لازم باشد آنها را به حافظه‌ی اصلی منتقل می کنیم.

دیسک سخت (Hard Disk)

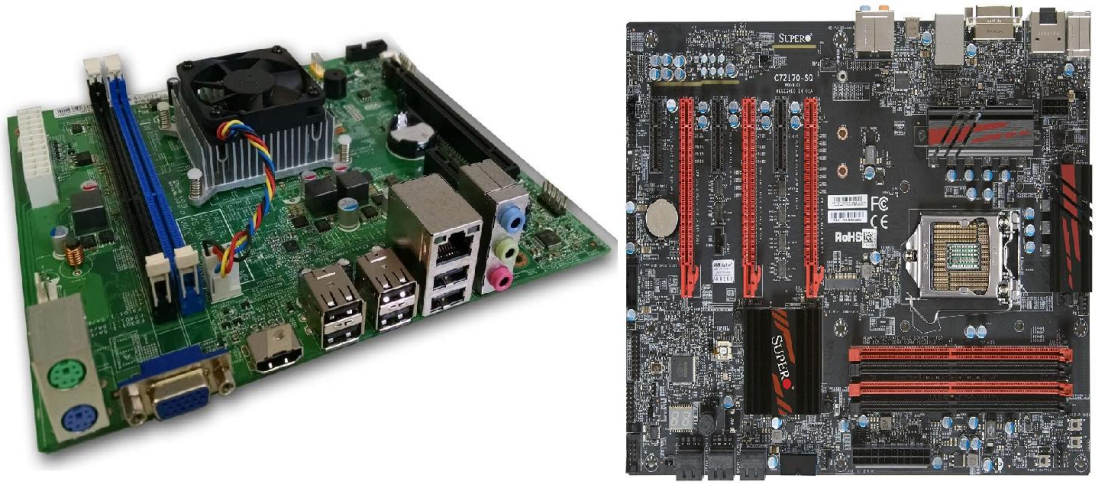


انواع حافظه های جانبی

دستگاه‌های ورودی : دستگاهی همانند صفحه کلید، ماوس، اسکنر، میکروفون، دوربین

دستگاه‌های خروجی : دستگاهی همانند صفحه نمایش، چاپگر، رسام، بلندگو و گوشی

لازم به توضیح است که در کامپیوتر های شخصی قطعات اصلی سیستم مانند **ریزپردازنده** و **حافظه**، روی برد اصلی قرار می گیرند همچنین وسایلی که در بیرون محفظه‌ی کامپیوتر قرار دارند از قبیل صفحه کلید، ماوس و صفحه نمایش فقط از طریق بُرد اصلی می توانند با سیستم کامپیوتر ارتباط برقرار کنند.



برد اصلی (Motherboard)

دستورهایی که پردازنده انها را اجرا می کند :

۱- دستورهای ورودی و خروجی :

۲- دستور های محاسبات ریاضی و منطقی :

ریاضی همانند جمع- ضرب- جذر - به توان رساندن- ... منطقی همانند بزرگتر – کوچکتر – and – or - ۰۰۰۰

۳- **دستورهای کنترل اجرا** :

دستور انتخاب (تصمیم گیری) - حلقه تکرار

شبه کدی بنویسید که سه عدد را از ورودی خوانده میانگین آنها را نمایش دهد.

۰- شروع
۱- سه مقدار a و b و c را بخوان.
۲- s = a + b +c
۳- avg = s / 3
۴- avg را نمایش بده.
۵- پایان

مثال : شبه کدی دو مقدار a و b را از ورودی خوانده ریشه معادله ax+b=0 را نمایش دهد.

ax = - b , x = -b / a

۰- شروع
۱- دو مقدار a و b را بخوان.
۲- x = - b / a
۳- x را نمایش بده.
۴- پایان.

مثال برای **دستور انتخاب :**

قالب کلی :

اگر (شرط) برقرار باشد آنگاه

.....

در غیر این صورت

.....

شبه کدی بنویسید که مقداری از ورودی خوانده مثبت یا منفی بودن آن را در خروجی نمایش دهد.

۰- شروع
۱- a را بخوان.
۲- اگر (a >= 0) باشد آنگاه
۲-۱- "مثبت" را در خروجی نمایش بده.
در غیر این صورت
۲-۲- "منفی" را در خروجی نمایش بده.
۳- پایان.

تمرین : شبه کدی بنویسید که مقداری را به عنوان نمره یک دانشجو از ورودی خوانده قبولی یا عدم قبولی را در خروجی مشخص کند. (نمره کوچکتر از ۱۰ مردود – نمره ۱۰ به بالا قبول می باشد.)

حلقه تکرار معین:

با استفاده از این دستورالعمل می‌توان مجموعه ای از دستورالعمل های دیگر را به دفعات از پیش تعیین شده ای تکرار نمود.

قالب کلی :

- دستور های زیر را .... مرتبه تکرار کن.

.....

.....

مثال : شبه کدی بنویسید که ۴۰ مقدار را از ورودی خوانده مجموع آنها را نمایش بدهد.

۰- شروع

۱- S=0

۲- دستورهای زیر را ۴۰ مرتبه تکرار کن

۲-۱- a را بخوان.

۲-۲- s = s + a

۳- s را نمایش بده.

۴- پایان.

شبه کدی بنویسید که مجموع مقادیر ۱ تا ۴۰ را محاسبه کرده نمایش دهد.

۰- شروع

۱- i = 1

۲- دستورهای زیر را ۴۰ مرتبه تکرارکن.

۲-۱ i را نمایش بده.

۲-۲ i = i + 1

۳- پایان

شبه کدی بنویسید که حاصلضرب مقادیر ۱ تا ۴۰ را محاسبه کرده نمایش دهد.

۰- شروع

۱- i = 1 و f = 1

۲- دستورهای زیر را ۴۰ مرتبه تکرار کن

۲-۱- f = f * i

۲-۲- i = i + 1

۳- F را نمایش بده.

۴- پایان

شبه کدی بنویسید که مقداری را از ورودی خوانده فاکتوریل آن را نمایش دهد.

۰- شروع
۱- n را بخوان.
۲- i = 1 و f = 1
۳- دستورهای زیر را n مرتبه تکرار کنید.
  ۳-۱- f = f * i
  ۳-۲- i = i + 1
۴- f را نمایش بده.
۵- پایان

حلقه تکرار نامعین:

در این دستورالعمل دفعات تکرار زیر مجموعه قبل از اجرا تعیین نخواهد شد، بلکه تکرار بر اساس یک شرط انجام خواهد شد و تا زمانی که شرط صحیح باشد.

قالب کلی :

- دستورهای زیر را تا زمانی تکرار کن که (شرط) برقرار باشد
  - .....
  - .....

شبه کدی بنویسید که چند مقدار را از ورودی دریافت کرده مجموع آنها را نمایش دهد. خواندن مقدار تا زمانی ادامه یابد که مقدار صفر وارد شود.

۰- شروع
۱- s = 0
۲- a را بخوان.
۳- دستورهای زیر را تا زمانی تکرار کن که (a != 0) باشد.
    ۳-۱- s = s + a
    ۳-۲- a را بخوان
۴- s را نمایش بده.
۵- پایان

مثال : شبه کدی بنویسید که چند مقدار را از ورودی دریافت کرده تعداد مقادیر بزرگتر از صفر را نمایش دهد. خواندن مقدار تا زمانی ادامه یابد که مقدار ۱- وارد شود.

۰- شروع
۱- c = 0
۲- a را بخوان.
۳- دستورهای زیر را تا زمانی تکرار کن که (a != -1) باشد.
    ۳-۱- اگر (a > 0) بشد آنگاه
        ۳-۱-۱- c = c + 1
    ۳-۲- a را بخوان.
۴- c را نمایش بده.
۵- پایان

تمرین : شبه کدی بنویسید که چند مقدار را از ورودی دریافت کرده میانگین آنها را نمایش دهد. خواندن مقدار تا زمانی ادامه یابد که مقدار صفر وارد شود.

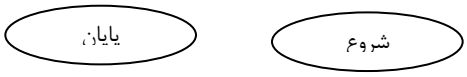
فلوچارت

برای ارائه الگوریتم به صورت گرافیکی تا مخاطب به آسانی و با سرعت بالا یک دید کلی نسبت روند اجرای الگوریتم پیدا کند.

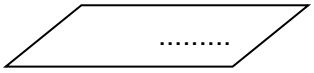
استانداردها مختلفی برای طراحی فلوچارت وجود دارد که یکی از آنها را در اینجا مورد استفاده قرار خواهیم داد.

شکل های مورد استفاده در رسم فلوچارت:

۱- بیضی : برای شروع و پایان الگوریتم

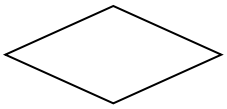


۲- متوازی الاضلاع : برای ورودی و خورجی

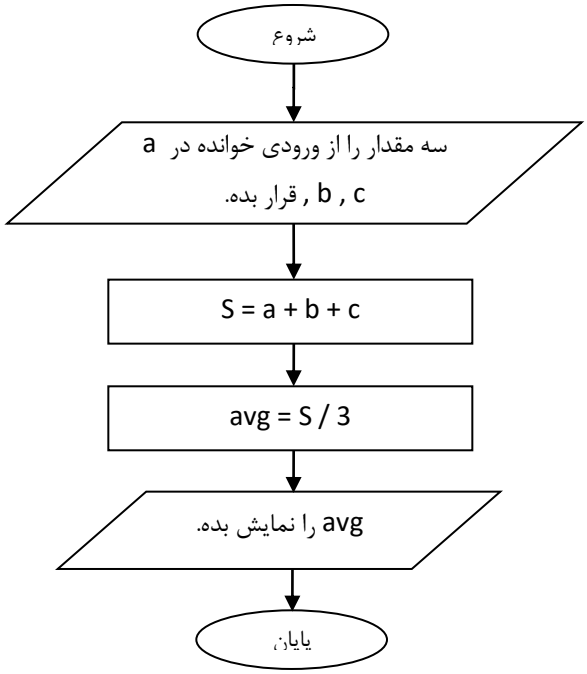


۳- مستطیل : برای محاسبات

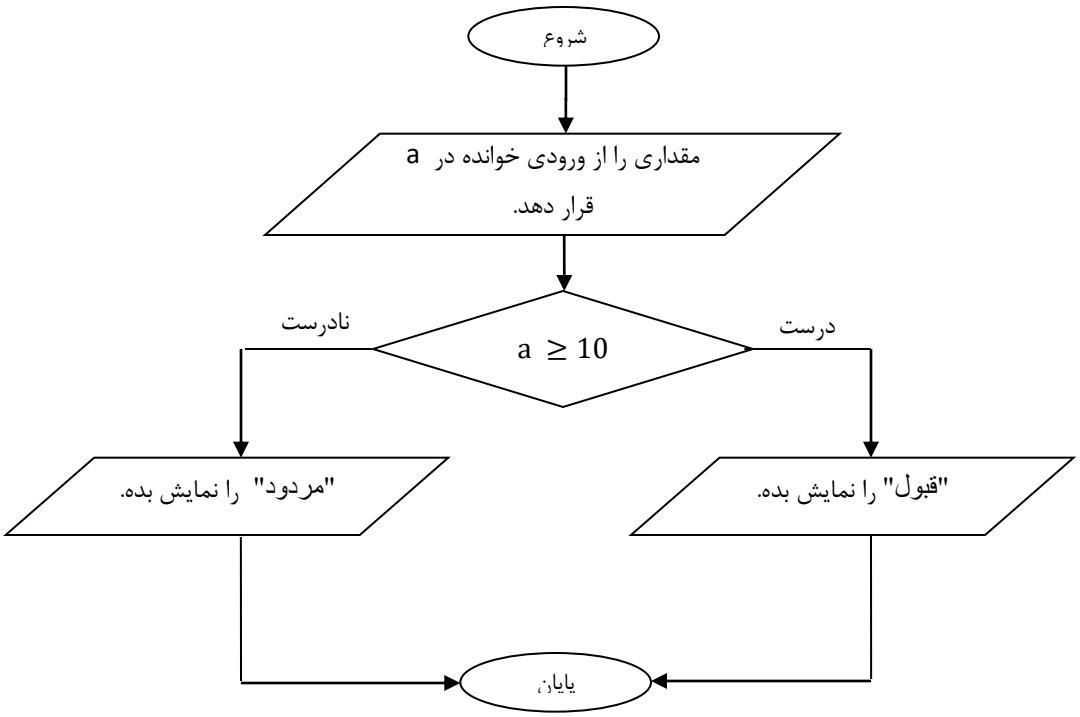
۴- لوزی : برای بیان شرط



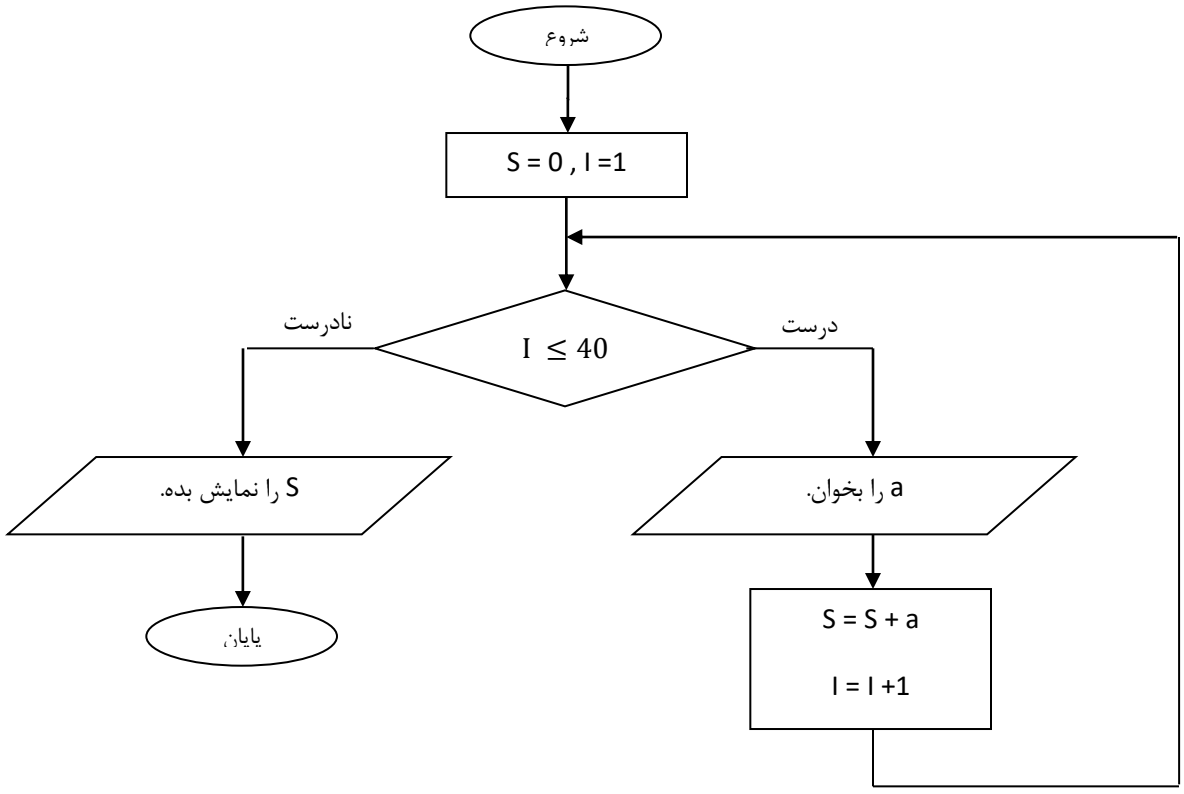
مثال : فلوچارتی رسم کنید که سه عدد را از ورودی خوانده میانگین آنها را نمایش دهد.



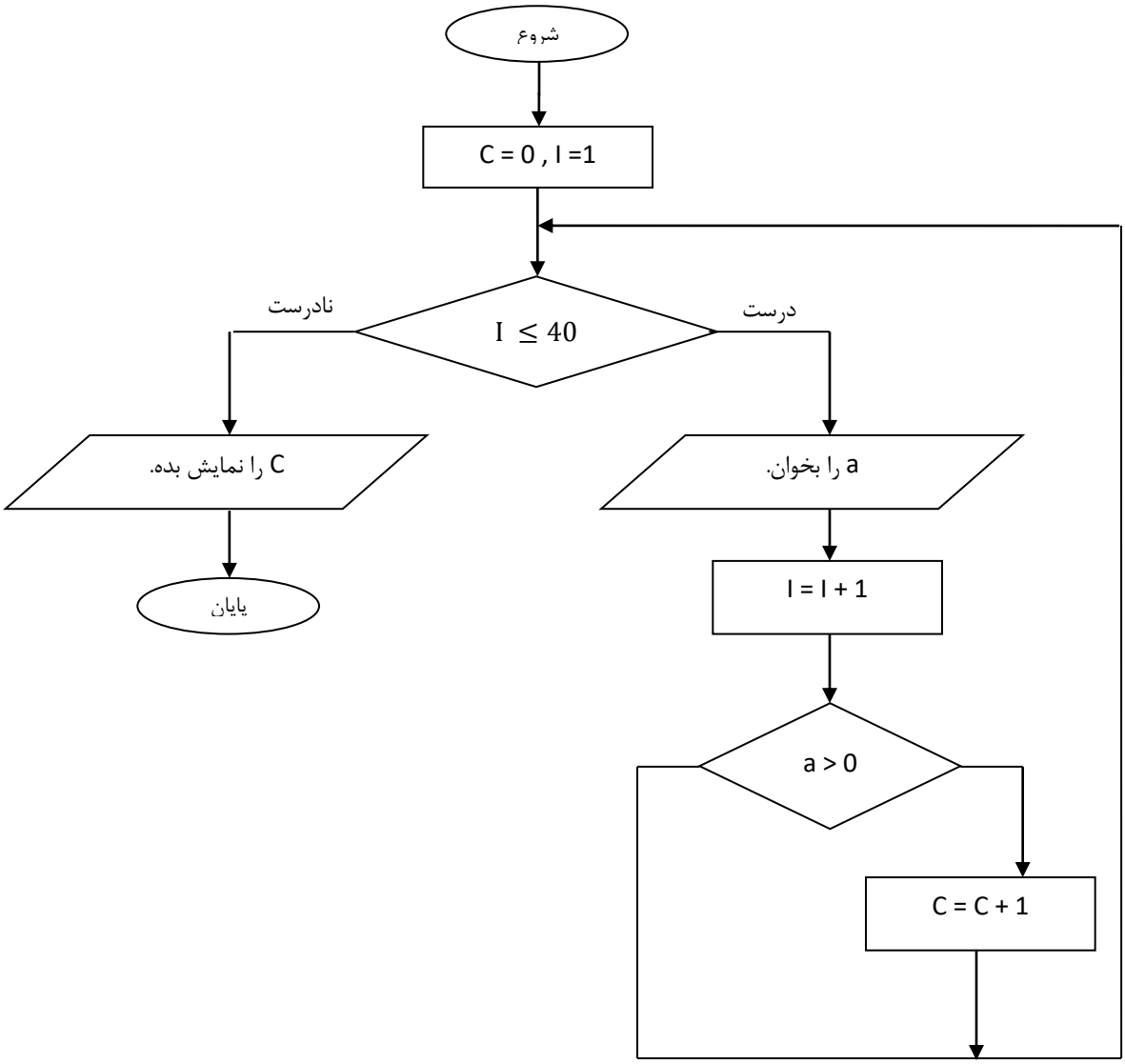
مثال : فلوچارتی رسم کنید که عددی را به عنوان نمره یک دانشجو دریافت کرده قبولی یا مردودی آن در خروجی مشخص کنید.



فلوچارتی رسم کنید که ۴۰ عدد را از ورودی خوانده مجموع آنها را نمایش دهد.



فلوچارتی رسم کنید که ۴۰ عدد را از ورودی خوانده تعداد مقادیر بزرگتر از صفر را شمارش کند.

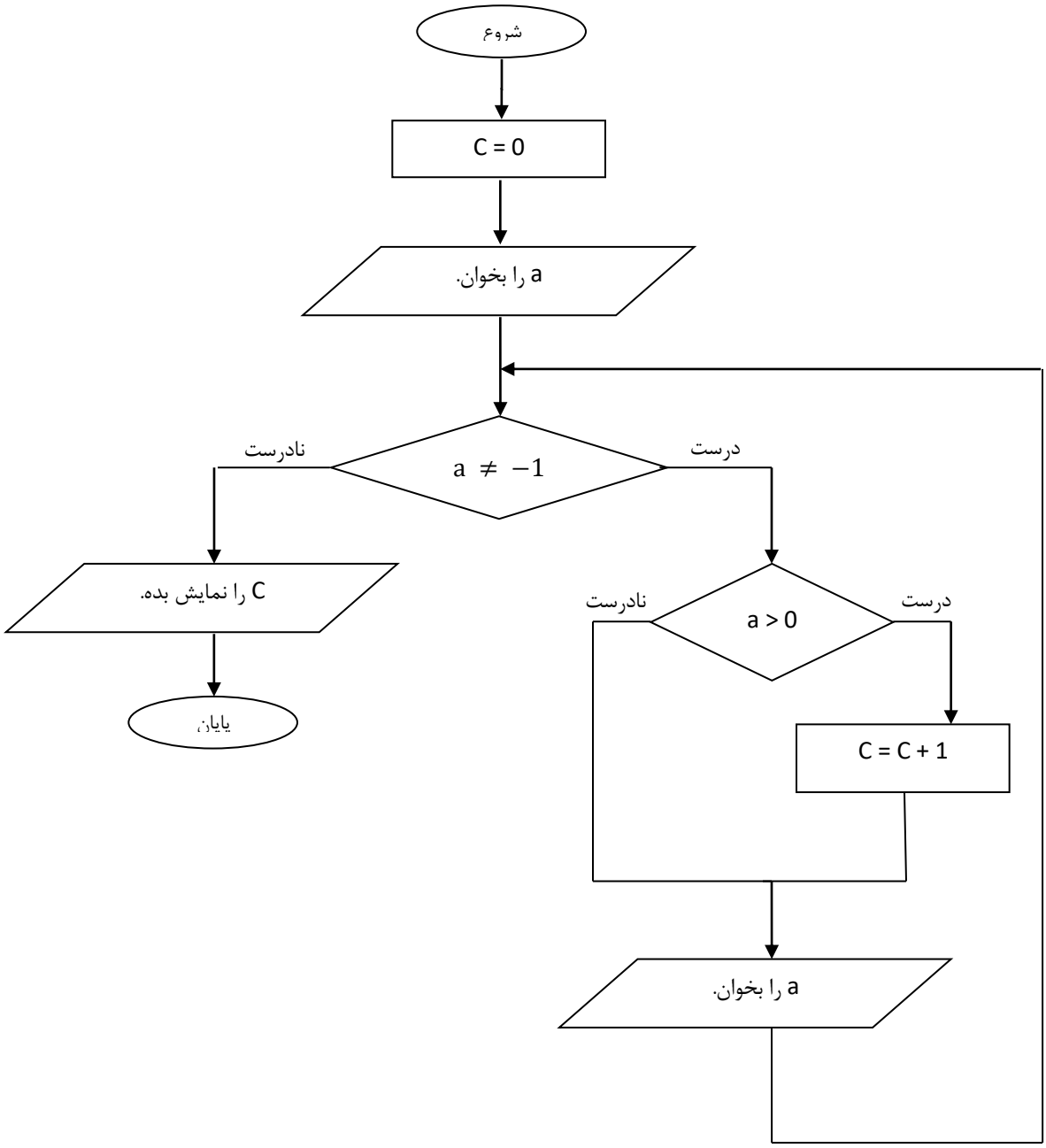


تمرین : فلوچارتی رسم کنید که مقدار n را از ورودی خوانده، فاکتوریل آن را نمایش دهد.

$$n\,! = 1 * 2 * 3 * 4 * \ldots * n$$

تمرین : فلوچارتی رسم کنید که ۵۰ عدد را از ورودی خوانده میانگین مقادیر بزرگتر از صفر را محسبه کرده نمایش دهد.

مثال : فلوچارتی رسم کنید که چند مقدار را از ورودی دریافت کرده تعداد مقادیر بزرگتر از صفر را نمایش دهد. خواندن مقدار تا زمانی ادامه یابد که مقدار ۱- وارد شود.



تمرین : فلوچارتی رسم کنید که چند مقدار را از ورودی دریافت کرده مجموع آنها را نمایش دهد. خواندن مقدار تا زمانی ادامه یابد که مقدار صفر وارد شود.

برنامه نویسی به زبان C

الگوریتم های بیان شده به دو روش شبه کد و فلوچارت توسط کامپیوتر قابل اجرا نمی باشند. به منظور اجرای الگوریتم ها باید آنها را در قالب یکی از زبان های برنامه برنامه نویسی بیان نمود. ما در این درس بنا به دلایل مختلف از زبان برنامه نویسی C استفاده خواهیم کرد. از دلایل موارد زیر را می توان بیان کرد.

- انعطاف پذیری بالای دستورها
- دسترسی سطح پایین به سخت افزار
- استفاده از زبان در برنامه نویسی بسیاری از چیپ ها و کنترل کننده ها
- شباهت بسیار زیاد زبان ها دیگر به دستورها اصلی زبان C
- ...

