الگوریتم و فلوچارت

Algorithm and Flowchart

- مقدمه
- تعریف الگوریتم
- ویژگیهای الگوریتم
- انواع دستورالعمل‌ها
- نحوه بیان الگوریتم
- فلوچارت (رَوَند نما)
- الگوریتم‌های حلقوی

مقدمه

در زندگی روزمره، انسان با مسائل مختلفی روبروست و برای هر کدام از این مسائل راه حلی را انتخاب می‌کند. این مسائل عبارتند از راه رفتن، غذا خوردن و غیره که بشر تقریباً هر روز آنها را پیش روی خود دارد.

همه این مسائل نیاز به روشی برای حل کردن دارند مثلا راه رفتن باید با ترتیب خاصی و مراحل معینی انجام شود، تا مسئله راه رفتن برای بشر حل شود. اصطلاحاً روش انجام کار یا حل مسئله را الگوریتم آن مسئله می‌نامند.

تعریف الگوریتم Algorithm

به مجموعه دستورالعمل‌هایی که مراحل انجام کاری را با زبانی دقیق و با جزئیات کافی بیان نماید بطوریکه ترتیب مراحل و شرط خاتمه عملیات در آن کاملاً مشخص شده باشد، الگوریتم گویند.

منظور از زبان دقیق آن است که الگوریتم دقیقا" به همان صورتیکه مورد نظر نویسنده است، اجرا گردد.

منظور از جزئیات کافی، آن است که در طول اجرای الگوریتم عملیات ناشناخته پیش نیامده و باعث انحراف از مسیر و هدف اصلی نگردد.

منظور از ترتیب مراحل، آن است که مراحل اجرای الگوریتم قدم به قدم و با رعایت تقدم و تأخر مشخص شده باشد.

منظور از شرط خاتمه، پایان پذیر بودن الگوریتم می باشد و بهر حال الگوریتم باید در زمانی دلخواه و تحت شرط یا شرایط داده شده خاتمه پذیرد.

ویژگیهای الگوریتم

۱- الگوریتم باید ما را به نتیجه مورد نظر برساند.

۲- در زمان محدود پایان یابد.

۳- دستورالعملها باید به ترتیب منطقی پشت سرهم قرار گیرند.

۴- جملات الگوریتم ها باید به صورت امری یا سؤالی باشند.

۵- هر الگوریتم باید نقطه آغاز و پایان داشته باشد.

۶- مراحل دارای جزییات کافی باشند.

۷- مراحل با زبان دقیق نوشته شوند.

انواع دستورالعمل

۱- دستورالعمل‌های محاسباتی و انتسابی

(انتساب یک مقدار به یک متغیر*، عملیات ریاضی)

۲- عبارات توضیحی (اضافه کردن توضیح به الگوریتم)

۳- دستورالعمل‌های شرطی (بررسی شرط)

۴- دستورالعمل‌های ورودی و خروجی(ورود اطلاعات، چاپ داده‌ها و نتایج)

*** برای انتساب یک مقدار به یک متغیر* میتوان از علامت = یا ← استفاده کرد.**

*** متغیر به مکانی از حافظه اشاره می کند که داده در آن ذخیره می شود.**

نحوه بیان الگوریتم

بیان الگوریتم

الگوریتم را معمولا به چند روش میتوان نوشته و بیان کرد. در اینجا ما از دو روش استفاده میکنیم:

نوشتن الگوریتم به زبان فارسی

نوشتن الگوریتم به روش نمودار گردشی (FLOW CHART)

1. با جملات فارسی : عدم استفاده از نمادهای خاص در نوشتن الگوریتم
2. با زبان ریاضی: استفاده از نمادهای خاص ریاضی در نوشتن الگوریتم

مثال ۱ : الگوریتمی بنویسید که مجموع دو عدد را محاسبه کند.

۱- دو عدد را بگیر

۲- آنها را با هم جمع کن

۳- پایان

مثال ۲ : الگوریتمی بنویسید که مجموع دو عدد را محاسبه و چاپ نماید.

۱- دو عدد را بگیر

۲- آنها را با هم جمع کن

۳- جمع حاصل را چاپ کن

۴- پایان

مثال ۳ : الگوریتم تعویض چرخ پنچر شده یک اتومبیل.

۱- شروع.

۲- جک را زیر اتومبیل بگذارید.

۳- پیچهای چرخ پنچر شده را باز کنید .

۴- چرخ را خارج کنید.

۵- چرخ یدک را به جای چرخ پنچر شده بگذارید.

۶- پیچها را ببندید.

۷- اگر پیچها سفت نشده اند به مرحله ۶ برو.

۸- جک را پایین بیاورید.

۹- چرخ پنچر شده را در صندوق عقب اتومبیل بگذارید.

۱۰- پایان.

مثال دو – الگوریتم محاسبه محیط و مساحت یک دایره

توضیح: در اینجا قبل از نوشتن الگوریتم باید دانیم که محیط و مساحت هیچ معنا و مفهومی برای کامپیوتر نداشته و صرفا یک سری عملیات محاسباتی را در بر دارد. میدانیم که با داشتن شعاع یک دایره میتوان مساحت را از رابطه $S=3.14*R^2$ و محیط را از فرمول $P=2*3.14*R$ بدست آورد.

۱- عدد (شعاع) را بخوان

۲- عدد خوانده شده را در ۶.۲۸ ضرب کن

۳- نتیجه (محیط) را نمایش بده

۴- عدد خوانده شده را به توان ۲ برسان و حاصل را در ۳.۱۴ ضرب کن

۵- نتیجه (مساحت) را نمایش بده

۶- پایان الگوریتم

Write an Algorithm?

An algorithm to find sum of two numbers:

Step 1: Start

Step 2: Assume two numbers **x** and **y** and a variable **sum=0**

Step 3: Add two numbers x and y; store the value in variable sum

Step 4: If you want to try again with different numbers
then goto step 2
else
goto step 5

Step 5: END

مفهوم متغیر

در مثال قبل میتوانستیم نامی لاتین برای مکان نما مشخص کنیم و آنگاه آنرا یک متغیر بنامیم. بعبارت دیگر منظور از متغیر محل معینی از حافظه است که یک نام سمبلیک به آن نسبت داده ایم و در طول اجرای برنامه میتوانیم از آن مقدار استفاده کنیم و یا آنرا تغییر بدهیم.

در هر زبان برنامه سازی، نامگذاری متغیرها دارای قوانین خاصی است. از این متغیرها در روابط ریاضی میتوان استفاده کرد.

مثال۴ : الگوریتمی بنویسید که سه عدد را از ورودی خوانده، میانگین آنها را محاسبه و چاپ نماید.

1- A,B,C را بخوان

2- Sum=A+B+C

3- Avg=Sum/3

4- Avg را چاپ کن

5- پایان

مثال ۵ : الگوریتمی بنویسید که طول و عرض مستطیلی را از ورودی بگیرد و محیط و مساحت آنرا محاسبه و نمایش دهد؟

۱- شروع

۲- X,Y را بخوان

۳- s=x*y

۴- p=2*(x+y)

۵- S,P را نمایش بده

۶- پایان

مثال ۶ : الگوریتمی بنویسید که عددی را دریافت کرده و تعیین کند عدد مثبت است یا منفی.