متغیر ها در:c

متغیر، یک نام می‌باشد که می‌تواند مقدارش در طول الگوریتم تغییر کند.

قوانین نامگذاری متغیرها:

- حروف A-Z یا) a-z در زبانC ، حروف کوچک و بزرگ با هم تفاوت دارند(.
- ارقام ۰-۹
- کاراکتر_
- حتما بایستی با یک حرف شروع شود**temp34 a_z sum a-b lhs3 Ab**.

داده ها در زبان:c

در زبان c شش نوع داده اصلی وجود دارد که عبارت اند از:

char : برای داده های کاراکتری

int : برای اعداد صحیح

float : برای اعداد اعشاری

double : برای اعداد اعشاری بزرگ ترfloat

عملگرهای محاسباتی در:c

ردیف	عملگر	نام	مثال
۱	+	جمع	x + y
۳	-	تفریق و منهای یکانی	x – y , -x
۳	*	ضرب	x * y
٤	/	تقسیم	x / y
٥	%	باقیمانده تقسیم صحیح	a % b
٦	--	کاهش یک واحد	x--یا --x
۷	++	افزایش یک واحد	++x یا x++

عملگرهای رابطه ای در:c

مثال	نام	عملگر
x > y	بزرگتر	>
x >=y	بزرگتر مساوی	>=
x < y	کوچکتر	<
x <= y	کوچکتر مساوی	<=
x == y	مساوی بودن	==
x != y	نامساوی	!=

عملگرهای منطقی در:c

مثال	نام	عملگر
a>y && y<x	و (And)	&&
x>y || y<x	یا (OR)	||
! x	نقیض (Not)	!

تقدم عملگرها در:c

۱. ()

۲. ! ++ —

۳. * / %

۴. + –

۵. =< < => > <

۶. =! ==

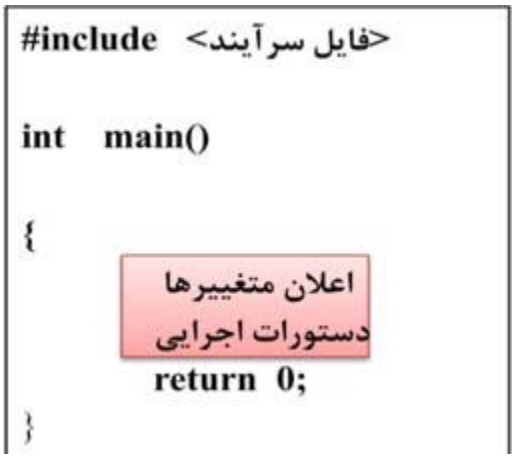
۷. &&

۸. ||

۹. ?

۱۰. =% =/ =* =- =+ =

برنامه ها در C :



برنامه ها در c از مجموعه ای از دستورات و تعدادی تابع تشکیل میشوند.

هرتابع برای حل بخشی از مسئله نوشته میشود و دارای نام اوست.

نام گذاری برای تابع , از قانون نام گذاری برای متغیر ها تبعیت میکند.

لدنه اصلی برنامه , تابعی به نام main() است.تمامی برنامه ها دارای این تابع هستند.

فایل های سرآیند:

کاربردهای پر استفاده از قبل در قالب توابعی آماده شده و به همراه کامپایلر ارائه میشود.بسیاری از اعمال برنامه نویسی با زبان c توسط این توابع از پیش نوشته شده انجام می شوند. برای مثال ورود داده ها از صفحه کلید و نوشتن آن ها در صفحه نمایش و انواع توابع ریاضی پیشرفته و بسیاری موارد دیگر قبلا نوشته شده و آماده شده اند. این توابع و سایر اطلاعاتی که کامپایلر برای ترجمه برنامه ها به آن نیاز دارد , در تعدادی از فایل ها به نام فایل های سرآیند (Header File) قرار دارند.پسوند فایل های سرآیند .h است و معمولا در محلی که کامپایلر نصب شده است , در داخل پوشه ای به نام includeقرار میگیرند. پس نیازمند روشی هستیم تا بتوانیم این فایل ها را به برنامه خود پیوند بزنیم.برای استفاده از این توابع باید از قبل بدانیم که هر تابع در کدام فایل قرار دارد.برای مثال توابع printf() و scanf() در فایل stdio.h قرار دارند.

اتصال فایل های سرآیند:

برای اتصال فایل های سرآیند از دستور #include استفاده میشود.این دستور از دستورات پیش پردازنده بوده و قبل از تابع main() قرار میگیرد.پیش پردازنده مترجمی است که با مشاهده دستوراتی که با # شروع میشوند, اجرا شده و آن ها را به دستورات زبان c تبدیل میکند.این دستور به ; ختم نمیشود. این دستورات به صورت زیر مورد استفاده قرار میگیرند.

#include < نام فایل سر آیند >

چاپ اطلاعات:

زمانی که برنامه های زبان c اجرا میشوند پنجره ای نمایش داده میشود که به آن پنجره خروجی یا نماش گفته میشود.معمولا چاپ اطلاعات با استفاده از تابع printf(...) که در فایل stdio.h قرار دارد انجام میشود. این تابع به دو روش استفاده میشود:

printf (“<عبارت ۱>”) ;

```
#include <stdio.h>
int main()
{
 printf("C is a language.");
 return 0;
}
```

خروجی:

C is a language.

printf (< ۲ عبارت>“ , <عبارت ۱>“) ;

>عبارت ۱ <شامل اطلاعات زیر میباشد:

- اطلاعاتی که باید عینا در خروجی چاپ شوند
- کاراکتر های تعیین کننده فرمت خروجی که با علامت ٪ شروع میشوند
- کاراکتر های کنترلی که با \ شروع میشوند

>عبارت ۲ <اطلاعاتی است که باید به خروجی منتقل شوند که معمولا لیستی از متغیرها میباشند

```
#include <stdio.h>
int main()
{
 int x = 15 ;
 printf("x is %d", x);
 return 0;
}
```

خروجی:

x is 15

کاراکتر	نوع اطلاعاتی که باید به خروجی برود
%c	یک کاراکتر
%d	اعداد صحیح دهدهی مثبت و منفی
%i	اعداد صحیح دهدهی مثبت و منفی
%e	نمایش علمی عدد همراه با حرف e
%E	نمایش علمی عدد همراه با حرف E
%f	عدد اعشاری ممیز شناور
%g	اعداد اعشاری ممیز شناور
%G	اعداد اعشاری ممیز شناور
%o	اعداد مبنای ۸ مثبت
%s	رشته‌ای از کاراکترها (عبارت رشته‌ای)
%u	اعداد صحیح بدون علامت (مثبت)
%x	اعداد مبنای ۱۶ مثبت با حروف کوچک
%X	اعداد مبنای ۱۶ مثبت با حروف بزرگ
%p	pointer (نشانه‌گر)
%n	موجب می‌شود تا تعداد کاراکترهایی که تا قبل از این کاراکتر به خروجی منتقل شده‌اند شمارش شده و در پارامتر متناظر با آن قرار گیرند
%%	علامت ٪

کاراکتر های تعیین کننده فرمت خروجی در:c

این کاراکتر ها نوع اطلاعاتی را که در عبارت ۲ ذکر شده اند و باید به خروجی بروند را مشخص میکنند.برای نمایش اعداد از نوع short و long به ترتیب از کارکتر های h و l ال) به همراه d , i , o و u میتوان به کاربرد.

```
#include <stdio.h>
int main()
{
 int x = 10;
 float y = 15.5;
 char ch = 'a';

 printf(" x=%d, y=%f , ch=%c" , x, y, ch);

 return 0;
}
```

خروجی:

```
x=10, y=15.500000, ch=a
```

کاراکتر	عملی که باید انجام شود
\f	موجب انتقال کنترل به صفحه جدید می‌شود.
\n	موجب انتقال کنترل به خط جدید می‌شود.
\t	انتقال به ۸ محل بعدی صفحه نمایش
\"	چاپ کوتیشن دوبل ("")
\'	چاپ کوتیشن (')
\0	NULL (رشته تهی)
\\	back slash
\v	انتقال کنترل به ۸ سطر بعدی
\N	ثابت‌های مبنای ۸ (N عدد مبنای ۸ است)
\xN	ثابت‌های مبنای ۱۶ (N عدد مبنای ۱۶ است)

کاراکتر های کنترلی در:c

این کاراکتر ها شکل خروجی اطلاعات را مشخص میکند.

```
#include <stdio.h>
int main()
{
 int x = 10;
 float y = 15.5;
 char ch = 'a';

 printf(" x=%d \n y=%f \n ch=%c\n" , x, y, ch);

 return 0;
}
```

خروجی:

```
x=10
y=15.500000
ch=a
```

دستور cin>>:

برای خواندن از ورودی به شکل زیر قابل استفاده است.

cin >> یک متغییر ;

دستور cout>>:

برای چاپ مقادیر در خروجی به شکل زیر قابل استفاده می باشد.

cout>> یک متغییر یا عبارت یا مقدار ثابت ;

حلقه:for

در حالت عادی دستورالعمل های برنامه ه به ترتیب یک بار اجرا میشوند و برنامه خاتمه می یابد. اما در برخی موارد تعدادی از دستورات باید بیش از یک بار اجرا شوند. برای تکرار اجرای دستورالعمل ها از ساختارهای تکرار استفاده میشوند.همچنین در برخی موارد برنامه با توجه به شرایط خاصی مسیر های متفاوتی خواهد داشت.یعنی اجرای تعدادی از دستورالعمل های برنامه وابسته به شرایط مختلف متفاوت خواهد بود.برای بررسی این شرایط و انتخاب دستورالعمل های مناسب از ساختارهای تصمیم استفاده میشود.

ساختار تکرار:for

در این ساختار برای تکرار اجرای دستورات حلقه ای ایجاد میشود و یک یا چند دستور در داخل حلقه قرار میگیرند.معمولا در مواردی که تعداد تکرار حلقه مشخص باشد از for استفاده میشود. در این ساختار از متغیری برای کنترل تعداد حلقه استفاده میشود که آن را شمارنده یا اندیس حلقه تکرار مینامیم.اندیس حلقه دارای یک مقدار اولیه است و در هر بار تکرار حلقه (اجرای دستورات حلقه) مقداری به آن اضافه میشود. این مقدار را پس از هر بار اجرای دستورات حلقه به اندیس حلقه افزوده میشود گام حرکت می گوییم.گام حرکت میتواند عددی صحیح و اعشاری , مثبت یا منفی و کاراکتری باشد.

```
for   (گام حلقه ; شرط حلقه ; مقدار اولیه اندیس حلقه)
{
        دستور ۱ ;
        دستور ۲ ;
        .....
        دستور n ;
}
```

همچنین هر حلقه دارای شرطی است که تعیین میکند حلقه تا چه زمانی باید ادامه داشته باشد که به آن شرط حلقه می گوییم.اگر شرط حلقه دارای ارزش درست باشد حلقه تکرار میشود , اما اگر ارزش شرط حلقه نادرست باشد حلقه به پایان خواهد رسید.

نکته۱:اگر حلقه فقط یک دستور داشته باشد نیازی به **{** و **}** نمی باشد.

نکته۲ : for(; ;) برای ایجاد حلقه بی نهایت مورد استفاده قرار می‌گیرد. حلقه بی نهایت فاقد شرط پایان است پس هیچ وقت متوقف نخواهد شد. در چنین مواقعی توقف اجرای برنامه از کلید های Ctrl + Breakمورد استفاده قرار می گیرند

for (i = 1 ; i <= 10 ; i ++) printf (" * ");	* * * * * * * * * *
for (i = 1 ; i <= 4 ; i ++) printf ("%d \n" , i);	1 2 3 4
for (i = 3.5 ; i <= 5 ; i +=0.5) printf ("%3.1f \n" , i);	3.5 4.0 4.5 5
for (i = 'A' ; i <= 'F' ; i +=2) printf ("%c \n" , i);	A C D F

int x=2, n=5;
for (i = 1 ; i <= n ; i ++)
x=2*x;
printf("%d", x);
	64

برنامه‌ای بنویسید که کاراکترهای 'a' تا 'z' را به همراه کد اسکی آن‌ها چاپ کند.