۱- شروع

۲- N را بخوان

۳- اگر N > 0 آنگاه پیغام " عدد مثبت است " را نمایش بده

۴- اگر N < 0 آنگاه پیغام " عدد منفی است " را نمایش بده

۵- پایان

مثال چهار – الگوریتم محاسبه مجموع اعداد یک تا صد

۱- عدد یک را در M قرار بده

۲- عدد صفر را در S قرار بده

۳- اگر M>100 بود، آنگاه به دستورالعمل شماره ۷ برود

۴- محتوای M را با محتوای S جمع کن و نتیجه را در S قرار بده

۵- به محتوای M یک واحد اضافه کن و نتیجه را در M قرار بده

۶- به دستورالعمل شماره ۳ برو

۷- محتوای S را چاپ کن

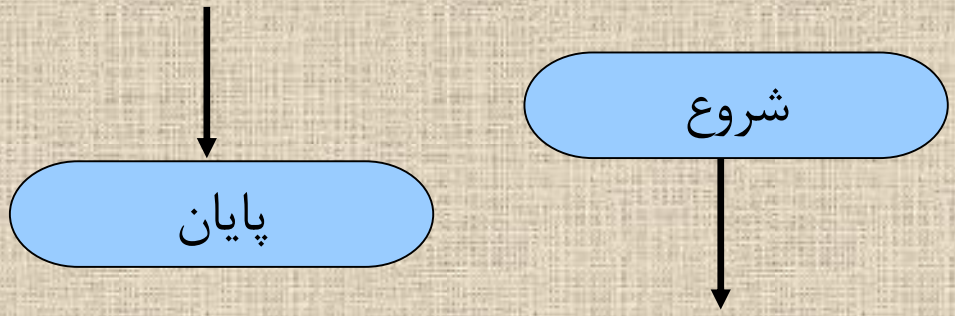
۸- پایان الگوریتم

فلوچارت (روند نما) Flowchart

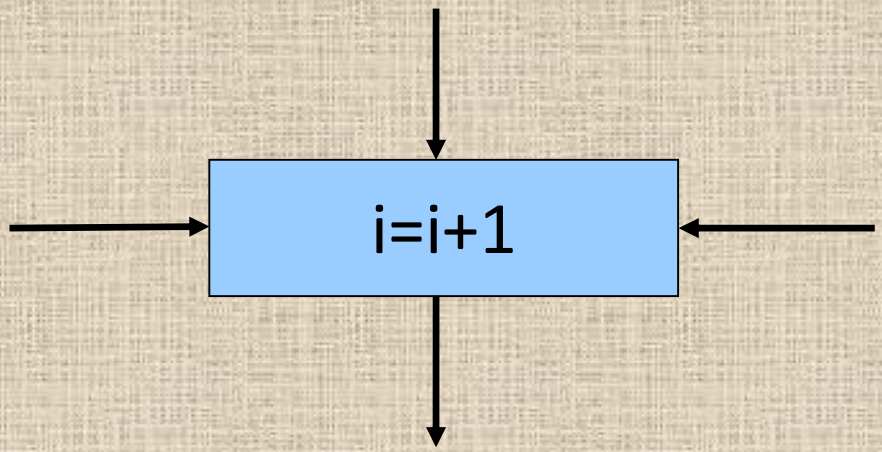
نمـوداری اسـت بـرای نمـایش داده‌هـا، اطلاعـات و رونـد کـار یـک الگوریتم، به‌وسیله نمادهای خاص و خطوط جهت‌دار بین آنها

فلوچارت در واقع نقشـه‌ای اسـت کـه برنامـه‌نویسـان رایانـه قبـل از نوشتن برنامه به زبان برنامه‌نویسی اصلی آن را ترسیم می‌کنند.

علائم و نمادهای استاندارد فلوچارت

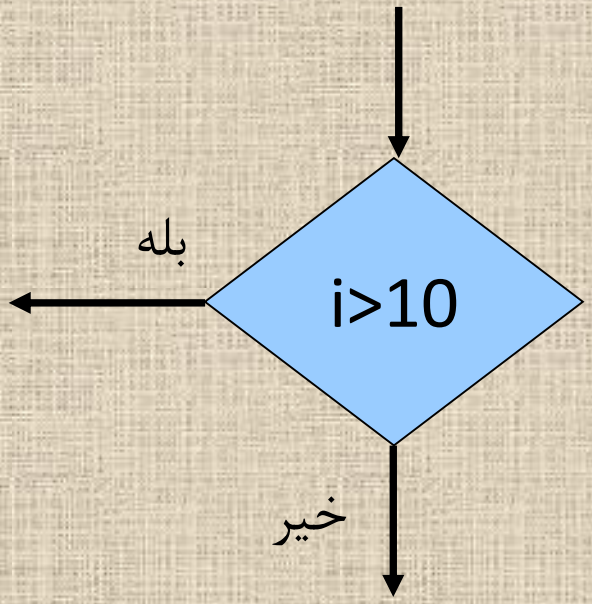


۱- شروع و پایان

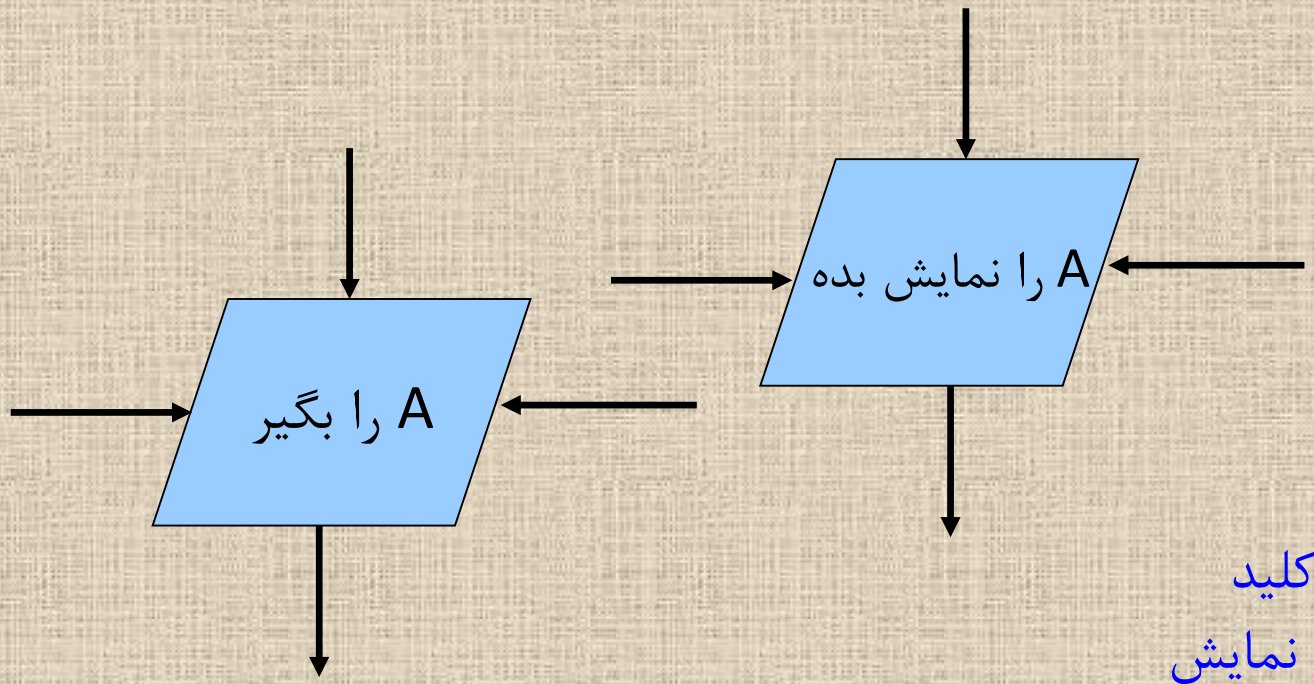


۲- محاسبات، مقدار دهی

۳- تصمیم گیری، شرط



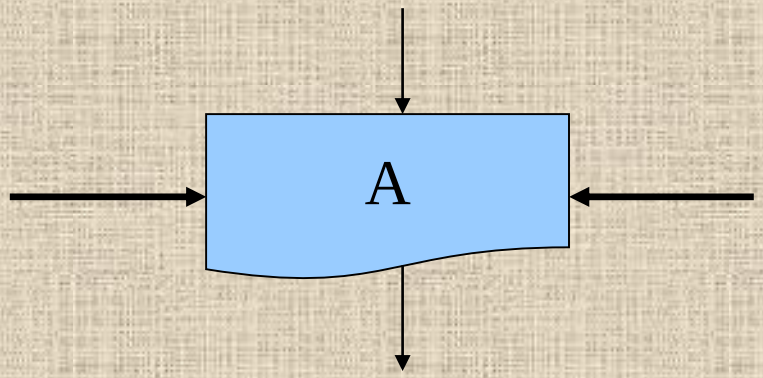
۴- ورودی و خروجی



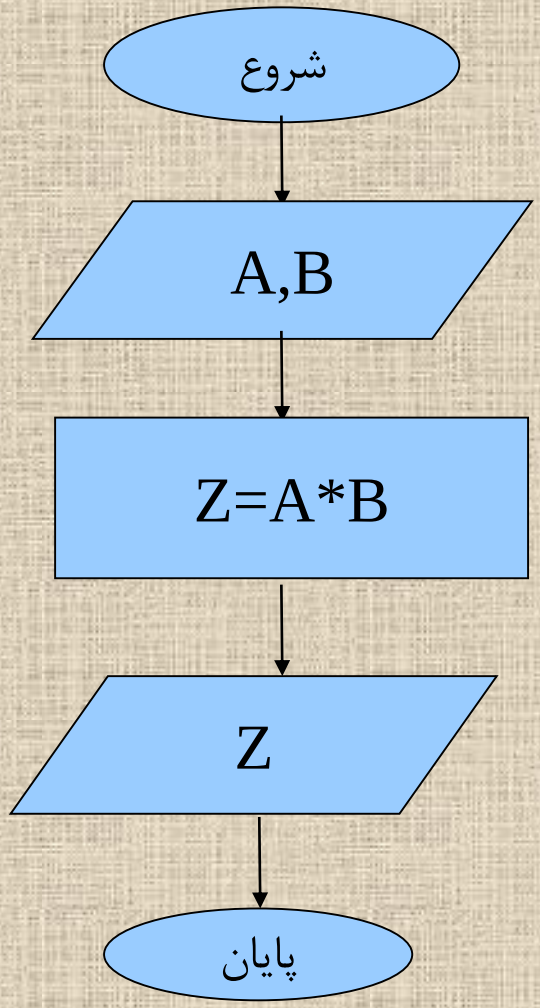
ورودی استاندارد: صفحه کلید

خروجی استاندارد: صفحه نمایش

۵- چاپ اطلاعات بر روی کاغذ (چاپگر)



مثال ۷ : فلوچارتی رسم کنید که دو عدد را خوانده و حاصلضرب آنها را نمایش دهد.