متغییرها:

ch شمارنده کاراکترها

```
#include <stdio.h>
#include <conio.h>

int main()
{
 char  ch;

 for (ch = 'a' ; ch <= 'z' ; ch++)
     printf("ch = %c, code = %d\n" , ch , ch );

 getch();
 return 0;
}
```

خروجی:

```
ch = a, code = 97
ch = b, code = 98
ch = c, code = 99
ch = d, code = 100
ch = e, code = 101
ch = f, code = 102
. . .
ch = z, code = 122
```

```
for  ( i = 0 ; i <= 2 ; i ++) {

     …..

     for  ( j = 1 ; j < 4 ; j++ )  {
          …..
     }

     …..
}
```

حلقه های تکرار تودرتو:

اگر حلقه تکراری داخل حلقه تکرار دیگری قرار بگیرد, اصطلاحا حلقه های تودرتو گفته می شود.به ازای هر بار تکرار حلقه خارجی , حلقه داخلی به طور کامل اجرا میشود.

```
for  ( i = 1 ; i <= 8 ; i ++) {
      for  ( j = 1 ; j <= i ; j++ )  {
           printf("*");
      }
      printf("\n");
}
```

```
*
**
***
****
*****
******
*******
********
```

عملگر کاما: ,

نکته:عملگر کاما انعطاف پذیری بیشتری به حلقه ی for می بخشد.با استفاده از این عملگر می توان در قسمت های مقدار اولیه و گام حرکت ,دو یا چند عبارت را با هم ترکیب کرد.مثال:

```
for(i=0 , m+=i ; i<10 ; i++ , m++)
```

ساختار تکرار:while :

```
while   (شرط)
{
        دستور ۱ ;
        دستور ۲ ;
        دستور ۳ ;
          ...
          ...
          ...
        دستور n ;
}
```

در مواردی که در ابتدای کار تعداد تکرار های دستورالعمل ها مشخص نمی باشد معمولا از این ساختار استفاده می شود.این ساختار به روش فوق مورد استفاده قرار می گیرد.در صورتی که دستورات تکرار شونده فقط یکی باشد نیازی به **{** و **}** نمی باشد.هنگام اجرای این دستور , اگر شرط حلقه ارزش درستی داشته باشد دستورات حلقه اجرا میشوند وگرنه کنترل برنامه از حلقه خارج می شود.برای اینکه حلقه خاتمه یابد , شرط حلقه باید داخل حلقه نقض شود.یعنی باید شرایطی در داخل حلقه ایجاد شود تا شرط حلقه بعد از مدتی ارزش نادرستی پیدا کرده و حلقه خاتمه یابد.اگر شرط حلقه همیشه درست باشد (هیچ گاه نقض نشود), حلقه تکرار بی نهایت ایجاد می شود.

مثال ۹-۵ : برنامه‌ای بنویسید که جمله‌ای را از ورودی خوانده و تعداد کاراکترهای آن را محاسبه کند. در انتهای جمله Enter وارد می‌شود.(Enter با استفاده از \r قابل تشخیص است.)

متغییرها: count تعداد کاراکترها

```
#include <stdio.h>
#include <conio.h>
 int main()
{
  int  count = 0;
  clrscr();
  printf("Type a statement and ENTER to end: \n");

  while (getche() !='\r')
        count ++ ;

  printf("\nLength of statement is:%d\n", count);
  getch();
  return 0;
}
```

خروجی:

```
Type a statement and ENTER to end:
This is a statement . , ()
Length of statement is:28
```

ساختار تکرار: do ... while :

```
do
{
        دستور ۱ ;
        دستور ۲ ;
        دستور ۳ ;
          ...
          ...
          ...
        دستور n ;
} while   (شرط) ;
```

این ساختار نیز عملکردی همچون حلقه های while دارد با این تفاوت که شرط حلقه while در ابتدای حلقه بررسی میشود اما شرط حلقه do ... while در انتهای حلقه بررسی می شود.دستورات موجود در حلقه do ... while بدون توجه به درست یا نادرست بودن شرط حلقه حداقل یک بار اجرا خواهند شد.در صورتی که که تعداد دستورات تکرار شونده فقط یک دستور باشد نیازی به استفاده از **{** و **}** نمی باشد.در این ساختار نیز اگر شرط حلقه در داخل حلقه نقض نشود , حلقه های بی نهایت ایجاد می شود.

مثال ۱۱-۵ : برنامه‌ای بنویسید که تعدادی عدد را از ورودی خوانده، وارون آن‌ها را به خروجی می‌برد. وارون عددی مثل x، عددی مثل y است به طوری که ارقام عدد x از راست به چپ مثل ارقام عدد y از چپ به راست باشد. به عنوان مثال، وارون عدد ۲۱۵۳، عدد ۳۵۱۲ است.

متغییرها:
num عدد خوانده شده
digit رقمی که در هر مرحله محاسبه می‌شود.

خروجی:

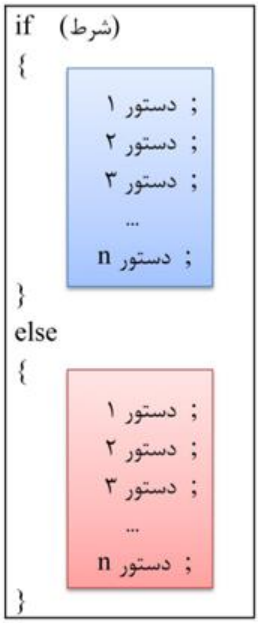
```
Enter a number: 12345
inverse = 54321
Enter a number: 9043567
inverse = 7653409
Enter a number:
```

```
#include <stdio.h>
#include <conio.h>
int main()
{
 int  num, digit = 0;
 while (1){
       printf("\nEnter a number: ");
       scanf("%d", &num);
       printf("\ninverse = ") ;
       do {
            digit = num %10;
            printf("%d", digit);
            num /= 10;
        } while (num != 0);
  }//end of while(1)
 getch();
 return 0;
}
```

از کدام حلقه استفاده کنیم:

معمولا در مواقعی که تعداد دفعات تکرار و گام تکرار نیز مشخص باشد از for استفاده میشود.در سایر موارد و بیشتر در مواردی که پایان تکرار حلقه منوط به رخ دادن شرایطی خاص باشد از while و do ... while استفاده می شود.اگر شرایطی وجود داشته باشد که بدون توجه به شرط حلقه ,دستورات باید حداقل یکبار اجرا شوند در این شرایط از do...while استفاده می کنیم.

ساختار شرطیif :



در این ساختار ابتدا شرطی چک می شود و در صورتی که ارزش شرط درست باشد مجموعه ای از دستورات اجرا خواند شد.در صورتی که ارزش شرط نادرست باشد مجموعه ای دیگر از دستورات اجرا خواهند شد.در صورتی که هر کدام از مجموعه دستورات شامل فقط یک دستور باشند ,استفاده از **{** و **}** مورد نیاز نمی باشد.در صورتی که ارزش شرط درست باشد دستورات بعد از if اجرا می شوند و سپس اجرا از ساختار if خارج می شود, اگر ارزش شرط نادرست باشد فقط دستورات بخش else اجرا می شود و سپس اجرا از ساختار if خارج می‌شود. این ساختار می تواند فاقد بخش else نیز باشد.

برنامه‌ای بنویسید که دو عدد صحیح را از ورودی دریافت کرده و عدد بزرگتر را در خروجی چاپ ک

متغییرها:
num1 عدد اول
num2 عدد دوم
max عدد بزرگتر

خروجی:

```
Enter two numbers:34 65
Max of numbers is : 65
```

```
ers:");
&num2);

is : %d" , max);
```

```
if   (شرط ۱)
  {
        دستور ۱ ;
          ...
        دستور n ;
  }
else if (شرط ۲)
  {
        دستور ۱ ;
          ...
        دستور n ;
  }
else if (شرط ۳)
  {
        دستور ۱ ;
          ...
        دستور n ;
  }
```

ساختار: else if

در استفاده از ساختار if براب بررسی کردن شرط های متعدد باید آن ها را به صورت تودرتو استفاده کنیم.کاربرد ifبه صورت تودرتو , نه تنها موجب طولانی شدن برنامه می شود, بلکه از خوانایی برنامه نیز خواهد کاست.ساختار else ifمی تواند جایگزین چندین if تودرتو شود.

برنامه‌ای بنویسید که روز هفته را گرفته و نام آن را چاپ کند.

```
#include <stdio.h>
int main()
{
 int n;
 printf("Enter number of day between 1 – 7: ");
 scanf("%d", &n);
 if (n ==1)  printf("\n Saturday");
 if (n ==2)  printf("\n Sunday");
 if (n ==3)  printf("\n Monday");
 if (n ==4)  printf("\n Tuesday");
 if (n ==5)  printf("\n Wednesday");
 if (n ==6)  printf("\n Thursday");
 if (n ==7)  printf("\n Friday");
 getch();
 return 0;
}
```

```
#include <stdio.h>
int main()
{
 int n;
 printf("Enter number of day between 1 - 7: ");
 scanf("%d", &n);
 if (n ==1)  printf("\n Saturday");
 else if (n ==2)  printf("\n Sunday");
 else if (n ==3)  printf("\n Monday");
 else if (n ==4)  printf("\n Tuesday");
 else if (n ==5)  printf("\n Wednesday");
 else if (n ==6)  printf("\n Thursday");
 else if (n ==7)  printf("\n Friday");
 else printf("\n Error");
 getch();
 return 0;
}
```

انتقال کنترل غیر شرطی:

دستور کنترل شرطی if بعد از بررسی کردن شرط , با توجه به نتیجه شرط اقدام به اجرای دستورالعمل ها می کند.برخی دستورات دیگر بدون بررسی کردن هیچ شرطی قادر به انتقال کنترل برنامه از نقطه ای به نقطه دیگر هستند.(break ; continue ; goto).

دستور **break**موجب خروج از حلقه تکرار می شود.نحوه استفاده از این دستور به این صورت است . **break;** : اگر چندین حلقه تودرتو وجود داشته باشد , این دستور موجب خروج از حلقه ای که در آن است می شود.برای خاتمه دادن به ساختار switch نیز از این دستور استفاده می شود.

دستور **continue** در حلقه تکرار موجب انتقال کنترل به ابتدای حلقه می شود.پس از انتقال کنترل به ابتدای حلقه , شرط حلقه مورد بررسی قرار می گیرد.چنانچه شرط درست باشد , اجرای دستورات حلقه ادامه می یابد وگرنه حلقه تکرار خاتمه می یابد.استفاده از این دستور به این صورت است. **continue**; :

دستور **goto** سبب انتقال کنترل از نقطه به نقطه دیگر برنامه می شود.این دستور معمولا به ندرت مورد استفاده قرار می گیرد.چون استفاده از آن خوانایی برنامه را بسیار کاهش می دهد.نحوه استفاده از آن به این صورت است**<goto .برچسب ;>**:برچسب دستور همانند متغیرها نامگذاری می شود و به **(;)**ختم می شود.انتقال کنترل توسط goto فقط داخل یک تابع امکان پذیر است.

```
برچسب :                    int  i = 0 ;
دستور ۱ ;                  lable1:
دستور ۲ ;                  printf("%d\n" , i );
دستور ۳ ;                  i ++ ;
. . . .                    if ( i < 10)
دستور n ;                    goto lable1;
goto برچسب ;               printf("finish") ;
```

ساختار تصمیم switch :

```
switch (عبارت) {
    case <مقدار ۱>:
        <دستورات ۱>
        break ;
    case <مقدار ۲>:
        <دستورات ۲>
        break ;
        . . .
        . . .
        . . .
    default :
        <دستورات n>
}
```

از این ساختار برای تصمیم گیری های چندگانه بر اساس مقادیر مختلف یک عبارت استفاده می شود.معمولا در تمام تصمیم گیری هایی که بیش از سه انتخاب وجود داشته باشد بهتر است از ساختار switch استفاده شود.این ساختار به صورت زیر مورد استفاده قرار می گیرد:

- ابتدا عبارت مقابل switch به مقدار صحیح ارزیابی می شود و مقدار آن تعیین می شود.
- اگر مقدار عبارت برابر با >مقدار ۱ <باشد >دستورات ۱ <اجرا می شوند و اجرای دستورات تا رسیدن به breakادامه خواهد یاقت.
- دستور break اجرای برنامه را از ساختار switch خارج می سازد.
- در صورتی که مقدار غبارت با >مقدار ۱ <برابر نباشد با >مقدار ۲ <مقایسه می شود و همین روند ادامه می یابد.
- تا زمانی که مقدار عبارت برابر با یکی از مقادیر نباشد عمل مقایسه ادامه می یابد.
- اگر مقدار عبارت با هیچ کدام از مقادیر برابر نباشد , دستورات بخش default به اجرا در می آید.