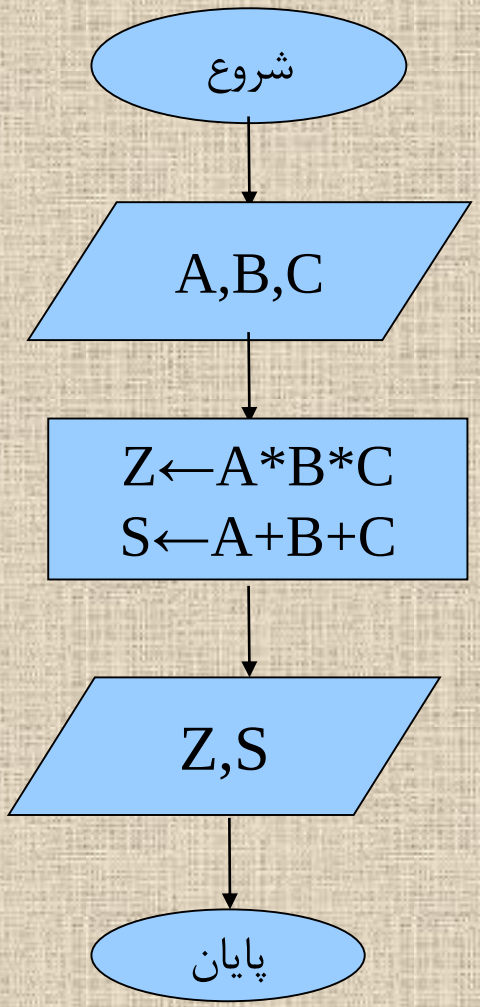


خواندن A و B

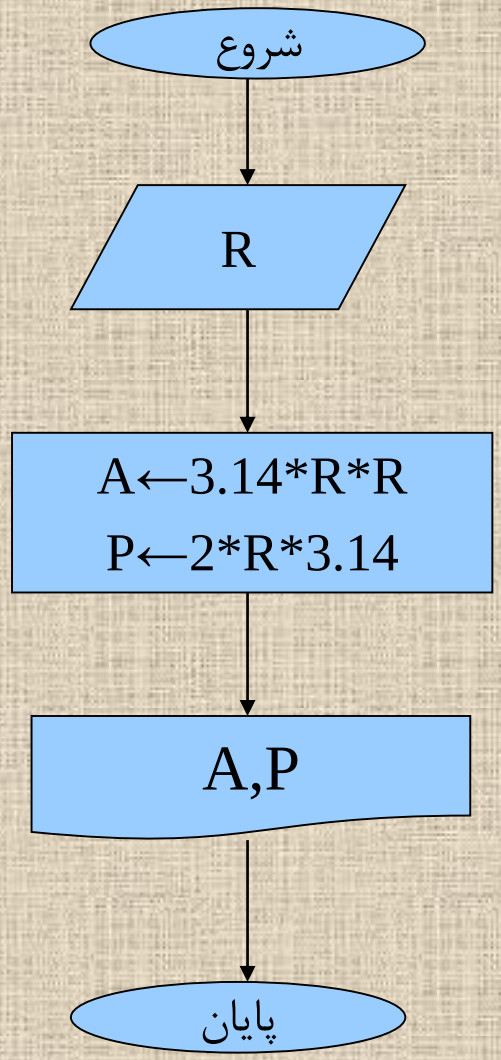
محاسبه حاصلضرب دو عدد و قرار دادن نتیجه در Z

نمایش Z

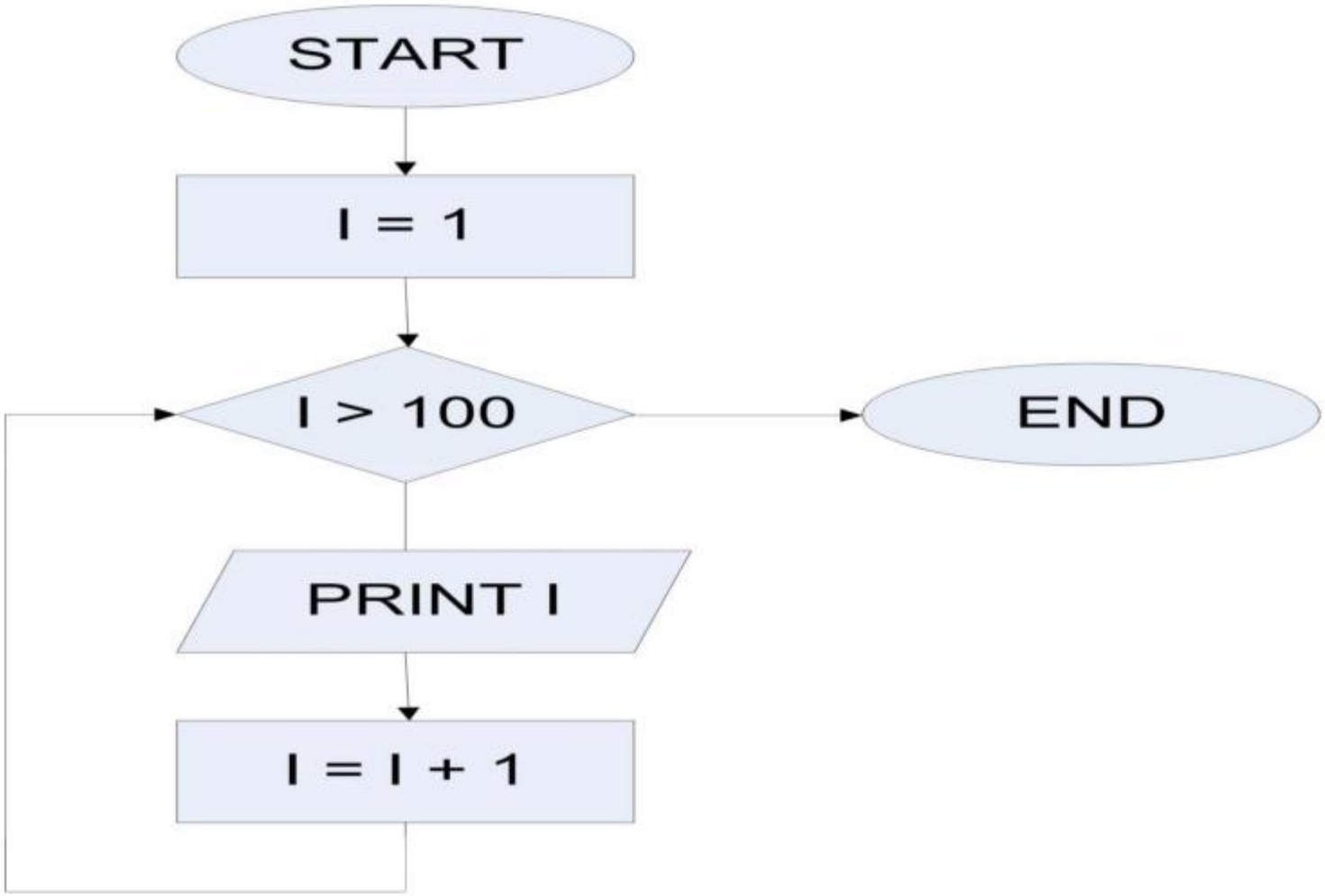
مثال ۸ : فلوچارتی رسم کنید که سه عدد را خوانده، حاصلضرب و حاصل جمع آنها را نمایش دهد.



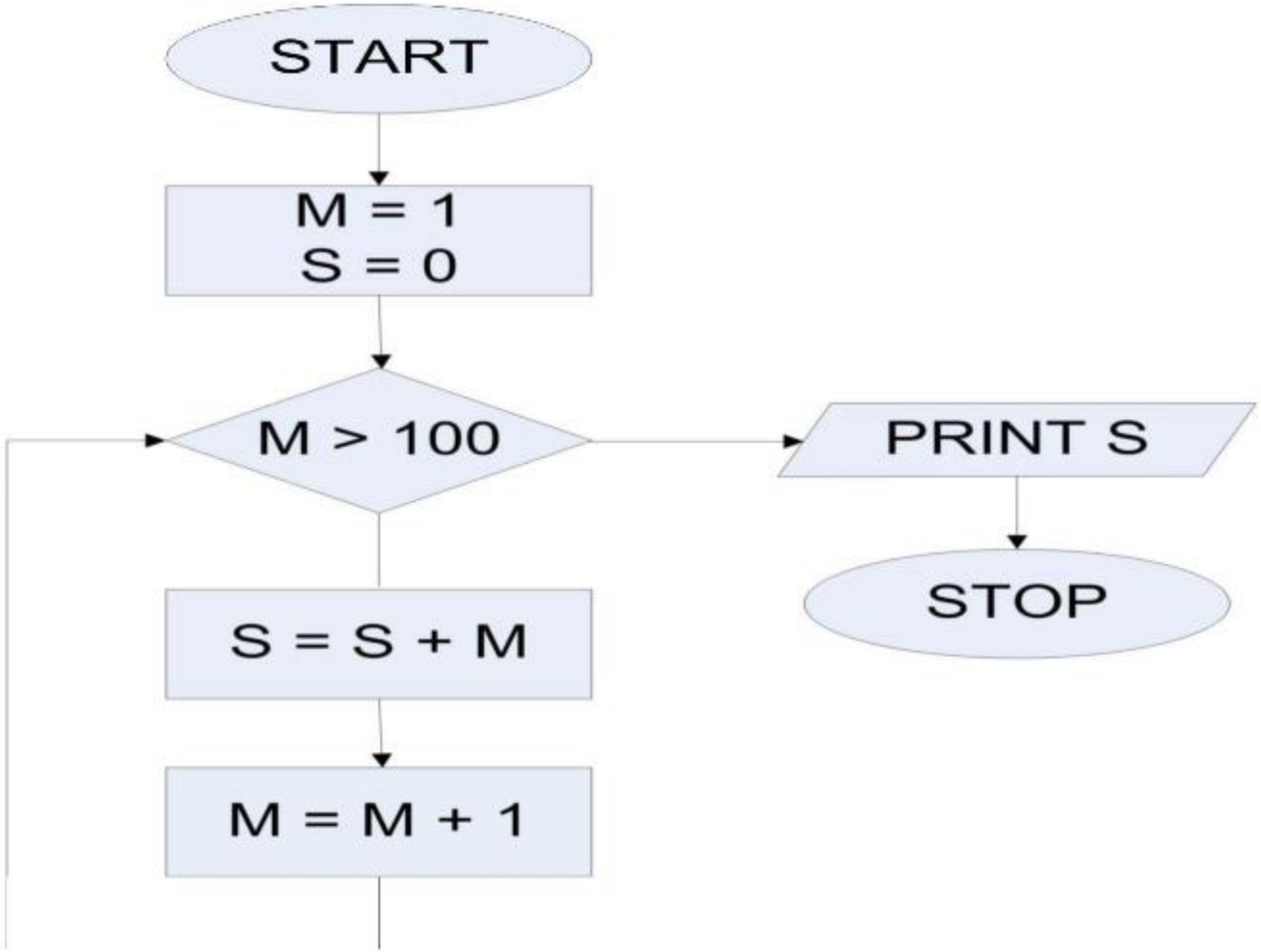
مثال ۹: فلوچارتی رسم کنید که شعاع یک دایره را خوانده، مساحت و محیط آنرا بر روی کاغذ چاپ نماید.



مثال سوم – فلوچارت چاپ اعداد یک تا صد



مثال چهارم – فلوچارت محاسبه مجموع اعداد یک تا صد



مثال شش – الگوریتم یافت کوچکترین عدد بین N عدد

۱- عدد یک را در متغیر I قرار بده

۲- عددی را بخوان و در متغیر M قرار بده

۳- اگر I=100 بود آنگاه به دستورالعمل شماره ۸ برو

۴- به محتوای متغیر I یک واحد اضافه کن

۵- عددی را بخوان و در متغیر L قرار بده

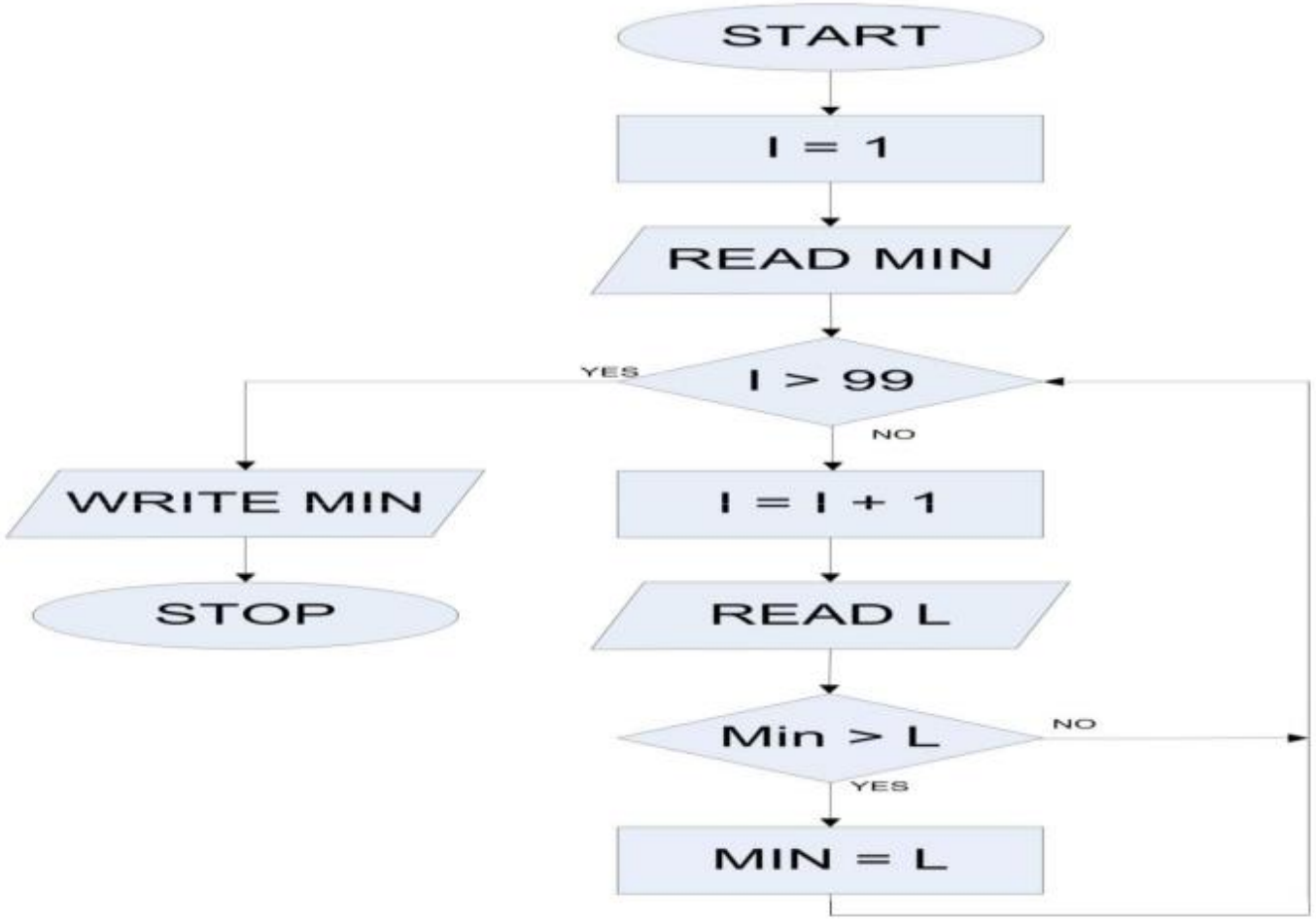
۶- اگر L<M است، آنگاه محتوای L را در M قرار بده

۷- به دستورالعمل شماره ۳ برو

۸- محتوای متغیر M را چاپ کن

۹- پایان الگوریتم

مثال ششم – فلوچارت یافتن کوچکترین عدد بین ۱۰۰ عدد



تمرینات:

۱- فلوچارتی ترسیم کنید که سه ضریب معادله درجه دو را از ورودی بخواند و ریشه های معادله را در تمام حالات پیدا کرده و با پیغام مناسبی چاپ کند.

۲- فلوچارتی ترسیم کنید که یکصد عدد را از ورودی بخواند و بزرگترین، کوچکترین و میانگین آنها را پیدا کرده و در خروجی چاپ کند (مجاز به استفاده از آرایه نیستید).

۳- فلوچارتی ترسیم کنید که عددی را از ورودی بخواند و فاکتوریل آن را محاسبه کند و در خروجی چاپ کند.

۴- فلوچارتی ترسیم کنید که یکصد عدد را از ورودی بخواند و تعیین کند کدام دو عدد مجاور از همه بزرگتر هستند.

۵- فلوچارتی ترسیم کنید که یکصد عدد را از ورودی بخواند و عدد بزرگتر دوم را پیدا کرده و در خروجی چاپ کند.