برنامه‌ای بنویسید که روز هفته را گرفته و نام آن را چاپ کند.

```
#include <stdio.h>
int main()
{
 int n;
 printf("Enter number of day between 1 – 7: ");
 scanf("%d", &n);
 if (n ==1)  printf("\n Saturday");
 if (n ==2)  printf("\n Sunday");
 if (n ==3)  printf("\n Monday");
 if (n ==4)  printf("\n Tuesday");
 if (n ==5)  printf("\n Wednesday");
 if (n ==6)  printf("\n Thursday");
 if (n ==7)  printf("\n Friday");
 getch();
 return 0;
}
```

```
#include <stdio.h>
int main()
{  int n;
 printf("Enter number of day between 1 - 7: ");
 scanf("%d", &n);
 switch (n)  {
     case 1:  printf("\n Saturday"); break;
     case 2:  printf("\n Sunday"); break;
     case 3:  printf("\n Monday"); break;
     case 4:  printf("\n Tuesday"); break;
     case 5:  printf("\n Wednesday"); break;
     case 6:  printf("\n Thursday"); break;
     case 7:  printf("\n Friday"); break;

     default : printf("\n Error");       }
 getch();
 return 0;
}
```

نکته:

- ساختار switch می تواند فاقد بخش default باشد.در این صورت اگر مقدار عبارت با هیچ یک از مقادیر برابر نباشد , هیچ کدام از دستورات داخل switch اجرا نخواهد شد.
- مقادیر موجود در case ها نمی توانند با هم مساوی باشند. یعنی هیچ کدام از مقادیر >مقدار ۱ , < >مقدار ۲ <و...نباید مساوی باشند.
- اگر در یک case از دستور break استفاده نشود , با case بعدی or می شود.برای اینکه دو یا چند شرط را در ساختار switch با همدیگر or کنیم , آن ها را بدون break پشت سر هم می نویسیم.
- یکی از تفاوت های if و switch در این است که در ساختار if می توان عبارت منطقی یا رابطه ای را مورد بررسی قرار داد ولی در ساختار switch فقط مساوی بودن مقادیر مورد بررسی قرار می گیرد.
- چند ساختار switch را می توان به صورت تودرتو مورد استفاده قرار داد.

آرایه های یک بعدی:

در زبان c با استفاده از متغیرها و ثوابت می توانیم داده ها را ذخیره کرده و در صورت لزوم از آن ها استفاده کنیم.در برخی موارد نیازمند ذخبره تعداد زیادی داده هستیم.نمرات و معدل دانشجویان یک کلاس ,حقوق کارمندان یک شرکت, موجودی کالاهای یک انبار و بسیاری موارد دیگر از جمله مواردی هستند که نیازمند استفاده از تعداد زیادی متغیر برای ذخیره و بازیابی اطلاعات هستند. در این موارد تعریف و به کار گیری متغیر ها به روش معمول ,کاری بسیار طاقت فرسا و پیچیده است. لذا از مجموعه ای از متغیر ها با نام آرایه استفاده می کنیم.آرایه مجموعه ای از عناصر هم نوع است.هر آرایه دارای نامی است که مانند متغیر های معمولی نام گذاری می شود.برای دسترسی به عناصر آرایه از متغیری به نام اندیس استفاده می شود.به همین دلیل آرایه را نتغیر اندیس دار نیز می گویند.

آرایه یک بعدی:این آرایه ها را لیست نیز می نامند و به صورت زیر تعریف می شوند:

نوع آرایه نام آرایه [طول آرایه] ;

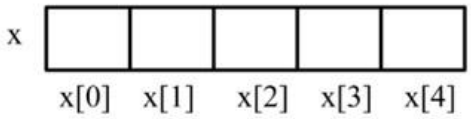
- نوع آرایه یکی از انواع قابل قبول در c است.
- نام آرایه ,برای دسترسی به عناصر آرایه مورد استفاده قرار می گیرد.
- طول آرایه با یک عدد صحیح مثبت و در همان ابتدای تعریف آرایه مشخص می شود.

مثال: int a [5] ;

- اندیس آرایه ها در زبان c از صفر شروع می شود.
- اندیس آرایه ای به طول n از صفر شروع شده به n-1 ختم می شود.
- عناصر آرایه در محل های متوالی از حافظه ذخیره می شوند.

مثال :

```
int x[5];
```



- میزان حافظه ای که در آرایه اختصاص داده می شود از رابطه ی زیر قابل محاسبه است:

) طول آرایه) * (طول نوع آرایه) = میزان حافظه آرایه

- برای مثاب اگر طول هر int را ۴ بایت در نطر بگیریم, میزان حافظه ای که آرایه x اشغال می کند , برابر با ۲۰=۵*۴ بایت خواهد بود.

```
printf ("%d", x[2]);

x[0] = 23;
```

```
int x[5] , i;
for (i=0;i<5;i++)
   scanf("%d", &x[i]);
```

```
for (i=0;i<5;i++)
   printf("%d\n", x[i]);
```

```
for (i=4;i>=0;i--)
   printf("%d\n", x[i]);
```

- با استفاده از اندیس آرایه می توان به عناصر آرایه دسترسی پیدا کرد.
- دستور رو به رو مقدار موجود در خانه ی سوم ارایه x را چاپ می کند.
- دستور انتساب رو به رو مقدار ۲۳ را در عنصر اول آرایه x قرار می دهد..
- برای استفاده راحت تر از آرایه ها معمولا از حلقه های تکرار استفاده می کنند.

مثال : برنامه‌ای که ۵ عدد را گرفته و به ترتیب در آرایه‌ای ذخیره می‌کند. آرایه را بصورت معکوس چاپ می‌کند.

```
#include <stdio.h>
int main()
{
 int x[5], i;

 printf("Enter 5 numbers:\n");
 for (i=0;i<=4;i++)
    scanf("%d",&x[i]);

 printf("Inverse of your numbers:\n");
 for (i=4;i>=0;i--)
    printf("%d ", x[i]);

 getch();
 return 0;
}
```

```
Enter 5 numbers:
12 0 93 45 66
Inverse of your numbers:
66 45 93 0 12
```

آرایه یک بعدی به عنوان آرگومان تابع

- برای ارسال آرایه به تابع ,باید نام آرایه به عنوان آرگومان ذکر شود.
- اگر آرایه به عنوان آرگومان تابع باشد, پارامتر معادل آن می تواند به سه صورت تعریف شود.

۱. آرایه ای به طول مشخص

۲. آرایه ای با طول نا مشخص که در این صورت بهتر است طول آرایه به عنوان آرگومانی دیگر به تابع ارسال شود

۳. اشاره گر.

```
void   func1 (int x[]);
void   func2 (int x[], int len);
int    main()
{
int   x[10];
. . . .
func1(x);
. . . .
func2(x, 10);
return  0;
}
void   func1 (int x[10])
{
   . . . .
}
void   func2 (int x[], int len)
{
   . . . .
}
```

آرایه های دو بعدی درc

علاوه بر لیست ها (آرایه های یک بعدی) ,می توان آرایه های نعریف کرد که بیش از یک بعد داشته باشد .برای مثال ,جدول ضرب و ماتریس آرایه ای دو بغدی است که یک بعد آن سطر و بعد دیگر آن ستون است. در آرایه های دو بعدی ,برای دستیابی به عناصر هر آرایه ,از دو اندیس استفاده می شود که اندیس اول را سطر و اندیس دوم را ستون می نامند. آرایه هایی با بیش از دو بعد (آرایه های nبعدی) قابل اسفاده اند. آرایه های دو بعدی در cبه صورت زیر تعریف می شود:

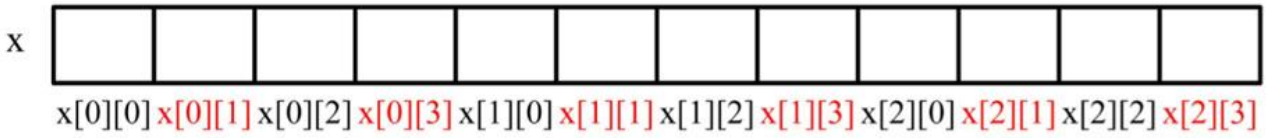
نوع آرایه نام آرایه [اندازه بعد ۱] [اندازه بعد ۲] ;

هر کدام از این اندیس ها از صفر شروع می شوند.

مثال:

```
int  x[3][4];
```

- آرایه ها در حافظه به صورت سطری و ستونی می توانند ذخیره شوند.
- در زبان C آرایه ها به صورت سطری ذخیره می شوند.یعنی ابتدا عناصر سطر اول ,سپس عناصر سطر دوم, عناصر سطر سوم و... ذخیره می شوند.
- برای مثال قبل می توان آرایه را در حافظه به این صورت نمایش داد:



- مثال : برنامه‌ای بنویسید که جدول ضرب اعداد را با استفاده از آرایه‌های دو بعدی تولید کرده و نمایش دهد.

```
#include <stdio.h>
int main()
{
 int table[10][10], i , j;
 for (i=0;i<10;i++)
    for (j=0;j<10;j++)
       table[i][j]=(i+1)*(j+1);
 for (i=0;i<10;i++)
  {for (j=0;j<10;j++)
       printf("%4d", table[i][j]);
   printf("\n");}
 getch();
 return 0;
}
```

```
   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
```

آرایه های دو بعدی به عنوان آرگومان تابع

- برای ارسال آرایه های دو بعدی از تابعی به تابع دیگر باید نام آرایه را به عنوان آرگومان تابع ذکر کرد.
- برای تعریف پارامتر معادل آن ,مانند آرایه های یک بعدی عمل می شود.با این تفاوت که در حالتی که در آرایه های یک بعدی ,پارامتر به صورت آرایه بدون طول ذکر می شد.در آرایع دو بعدی ,طول سطر

ذکر نمی شود ولی طول ستون ها حتما باید ذکر شود که در این صورت بهتر است طول سطر ,به عنوان آرگومان دیگری به تابع ارسال شود.

- به هنگام ارسال آرایه به عنوان پارامتر در آرایه های چند بعدی نیز تنها می توان اندیس اول را حذف کرد و عدد مربوط به بقیه اندیس ها صراحتا ذکر شود.

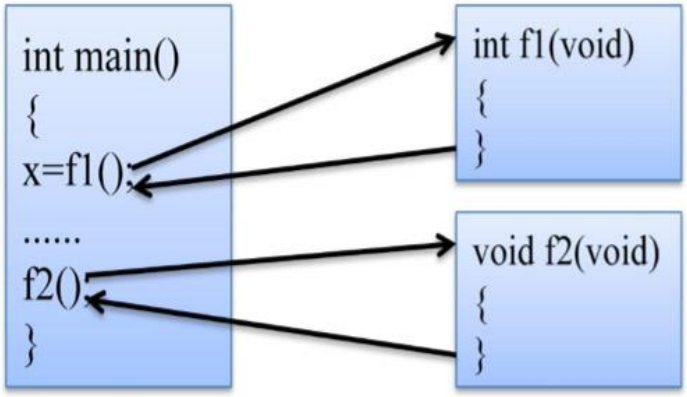
```
void    func1 (int x[5][10]);
void    func2 (int x[][10], int len);
int     main()
{
int    x[5][10];
. . . .
func1(x);
. . . .
func2(x, 5);
return   0;
}
void    func1 (int x[5][10])
{
   . . . .
}
void    func2 (int x[][10], int len)
{
   . . . .
}
```

```
#include <stdio.h>
void printTable(int table[][10]);
int main()
{
  int table[10][10], i , j;
  for (i=0;i<10;i++)
     for (j=0;j<10;j++)
        table[i][j]=(i+1)*(j+1);

  printTable(table);
  getch();
  return 0;
}
```

```
void printTable(int table[][10])
{
  int i,j;
  for (i=0;i<10;i++)
    {
     for (j=0;j<10;j++)
        printf("%4d", table[i][j]);
     printf("\n");
    }
}
```

تابع (زیربرنامه)

مقدمه:دستورالعمل های نوشته شده در تابع main() هنگام ترجمه و اجرای برنامه ها مورد استفاده قرار می گیرند.در برخی موارد به دلیل پیچیدگی و حجم بالای کد های نوشته شده ذر برنامه ها, بهتر است که کدهارا قطعه قطعه کنیم تا هر بخش از کد ها , بخشی از کل عملیات برنامه را انجام دهد.این بخش های منطقی و مستقل از هم را تابع می نامیم.برخی از توابع که در اغلب برنامه ها مورد استفاده قرار می گیرند ,از قبل نوشته شده و آماده اند.این توابع را توابع کتابخانه ای می نامیم.توابعی همچون clrscr, ()cos , ()sin() و بسیاری موارد دیگر در فایل های سرآیندخاصی قرار دارند که در صورت لزوم می توانیم از آن ها استفاده کنیم.با استفاده از توابع می توان برنامه های ساخت یافته نوشت.در این نوع برنامه ها هر قسمت از عملیات مربوط به برنامه توسط توسط بخش مستقلی انجام می شود که به آن تابع گفته می شود.