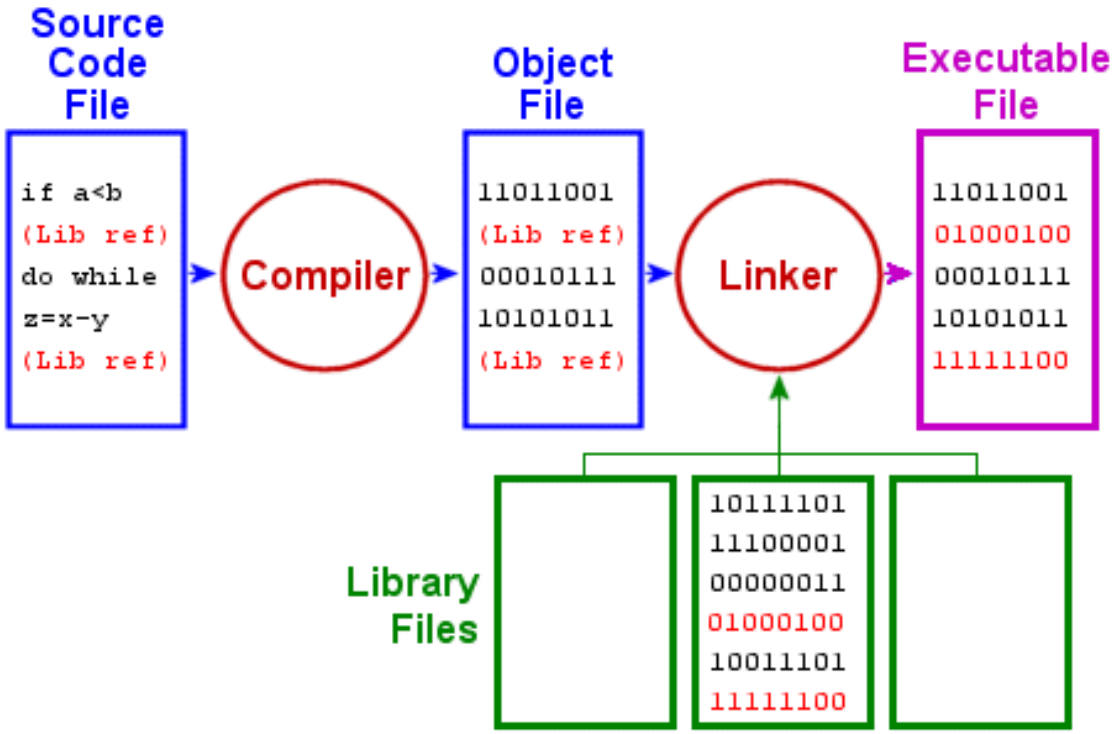
پایان جزوه دوم

- پیروز باشید.
- ظهیرپور
- اسفندماه ۹۸

زبان های برنامه‌نویسی

- نرم‌افزارها توسط زبان های برنامه‌نویسی نوشته می‌شوند. زبان های برنامه‌نویسی، یک سیستم ارتباطی هستند که توسط آن ها می‌توان دستورات لازم را به ماشین انتقال داد.
- هر زبان برنامه‌نویسی به مجموعه‌أی از علایم، قواعد و دستورالعمل‌ها گفته می‌شود که امکان ارتباط با کامپیوتر را جهت بیان کاری یا حل مسئله‌ای فراهم می‌کند.

- در حالت کلی زبان های برنامه‌نویسی را به سه دسته زیر تقسیم‌بندی می‌کنند:
- زبان های سطح بالا
- زبان های سطح پایین
- زبان های سطح میانی
- کامپایلر برنامه نوشته در یک زبان سطح بالا را به برنامه مقصد تبدیل می‌کند.



جلسه اول: الگوریتم ها

هدفهای کلی

شناخت حل مسئله و ارائه الگوریتم

شناخت اجزاء لازم برای حل مسئله

بررسی صحت الگوریتم

هدفهای رفتاری

دانشجو پس از مطالعه این فصل باید بتواند:

- الگوریتمی را برای حل مسئله ارائه دهد.
- الگوریتم های مختلف برای یک مسئله را مقایسه کند.
- شرط ها و حلقه ها را در الگوریتم بکار ببرد .

مقدمه

در زندگی روزمره، انسان با مسائل مختلفی روبروست و برای هر کدام از این مسائل (حل مشکلات) راه حلی و روشی را بر می‌گزیند. مسائلی از قبیل راه رفتن، غذا خوردن، خوابیدن و غیره که بشر تقریباً هر روز آنها را پیش روی خود دارد.

همه این مسائل نیاز به روشی برای حل کردن دارند مثلا راه رفتن باید با ترتیب خاصی و مراحل معینی انجام شود. تا مسئله راه رفتن برای بشر حل شود. اصطلاحاً روش انجام کار یا حل مسئله را الگوریتم آن مسئله می‌نامند.

تعریف الگوریتم

هر دستورالعملی که مراحل انجام کاری را با زبانی دقیق و با جزئیات کافی بیان نماید بطوریکه ترتیب مراحل و شرط خاتمه عملیات در آن کاملا" مشخص شده باشد را الگوریتم گویند. به عبارتی دیگر: الگوریتم مجموعه‌ای از دستورالعمل ها، برای حل مسئله می‌باشد که شرایط زیر را باید دارا باشد:

- ✓ *دقیق باشد*
- ✓ *جزئیات کامل حل مسئله را داشته باشد.*
- ✓ *پایان‌پذیر باشد.*

مراحل الگوریتم

برای حل یک مسئله باید الگوریتم آن مسئله را مشخص کنیم (یا بیابیم). که اصطلاحاً طراحی الگوریتم برای آن مسئله نامیده می‌شود. در طراحی الگوریتم معمولاً سه مرحله زیر را از هم جدا می‌کنند:

- ✓ ***خواندن داده‌ها***
- ✓ ***انجام محاسبات***
- ✓ ***خروجی‌ها***

مثال : الگوریتمی بنویسید که دو عدد از ورودی دریافت کرده مجموع دو عدد را محاسبه و چاپ نماید.

ورودی ها	انجام محاسبات	خروجی‌ها
a , b	جمع دو عدد	مجموع دو عدد

0ـ شروع
۱ـ a, b را بخوان.
۲ـ مجموع a , b را محاسبه و در sum قرار بده.
۳ـ sum را در خروجی چاپ کن
۴ـ پایان

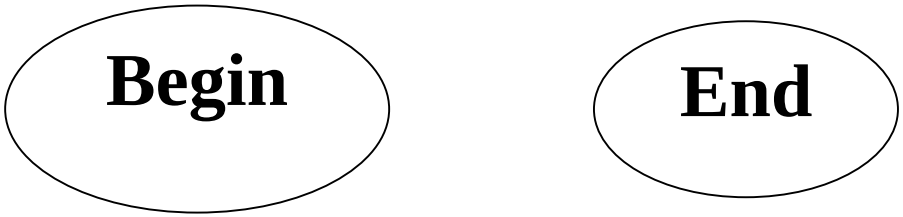
مثال: الگوریتمی بنویسید که سه عدد از ورودی دریافت کرده مجموع و میانگین سه عدد را محاسبه و چاپ کند.

ورودی ها	انجام محاسبات	خروجی‌ها
a	محاسبه مجموع	چاپ مجموع
b	محاسبه میانگین	چاپ میانگین
c		

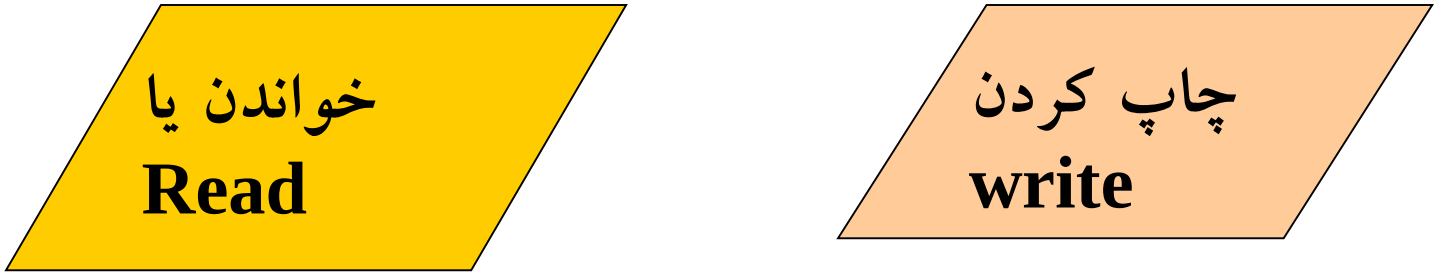
0ـ شروع
۱ـ سه عدد از ورودی بخوان
۲ـ مجموع سه عدد را محاسبه و در **sum** قرار بده.
۳ـ **sum** را بر سه تقسیم کرده،در **ave** قرار بده.
٤ـ **sum , ave** را در خروجی چاپ کن.
٥ـ پایان.

معمولا درک یک الگوریتم با شکل راحت تر از نوشتن آن بصورت متن می‌باشد. لذا الگوریتم را با فلوچارت(**flowchart**) نمایش می‌دهند.
فلوچارت از شکل‌های زیر تشکیل می‌شود.

علامت‌های شروع و پایان: که معمولا از یک بیضی استفاده می‌کنند:



علامتهای ورودی و خروجی: که معمولا از متوازی‌الاضلاع استفاده می‌شود:



علامتهای محاسباتی و جایگزینی: برای نمایش دستورات جایگزینی و محاسباتی از مستطیل استفاده می‌کنند:

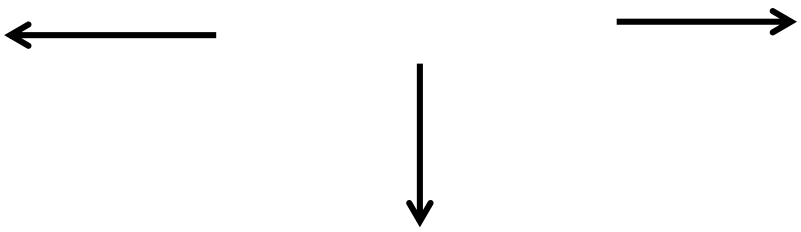
جایگزین یا محاسبات

علامت شرط:

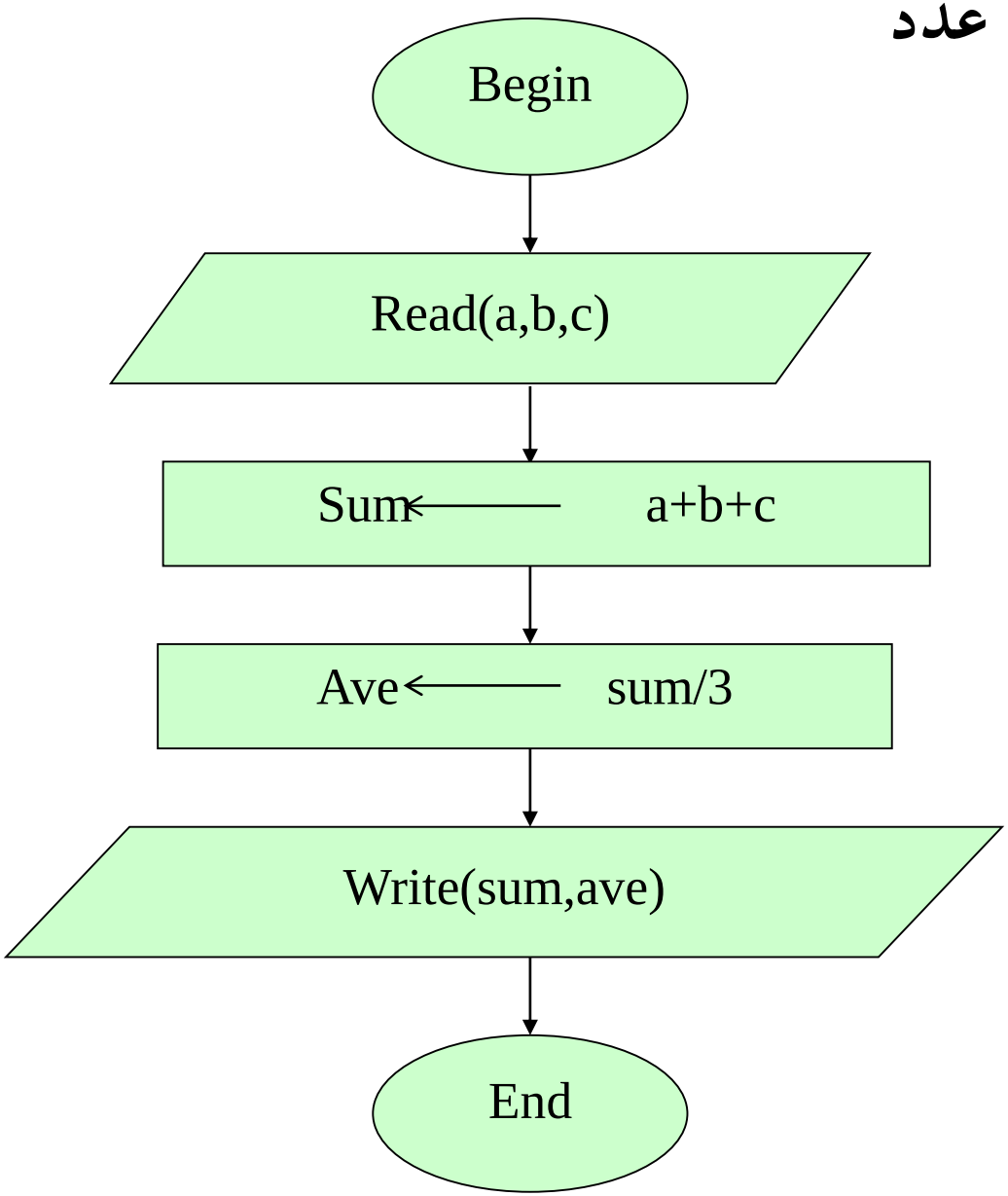
برای نمایش شرط از لوزی استفاده می‌شود.

علامت اتصال:

برای اتصال شکل‌های مختلف بهم از فلش‌های جهت‌دار استفاده می‌کنند.

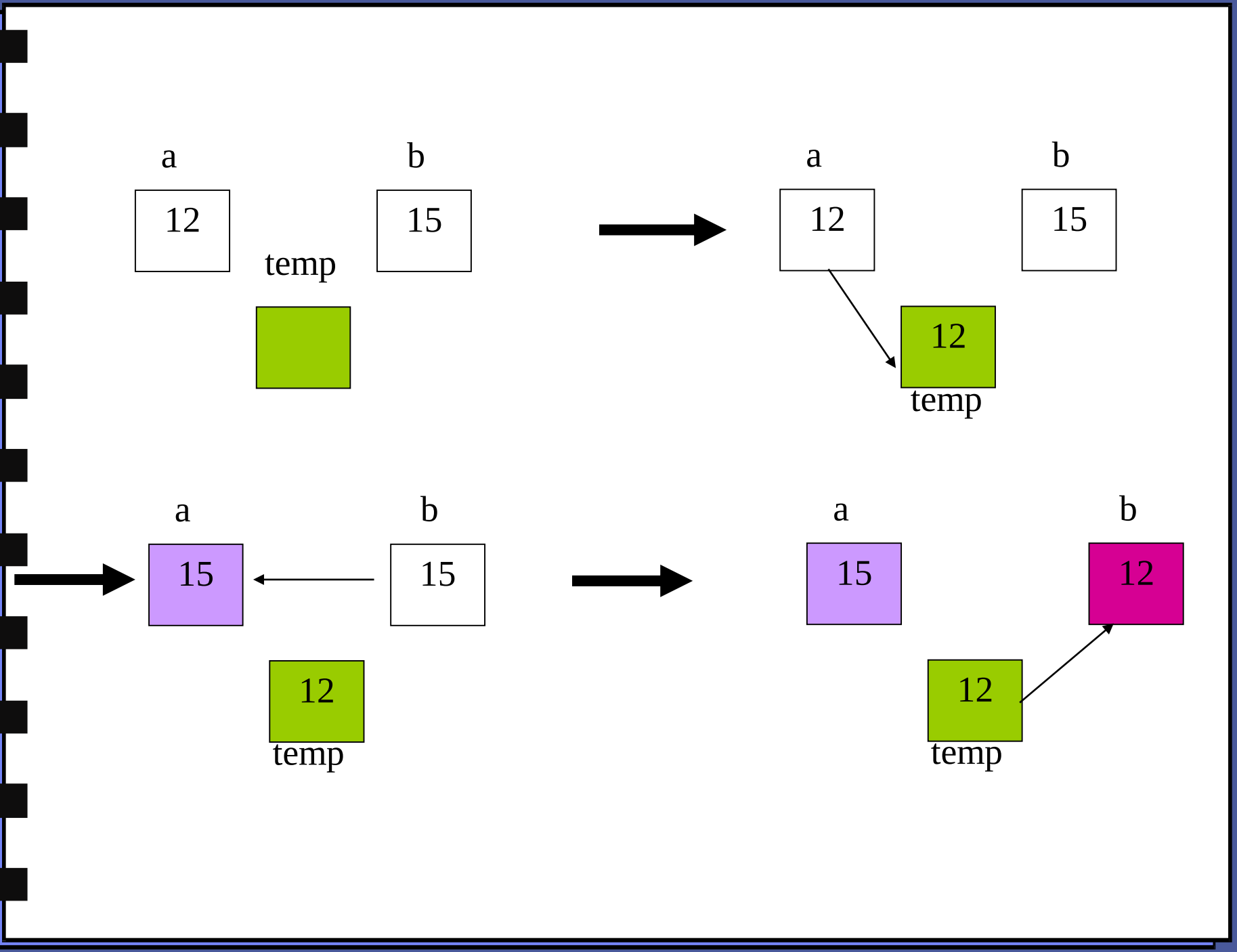


فلوچارت مجموع سه عدد

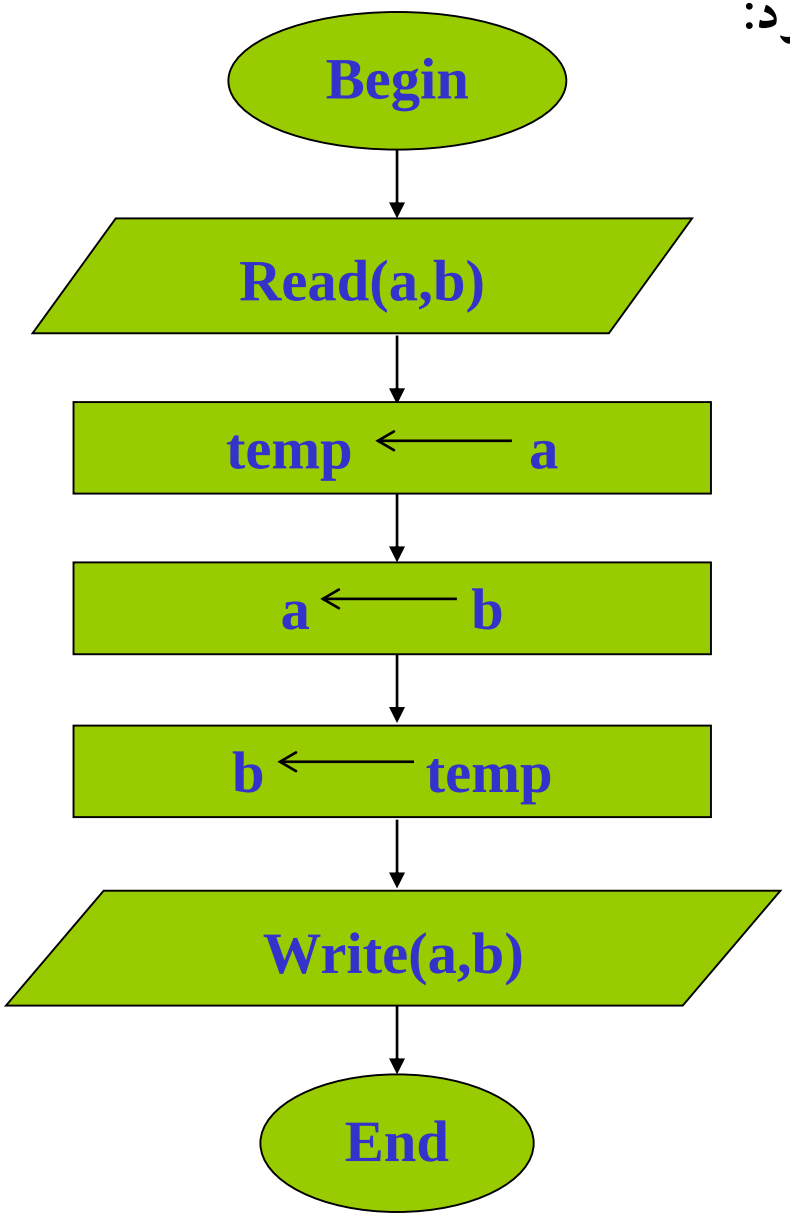


مثال: فلوچارتی رسم نمائید که دو عدد از ورودی دریافت کرده سپس محتویات دو عدد را با هم جابجا نماید.