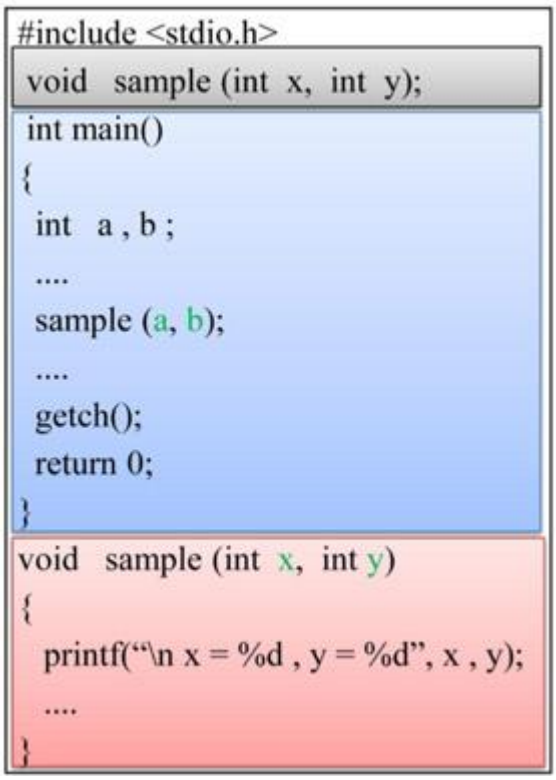


برنامه نویسی ساخت یافته:

از مزایای برنامه نویسی ساخت یافته می توان به موارد زیر اشاره کرد:

نوشتن برنام های ساخت یافته آسان تر است.چون برنامه های پیچیده به بخش های کوچکتری تقسیم می شوند. همکاری بین افراد را فراهم می کند , چون بخش های مختلف را می توان توسط افراد مختلف انجام داد . اشکال زدایی برنامه های ساخت یافته آسان تر است, چون پیدا کردن اشکال در بخش های کوچکتر به صورت ساده تری قابل انجام است. برنامه نویسی ساخت یافته موجب صرفه جویی در وقت می شود. میتوان از توابع نوشته شده در برنامه های دیگری نیز استفاده کرد .برای تهیه ی برنامه های ساخت یافته , ابتدا باید برنامه را به

بخش های مستقل از هم تقسیم کرد .سپس برای هر بخش ورودی و خروجی و ساختار تابع را تعریف کرد.برای این کار اهداف تابع نیز باید مشخص گردد.



```
#include <stdio.h>
void   sample (int  x,  int  y);
 int main()
{
 int   a , b ;
 ....
 sample (a, b);
 ....
 getch();
 return 0;
}
void   sample (int  x,  int y)
{
  printf("\n x = %d , y = %d", x , y);
  ....
}
```

بخش های اصلی توابع:

تعریف تابع:مجموعه ای از دستورات است که عملکرد تابع را مشخص می کند.

احضار یا فراخوانی تابع: دستوری است که برای اجرای تابع مورد استفاده قرار می گیرد.توابع با استفاده از نام تابع فراخوانی می شوند.

اعلان الگوی تابع : نوع تابع , نام تابع و تعداد و نوع پارامترهای تابع را به کامپایلر معرفی میکند.

تعریف تابع

```
<نوع تابع>  نام تابع  (لیست پارامترها)
{
        دستور ۱ ;
        دستور ۲ ;
        .....
        دستور n ;
}
```

مثال

```
int abs(int a)
{
 if (a<=0)
      a = a *(-1) ;
 return a;
}
```

تعریف توابع:

نوع تابع: یکی از انواع . void , char , double , float , int یکی از انواع تعریف شده توسط کاربر.

عنوان تابع : همانند قوانین نامگذاری متغیرها . وسیله ای برای دسترسی و استفاده از تابع.

پارامتر های تابع: متغیر هایی برای انتقال مقادیر از برنامه اصلی به یک تابع.

بدنه تابع : مجموعه دستوراتی که عملیات مربوط به تابع را انجام می دهند.

نکته: تعریف تابع در داخل تابع دیگر مجاز نمی باشد.

نوع تابع:

نوع تابع ,یکی از انواع موجود در زبان **c**یا انواع تعریف شده توسط کاربر است . اگر تابع بخواهد مقداری را به تابع فراخوان برگرداند, آن مقدار در نام تابع قرار می گیرد. چون هر مقداری در زبان c دارای نوع است . لذا تابع نیز دارای نوع است.

int x;
x= func1() + 12;
	int func1(void)
	{
	.....
	return 10;
	}

اگر تابع هیچ مقداری را به برنامه فراخوان برنگرداند, نوع تابع void خواهد بود.

void f1(void)
{
.....
}
	void f2(int x)
	{
	.....
	}

پارامترهای تابع:

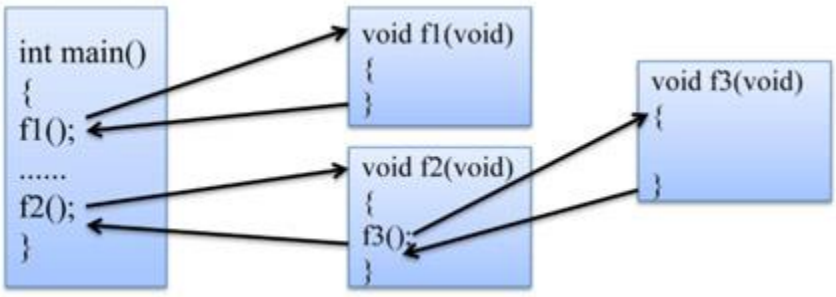
پارامتر ها اطلاعاتی هستند که بین تابع فراخوان و تابع فراخوانی شونده هستند.هنگام فراخوانی تابع , پارامتر ها از برنامه فراخوان به تابع ارسال می شوند. اگر تعداد پارامتر ها بیش از یک عدد باشد با کاما از هم جدا شوند .در لیست پارامتر ها ,نوع هر یک از پارامترها نیز مشخص می شود.پارامتر ها متغیر هایی هستند که هنگام تعریف تابع ,جلوی نام تابع و در داخل پرانتز قرار گرفته و تعریف می شوند. پارامترها به عنوان متغیری در داخل تابع قابل استفاده هستند. اگر تابعی آرگومان است ,به جایی لیست آرگومان ها , کلمه voidرا قرار دهید.

int f1(void)
{
.....
return 10;
}
	int f2(int x)
	{
	x++;
	return x;
	}

int f3(int x, int y)
{
return x+y;
}
	float f4(int x, float y)
	{
	y = y*x;
	return y;
	}

فراخوانی توابع:

- برای شروع اجرای یک تابع , با استفاده از نام تابع آن را فراخوانی می کنیم.
- اطلاعاتی هنگام فراخوانی تابع در جلوی نام تابع (در داخل پرانتز) قرار می گیرند ,آرگومان تابع نامیده می شوند.
- هنگام فراخوانی توابع دقت داشته باشید که تعداد و نوع پارامترها و آرگومان ها یکسان باشند.
- تمامی توابع در داخل یکدیگر قابل فراخوانی هستند.



الگوی تابع:

الگوی تابع مشخص می کند که تابع چگونه باید فراخوانی شود

) ;لیست پارامترها) نام تابع> نوع تابع<

- برای بکار گیری تابع در برنامه ,باید الگوی آن را در خارج از تابع main() به کامپایلر اعلان کرد.به این کار اعلان الگوی تابع گفته می شود.
- اگر تعریف توابع قبل از تابع main() باشد,دیگر نیازی به اعلان تابع نیست.
- اگر تابع بعد از تابع main() تعریف شود,باید الگوی آن قبل از تابع main() قرار داده شود.
- بهتر است الگوی تمامی توابع را قبل از تابع main()اعلان کنید.حتی می توانید در داخل تابع main()نیز قبل از استفاده از تابع آن را اعلان کنید.

نکته:هیچ تابعی نمی تواند از متغیرهای سایر توابع استفاده کند,مگر اینکه از طریق پارامتر ها منتقل شوند.

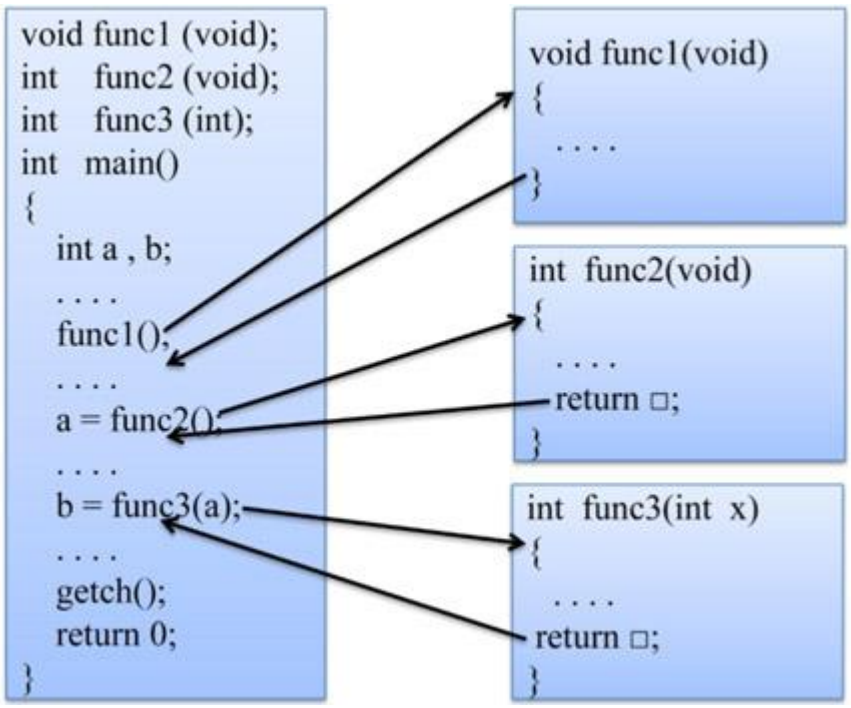
محل اعلان الگوی تابع:

```
#include <stdio.h>
void   f(int  x,  int  y);
  یا
void   f(int,  int);

 int main()
{
  ....
  f(a, b);
  ....
}

void   f(int  x,  int y)
{ ....}
```

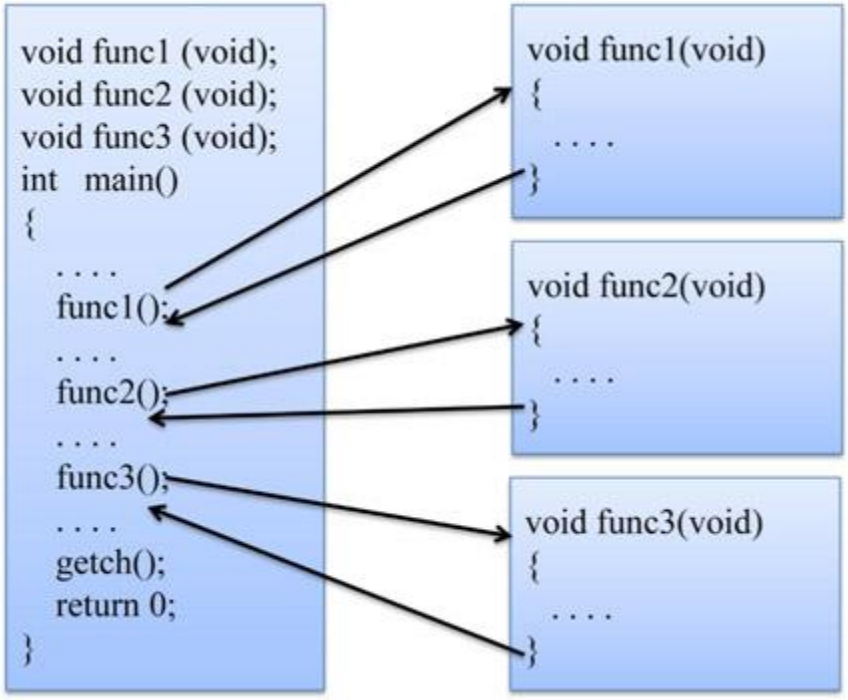
```
#include <stdio.h>
 int main()
{
void   f(int  x,  int  y);
  یا
void   f(int,  int);
  ....
  f(a, b);
  ....
}

void   f(int  x,  int y)
{ ....}
```

```
#include <stdio.h>

 void  f(int x,  int y)
{ ....}

 int main()
{
  ....
  f(a, b);
  ....
}
```

عملکرد توابع ۲:



عملکرد توابع ۱:



نکته:

نحوه تهیه توابع:

ابتدا بدون توجه به جزئیات پیاده سازی توابع,آرگومان ها و نتیجه ای را که از توابع انتظار دارید,مشخص کنید.به عبارت دیگر ,در قدم اول لازم نیست به جزئیات پیاده سازی تابع بپردازید.تنها مشخص کردن ورودی و خروجی تابع و چگونگی عملکرد آن اهمیت دارد.با توجه به فرضیات بالا برنامه اصلی را بنویسید.پس از نوشتن برنامه اصلی ,تعریف توابع دیگر را بنویسید.توابع را طوری طراحی و پیاده سازی کنید که هر تابع تنها به آنچه نیاز دارد دسترسی داشته باشد.برای ارتباط بین توابع از آرگومان ها و پارامترها استفاده کنید.