برای حل این مسئله **a , b** را دو متغیر که در آنها دو عدد خوانده شده، قرار می‌گیرند در نظر می‌گیریم. سپس با استفاده از یک متغیر کمکی محتویات این دو عدد را جابجا می‌کنیم :


a
b
12
15
temp
a
b
12
15
12
temp
a
b
15
15
12
temp
a
b
15
12
12
temp

فلوچارت مسئله بالا بصورت زیر خواهد بود:



تمرین

۱ـ فلوچارتی رسم نمائید که طول و عرض مستطیل را از ورودی دریافت کرده محیط و مساحت آنرا محاسبه و چاپ کند.

۲ـ فلوچارتی رسم نمائید که شعاع دایره‌ای را از ورودی دریافت کرده، محیط و مساحت آنرا محاسبه و چاپ نماید.

۳ـ فلوچارتی رسم کنید که سه عدد Third , second, first را از ورودی دریافت کرده، محتویات آن ها را جابجا نموده، حاصل را در خروجی چاپ کند.

۴ـ فلوچارتی رسم نمائید که دو عدد از ورودی دریافت کرده، سپس محتویات دو عدد را بدون استفاده از متغیر کمکی جابجا کند.

۵ـ فلوچارتی رسم نمائید که عددی (درجه حرارت برحسب سانتیگراد) را از ورودی دریافت کرده سپس آن را به درجه فارنهایت تبدیل کند.

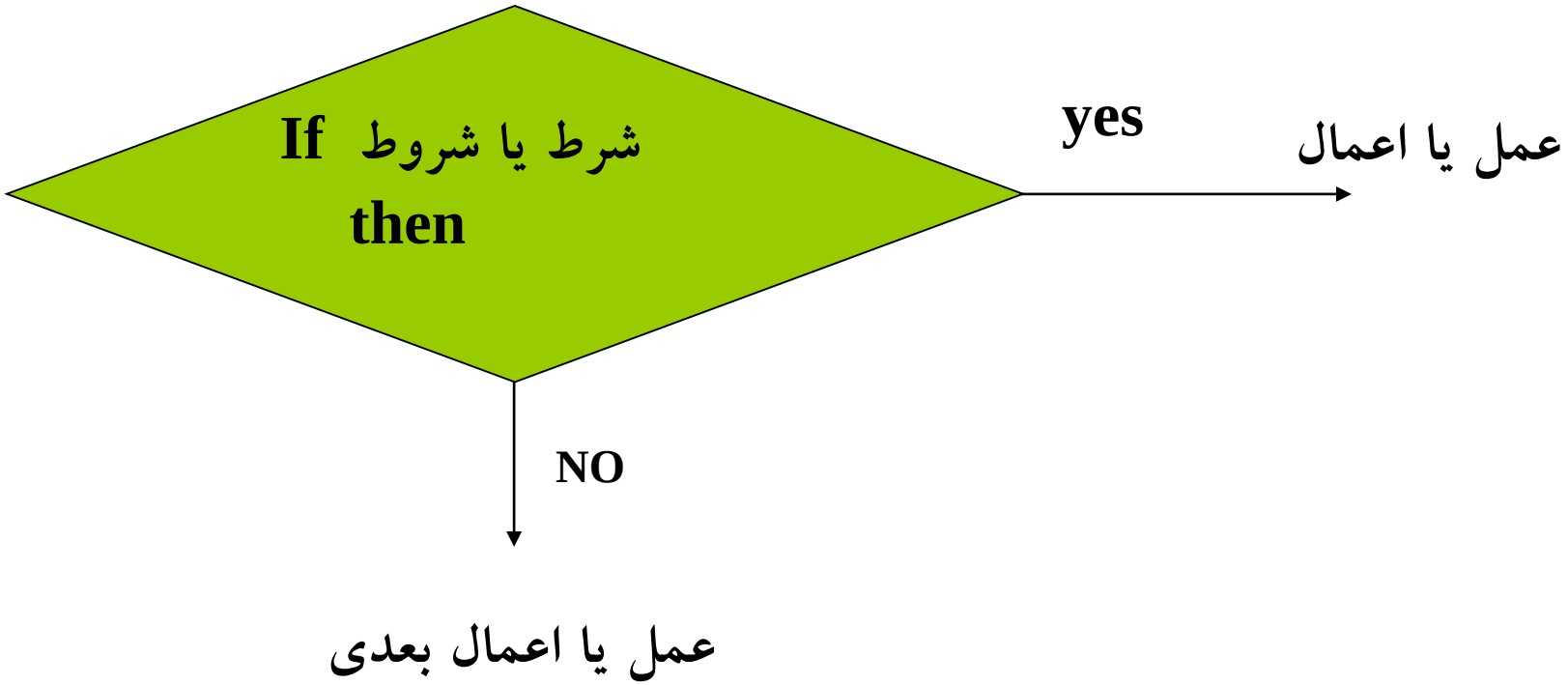
$$f = \frac{9}{5}\, c + 32$$

دستورالعمل‌های شرطی

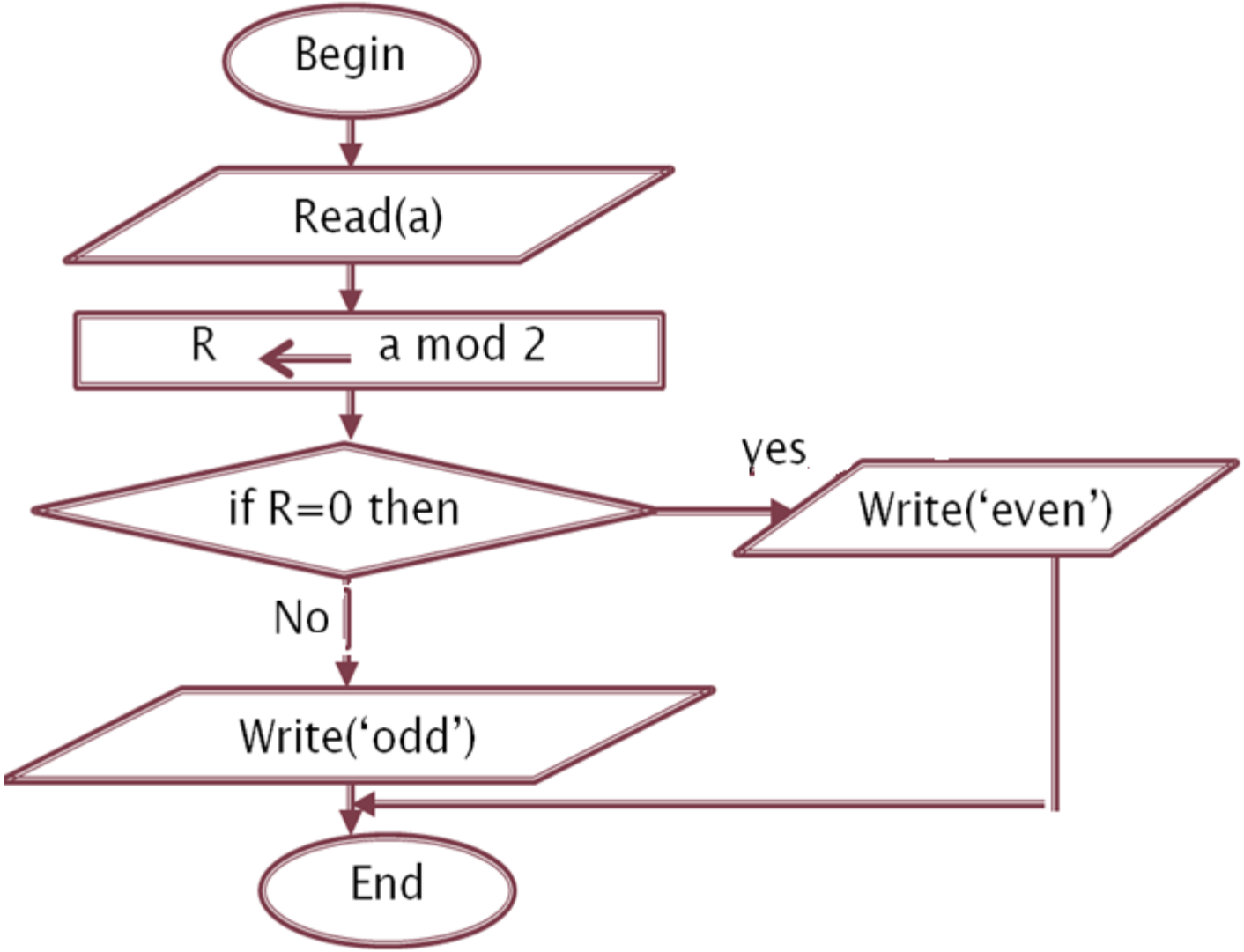
در حل بسیاری از مسائل یا تقریباً تمام مسائل نیاز به استفاده از شروط جزء، نیازهای اساسی محسوب می‌شود. همانطور که ما خودمان در زندگی روزمره با این شرط‌ها سرکار داریم. بطور مثال اگر هوا ابری باشد ممکن است چنین سخن بگوییم:

اگر هوا بارانی باشد سپس چتری برمی‌دارم. در غیر اینصورت چتر برنمی‌دارم.