روش های ارسال پارامتر به توابع:

روش فراخوانی با مقدار:در روش فراخوانی با مقدار ,هنگام فراخوانی, مقادیر آرگومان ها در پارامترها کپی می شوند و هر گونه تغییری در پارامترها ,تاثیری در آرگومان ها ندارد.

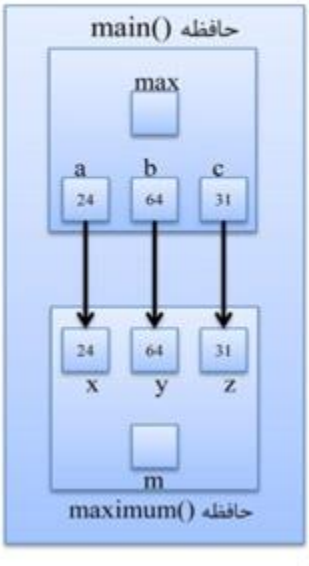
روش فراخوانی با ارجاع:در روش فراخوانی با ارجاع ,آدرس آرگومان ها به پارامترها متنقل می شود.بنابر این پارامترها باید قابلیت نگه داری آدرس را داشته باشند.این بحث مربوط به اشاره گرها بوده و در بخش مربوطه مطرح خواهد شد.

برنامه‌ای بنویسید که سه مقدار صحیح را از ورودی خوانده به تابعی ارسال می‌کند. تابع بزرگترین مقدار را از بین سه مقدار پیدا کرده و به خروجی می‌برد.

```
#include <stdio.h>

int maximum(int,int,int);

int main()
{
 int a , b , c , max;
 printf("enter 3 integer numbers:");
 scanf("%d%d%d", &a, &b, &c);

 max = maximum(a, b, c);
 printf("Maximum is: %d",max);

 getch();
 return 0;
}
```

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

```
enter 3 integer numbers:24 64 31
Maximum is: 64
```



انواع توابع:

توابع از نظر تعداد مقادیری که بر می گردانند به سه دسته تقسیم می شوند.

۱. توابعی که هیچ مقدار برنمی گردانند.(نوع void بر می گردانند**void f1 (....) ;**)
۲. توابعی که یک مقدار بر می گردانند.
۳. توابعی که چند مقدار بر می گردانند(با استفاده از اشاره گر ها(

توابعی که هیچ مقدار را بر نمی گردانند

توابعی هستند که عملیاتی را انجام داده و خروجی های مورد نظر را چاپ کرده و هیچ مقداری را به تابع فراخوان تحویل نمی دهد. همچنین ممکن است متغیر های عمومی را تغییر دهند.

مثال: برنامه‌ای که با استفاده از تابعی، زمانی را برحسب ساعت، دقیقه و ثانیه خوانده، زمان را برحسب ثانیه چاپ می‌کند.

```
#include <stdio.h>
void convert(void);
int main()
{
 convert();

 getch();
 return 0;
}
```

```
void convert(void)
{
   int hours, minutes,seconds;
   long int time;

   printf("enter time :(Hour, Minute, Second:");
   scanf("%d%d%d", &hours, &minutes, &seconds);
   time = 3600*hours+60*minutes+seconds;
   printf("\ntime is :%ld",time);
}
```

```
enter time :(Hour, Minute, Second:2 35 56
time is :9356 seconds
```

توابعی که یک مقدار را برمی گردانند:

بسیاری از توابعی که مورد استفاده قرار می گیرند بعد از انجام محاسباتی خاص,مقداری را به عنوان نتیجه باز می گردانند. مثلا تابعی مانند sin() اندازه زاویه را برمی گرداند.برای نوشتن اینگونه توابع ,نوع آن ها را باید در الگوی تابع و عنوان تابع مشخص کرد.برای برگرداندن مقداری توسط تابع از دستور return به صورت رو به رو استفاده می شود**<عبارت return);**

نوشتن پرانتز در دستور return الزامی نمی باشد.مقداری که توسط دستور returnبرگشت داده می شود ,در نام تابع قرار می گیرد.پس در برنامه فراخوان می توان از نام تابع به عنوان یک عبارت یا مقدار استفاده کرد.

برنامه‌ای که شعاع دایره‌ای را از ورودی خوانده به تابعی ارسال می‌کند و تابع مساحت دایره را محاسبه کرده و به برنامه اصلی برمی‌گرداند.

```
#include <stdio.h>
float area(float);
int main()
{
 float r ,s;
 printf("Enter the radius:");
 scanf("%f", &r);
 s= area(r);
 printf("\nArea is:%5.2f",s);
 getch();
 return 0;
}
```

```
float area(float r)
{
 float s;
 s= r*r*3.14;
 return s;
}
```

```
Enter the radius:5

Area is:78.50
```

متغیر های عمومی و محلی:

---

۱. متغیرهایی که در داخل یک تابع تعریف می شوند ,متغیرهای محلی می نامیم و تنها در داخل همان تلبع قابل استفاده اند .یعنی اصطلاحا حوزه (scope)این متغیر ها فقط در همان تابعی است که در آن تعریف می شوند.

۲. اگر متغیرها در خارج از توابع و در بالای تابع main() تعریف شوند,در تمامی توابع موجود در برنامه قابل استفاده اند و متغیرهای عمومی نامیده می شوند.

نکته:

- متغیرهای عمومی درک و نگه داری برنامه ها را مشکل می کنند.بنابراین توصیه می شود حتی الامکان از متغیرهای عمومی استفاده نشود
- استفاده از ثوابت عمومی به دلیل عدم امکان تغییر ثوابت توسط توابع,اثرات جانبی چندانی نمی تواند داشته باشد.لذا کاربرد ثوابت عمومی معمول تر از متغیرهای عمومی است.

- متغیر های محلی تا زمانی که مقدار نگرفته باشند دارای مقادیر معتبری نیستند ولی متغیر های عمومی دارای مقدار اولیه صفر هستند.
- اگر در تابعی متغیر محلی همنام با متغیر عمومی تعریف شود,در آن تابع ,آن متغیر عمومی دیگر قابل استفاده نیست و از متغیرهای محلی استفاده خواهد شد.

```
#include <stdio.h>
int x;
void func1(void);
void func2(void);
int main()
{
 printf("\nin main, first value of x is : %d", x);
 x=100;
 func1();
 func2();
 printf("\nin main, value of x is : %d", x);
 getch();
 return 0;
}
```

```
void func2(void)
{
    int x;
    printf("\nin func2 value of x changes: ");
    for (x = 1; x < 6 ; x++)
       printf("x=%d  ", x);
}
//*********************************
void func1(void)
{
 printf("\nin func1 value of x is : %d", x);
}
```

```
in main, first value of x is : 0
in func1 value of x is : 100
in func2 value of x changes: x=1  x=2  x=3  x=4  x=5
in main, value of x is : 100
```

توابع بازگشتی:

توابعی که خود را فراخوانی می کنند توابع بازگشتی می نامند.توابع به صورت مستقیم و غیر مستقیم می توانند خودشان را فراخوانی کنند.در روش مستقیم تابع در یکی از دستوراتش خودش را فراخوانی می کند.در روش غیرمستقیم تابع () f2 تابع() f3را فراخوانی می کند و تابع ()f3 تابع ()f2() را فراخوانی می کند.

```
void f1(void)
{
f1();
}
```

```
void f2(void)
{
f3();
}
```

```
void f3(void)
{
f2();
}
```

برای ایجاد تابع بازگشتی الگوریتمی که توسط تابع پیاده سازی می شود باید دارای خاصیت بازگشتی باشد.طرح کلی بازگشتی به صورت زیر می باشد:

- یک یا چند حالت که در آن ,تابع, وظیفه اش را به صورت بازگشتی انجام می دهد.یعنی این حالت ها خاصیت بازگشتی دارند.
- یک یا چند حالت که در آن تابع وظیفه خودش را بدون فراخوانی بازگشتی انجام می دهد.این حالت را حالت توقف گویند.

اغلب با استفاده از یک دستور if مشخص می شود که گدام یک از حالت ها باید انجام شوند.برای اینکه فراخوانی های بازگشتی به اتمام برسند باید حالت توقف اتفاق بیافتد.یعنی هر فراخوانی تابع,سرانجام باید به حالت توقف ختم شود.در غیر این صورت فراخوانی تابع خاتمه نمی یابد.

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

```
if   (به حالت توقف رسیدی)
        مسأله حالت توقف را حل کن.
else
        مسأله را بار دیگر فراخوانی کن.
```

```
int fact(int n)
{
  if (n==0)     return 1;
  else   return (n*fact(n-1));
}
```

محاسبه فاکتوریل با استفاده از تابع بازگشتی

```
#include <stdio.h>
int fact(int);
int main()
{
   int n,f;
   printf("Enter your number:");
   scanf("%d", &n);
   f=fact(n);
   printf("%d! = %d", n, f);

   getch();
   return 0;
}
```

```
int fact(int n)
{
  if (n==0)
     return 1;
  else
      return (n*fact(n-1));
}
```

```
Enter your number:5
5! = 120
```

رشته ها درc

مقدمه:

در زبان c رشته به صورت آرایه ای از کاراکتر ها تعریف می شوند. رشته ها برای ذخیره ,بازیابی ودستکاری متن ها مورد استفاده قرار می گیرند.برای تعیین انتهای رشته از کاراکتر خاصی به نام (تهی **NULL**) =استفاده می شود که با **'\0'**مشخص می گردد.بنابراین ,آخرین کرارکتر در هر رشته برابر با **'\0'**می باشد.لذا اگر رشته ای به طول n تعریف کنیم ,فقط از n-1 کاراکتر آن می توانیم استفاده کنیم, زیرا کاراکتر آخر '\0' است.مثال مثال :

char s[10];

اگر محتویات این رشته برابر با "sahand" قرار گیرد, این رشته به صورت زیر نمایش داده می شود:

s

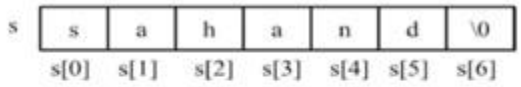
s	a	h	a	n	d	\0	?	?	?
s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]

مقداردهی اولیه به رشته ها

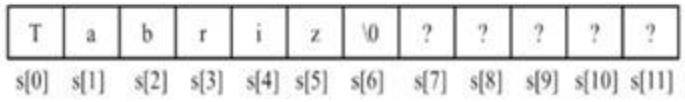
هنگام تعریف رشته ها می توان به آن ها مقدار اولیه داد.هنگام مقدار دهی اولیه می توان طول رشته را مشخص نکرد .در این صورت ,اندازه آرایه یک واحد بیشتر از تعداد کاراکترهایی است که به آن نسبت داده می شود.دو روش برای مقدار دهی به رشته ها وجود دارد:

روش اول :رشته در داخل کوتشن قرار گیرد و به متغیر رشته ای (آرایه رشته ای) نسبت داده شود:

char s[] = “sahand”;

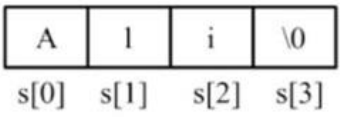


char s[12] = “Tabriz”;



روش دوم :هر یک از کاراکترهای رشته ای به عنوان یک عنصر رشته به آرایه نسبت داده شوند:

char s [] = {‘A’ , ‘l’ , ‘i’ , ‘\0’};



ورودی خروجی رشته ها درc

برای ورودی و خروجی رشته ها می توان از توابع scanf() و printf() استفاده کرد.نباید از عملگر آدرس **&**قبل از نام آرایه استفاده کرد. زیرا نام آرایه خود نشان دهنده ی آدرس است.**نکته :**خواندن scanf() تا رسیدن به اولین فضای خالی tab , یا Enter ادامه پیدا می کند. علاوه بر این دو تابع ,توابع زیر نیز از سرفایل stdio.h مورد استفاده قرار می گیرند :

تابع ورودی : **gets()** تابع خروجی : **puts()**

تابع()gets :

خواندن رشته با تابع gets() انجام می گیرد. این تابع در سرفایل stdio.h قرار دارد. این تابع بعد از خواندن رشته ,سطر جاری را رد می کند.نحوه کاربرد آن به صورت زیر است:

gets (متغیر رشته ای) ;

مثال:

```
char str[21];
gets(str);
```

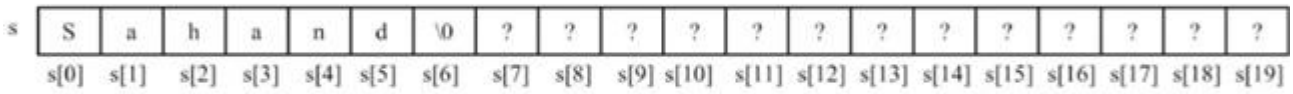
پس از وارد کردن رشته باید کلید ENTER را نیز فشار داد.

تفاوت ()gets با ()scanf

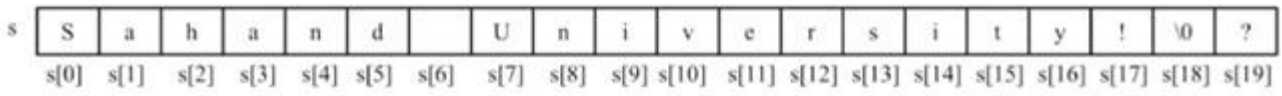
تفاوت ()gets با ()scanf در این است که در تابع ()gets فقط کلید ENTER انتهای رشته را مشخص می‌کند. لذا رشته می تواند حاوی فاصله (space) و یا tab باشد. در حالی که در تابع ()scanf فاصله و tab نیز به عنوان اتمام بخش تلقی شده و انتهای رشته را مشخص می کند.

```
char  s[20];
scanf("%s", s);
```

Sahand University!




```
gets(s);
```



تابع:()puts

چاپ رشته با تابع ()puts انجام می گیرد. این تابع در سرفایل stdio.h قرار دارد. این تابع پس از نوشتن رشته ,سطر جاری را رد می کند.کاربرد آن به صورت زیر است:

puts ("رشته");

puts (متغیر رشته ای) ;

برنامه‌ای بنویسید که رشته‌ای را دریافت کرده و در آرایه‌ای کاراکتری ذخیره کند. سپس طول رشته را محاسبه کرده و چاپ کند.

```
#include <stdio.h>
int main(){
  int i=0;
  char str[50];

  gets(str);

  while(str[i]!='\0')
      i++;

  printf("Length of your string is : %d", i);
  getch();
  return 0;
}
```

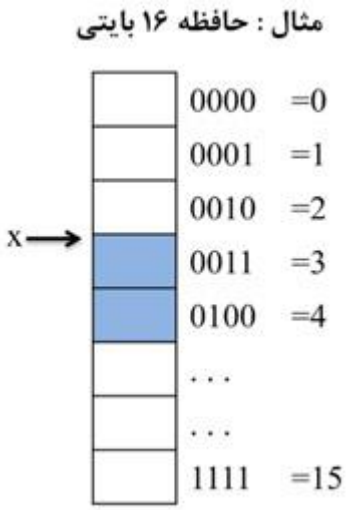
```
It is a statement.
Length of your string is : 18
```

```
void    func1 (char x[]);
void    func2 (char x[], int len);
int     main()
{
char    x[10];
. . . .
func1(x);
. . . .
func2(x, 10);
return  0;
}
void    func1 (char x[10])
{
  . . . .
}
void    func2 (char x[], int len)
{
  . . . .
}
```

اشاره گرها

مقدمه:

- **آدرس هر متغیر را در حافظه , اشاره گر گویند.**
- **بایت یکی از تقسیمات حافظه است .به عبارت دیگر ,حافظه کامپیوتر ,مجموعه ای از چندین بایت است . هر بایت دارای یک شماره ردیف است .شماره ردیف هر بایت از حافظه را آدرس آن محل از حافظه گویند.**
- **متغیرها نامی برای محل های حافظه اند ولذا بایت هایی از حافظه را اشتغال می کنند. آدرس اولین بایتی از حافظه که به متغیر اختصاص می یابد ,آدرس آن متغیر است.**



کاربردهای اشاره گرها

اقلب قابلیت های c به نقش اشاره گرها در این زبان برمی گردد:

تخصیص حافظه پویا : در این نوع تخصیص حافظه ,برنامه در زمان اجرا از سیستم حافظه می گیرد و در صورت عدم نیاز ,آن حافظه را به سیستم بر می گرداند. موجب بهبود کارایی توابع می شود (توابع را با استفاده از آدرس آن ها می توان فراخوانی کرد). سهولت کار با رشته ها و آرایه ها .ارسال آرگومان ها از طریق فراخوانی با ارجاع را امکان پذیر می سازد.

متغیر های اشاره گر

اشاره گر می تواند در متغیری ذخیره شود .اما ,با اینکه اشاره گر یک آدرس حافظه است و آدرس حافظه نیز یک عدد است ,ولی نمی توان آن را در متغیرهایی از نوع int , double و یا غیره ذخیره کرد. متغیری که می خواهد اشاره گر را ذخیره کند باید هم نوع با اشاره گر باشد .این متغیر ها را متغیرهای اشاره گر گویند. برای تعریف متغیرهای اشاره گر در زبان c به صورت زیر عمل می شود:

;متغیر* نوع

در تعریف متغیر اشاره گری که بخواهد آدرس متغیرهایی را نگه داری کند, باید نوع متغیر اشاره گری را هم نوع با آن متغیر ها در نظر گرفت و در کنار متغیر اشاره گر,علامت * را قرار داد.

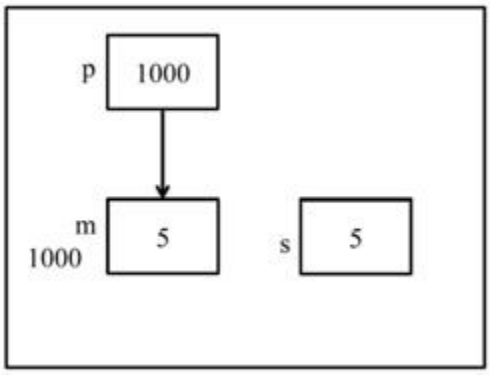
عملگرهای اشاره گر

عملگر & و *

هر یک از این دو عملگر یک عملوند دارند . عملگر & درس عملوند خودش را مشخص می کند .عملگر * محتویات جایی را مشخص می نماید که عملوندش به آن اشاره می کند

مثال:

```
int   *p , m , s ;
m = 5 ;
p = &m ;
s = *p ;
```



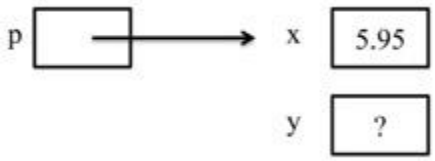
```
#include <stdio.h>
int main()
{
float  x, y ;
int  *p;
x = 5.95 ;
p = &x ;
y = *p ;
printf ("\n y = %f" , y) ;
return 0;
}
```

نکته:

اشاره گرها در زبان c دارای نوع اند . یعنی وقتی اشاره گری از نوع int تعریف می شود , باید به متغیر هایی از نوع int اشاره نماید و اشاره گر از نوع double باید به متغیر هایی از همین نوع اشاره نماید.

اگر نوع متغیری که آدرس آن در یک اشاره گر قرار می گیرد , با نوع اشاره گر یکسان نباشد ,کامپایلر زبان c خطایی را اعلام نمی کند.

مثال :خروجی کد مقابل چیست؟



اعمال روی اشاره گرها

تعداد اعمالی که می توان روی اشاره گر ها انجام داد بسیار کمتر از اعمالی است که روی متغیرهای دیگر می توان انجام داد.

عمل انتساب اشاره گرها به یکدیگر.

اعمال محاسباتی جمع و تفریق.

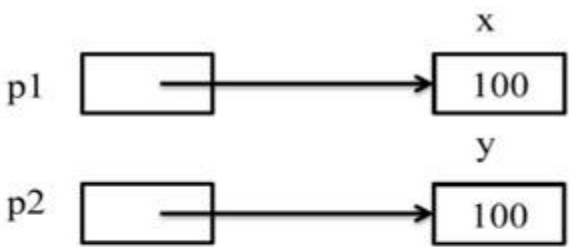
عمل مقایسه اشاره گرها.

```
int  *p1 , *p2 , x , y ;
x = 50;
y = 100 ;
p1 = &x ;
p2 = &y ;
*p1 = *p2 ;
```

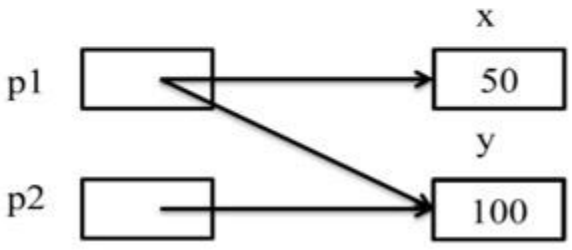
انتساب اشاره گرها به یکدیگر

اگر x و y دو متغیر باشند, دستور x=y آنچه را که در y قرار دارد در x قرار می دهد .این عمل را انتساب y به x گویند. دو اشاره گر را نیز می توان به یکدیگر نسبت داد . در این صورت هر دو اشاره گر به یک محل از حافظه اشاره خواهند کرد.

```
int  *p1 , *p2 , x , y ;
x = 50;
y = 100 ;
p1 = &x ;
p2 = &y ;
*p1 = *p2 ;
```




```
int  *p1 , *p2 , x , y ;
x = 50;
y = 100 ;
p1 = &x ;
p2 = &y ;
p1 = p2 ;
```



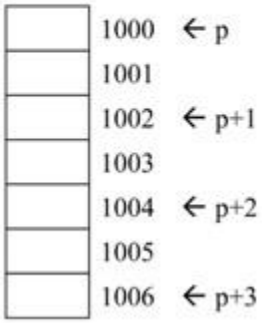
اعمال محاسباتی

اعمال جمع و تفریق را می توان بر روی اشاره گرها انجام داد.

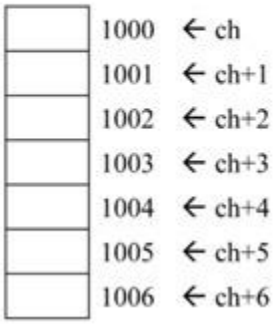
با افزایش یک واحد به اشاره گر,به اندازه طول نوع اشاره گر ,به آن اضافه می شود.با کاهش یک واحد از اشاره گر به اندازه طول نوع اشاره گر از آن کم می شود.

مثال:

```
int *p;
```




```
char *ch;
```

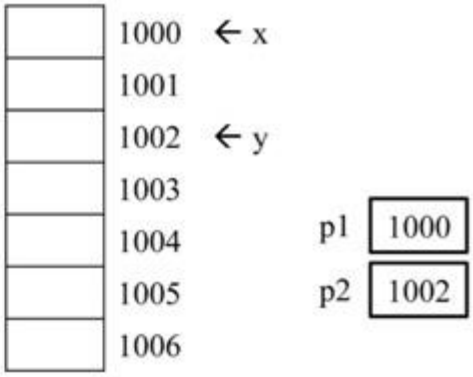


مقایسه اشاره گرها

اگر p1 و p2 دو اشارگر باشند, با استفاده از عملگرهای رابطه ای مانند متغیر های معمولی با هم مقایسه می شوند.یعنی مقدار آدرس هر یک از اشاره گرها ,عدد بزرگتری باشد ,آن اشاره گر بزرگتر خواهد بود.

مثال: با استفاده از دستورات زیر ,دو اشاره گر p1 و p2 با هم مقایسه می شوند.فرض کنید p1 به محل ۱۰۰۰ حافظه و p2 به محل ۱۰۰۲ حافظه اشاره می کند .در این صورت شرط (p1==p2) ارزش نادرستی دارد.

```
int   x , y , *p1 , *p2 ;
p1 = &x ;
p2 = &y ;
if (p1 = = p2)
   . . .
else
   . . .
```



رشته به عنوان آرگومان تابع

چون رشته ها به صورت آرایه ای از کاراکترها تعریف می شوند, شیوه ارسال رشته ها به توابع,همانند آرایه است در آرگومان تابع ,نام رشته ذکر می شود.

پارامتر معادل با رشته می تواند یکی از سه مورد زیر باشد:

۱. آرایه ای با طول معین
۲. آرایه ای با طول نامعین
۳. اشاره گر

برنامه‌ای بنویسید که رشته‌ای را خوانده، تمام حروف کوچک آن رشته را به حروف بزرگ تبدیل کرده و چاپ می‌کند.

```
#include <stdio.h>
void upper(char []);
int main()
{
  char s[21];
  printf("Enter a string: ");
  gets(s);
  upper(s);
  puts("Result is:");
  puts(s);
  getch();
  return 0;
}
```

```
Enter a string: It is a string.
Result is:
IT IS A STRING
```

```
void upper(char s[21])
{
  int i;
  for (i=0;i<21;i++)
     if (s[i]>='a' && s[i]<='z')
        s[i]-=32;

}
```