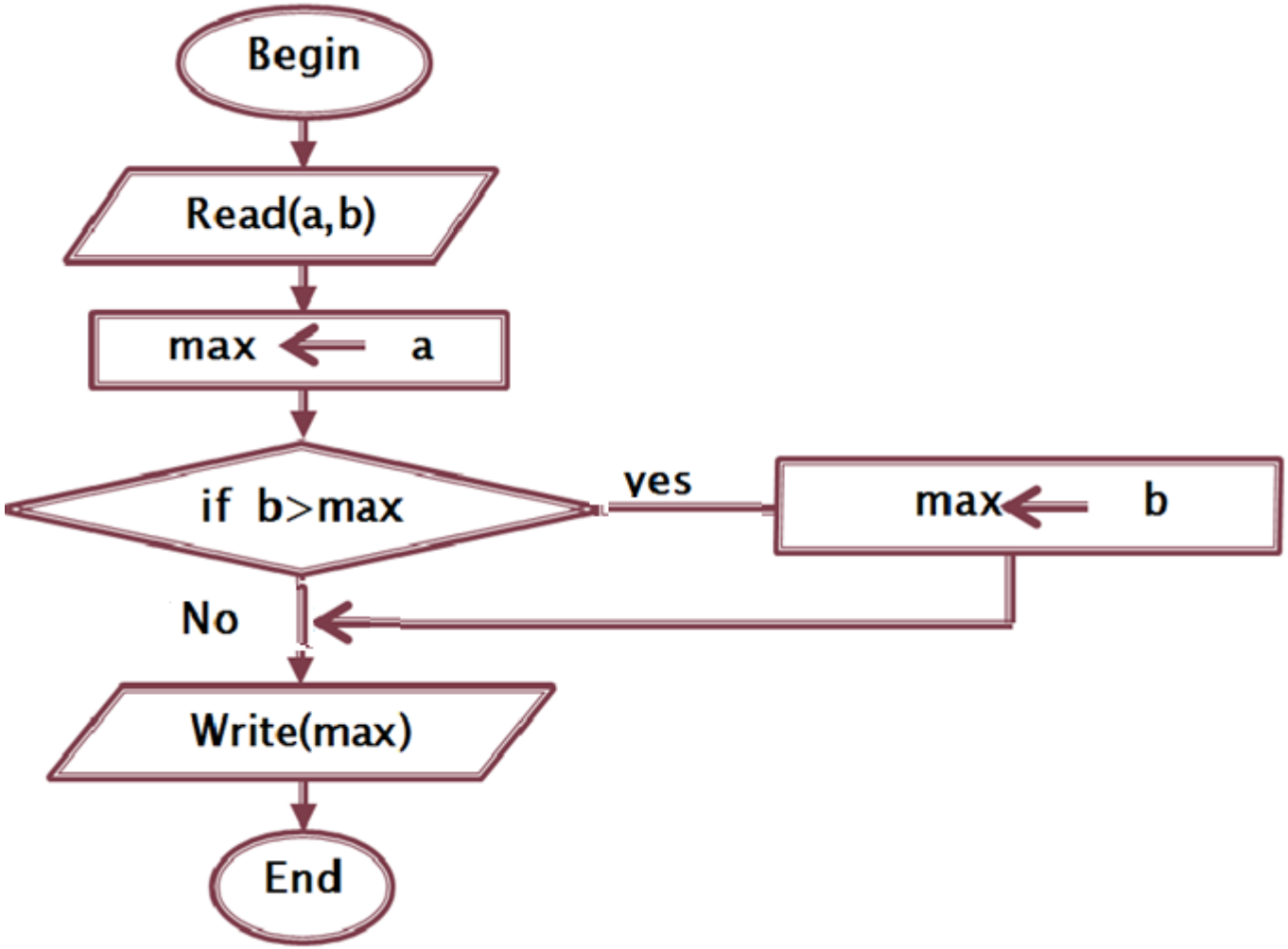
در حالت کلی شرط را بصورت زیر نمایش می‌دهند:



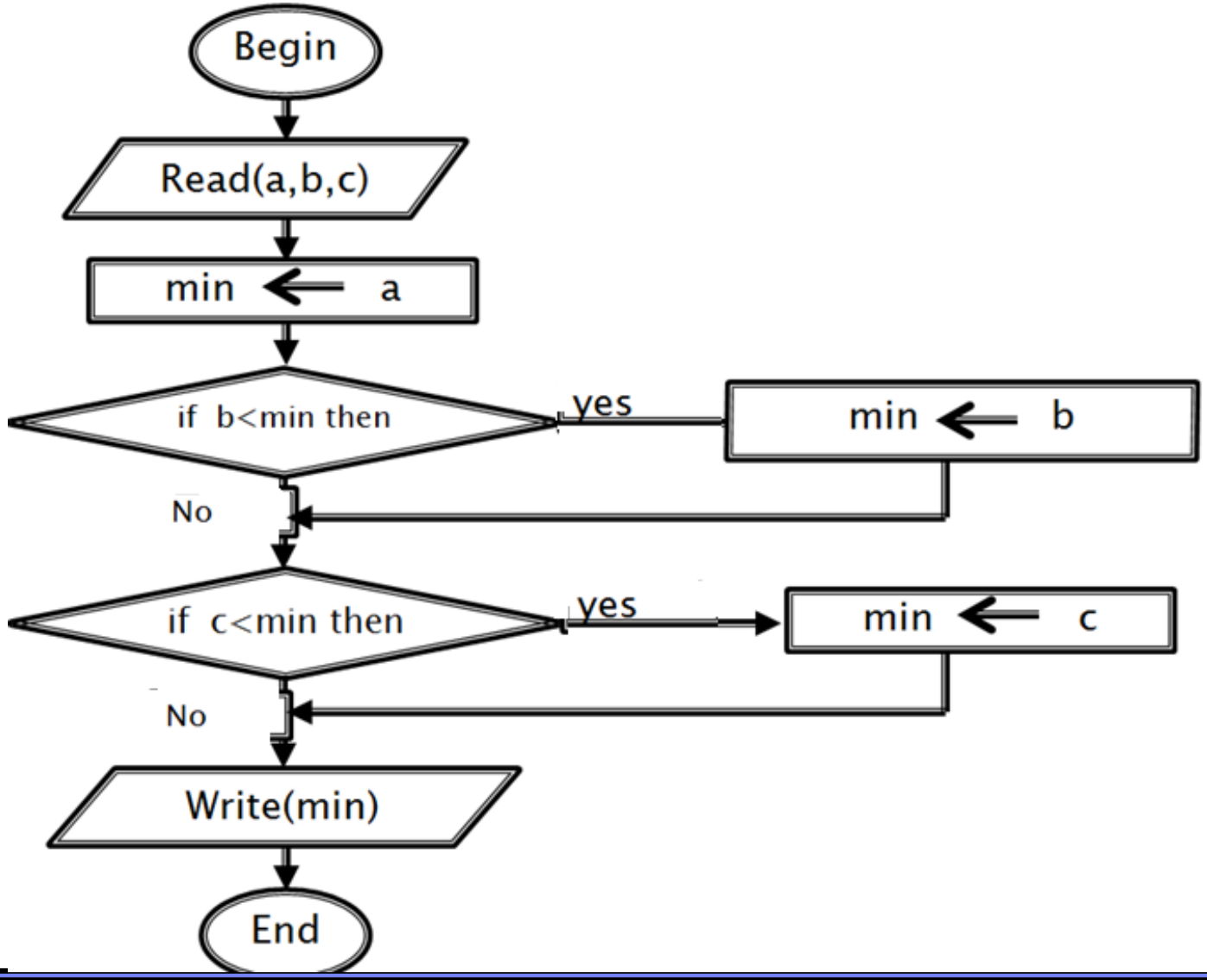
مثال : فلوچارتی رسم نمائید که عددی را از ورودی دریافت کرده، فرد یا زوج بودن آن را تشخیص دهد.



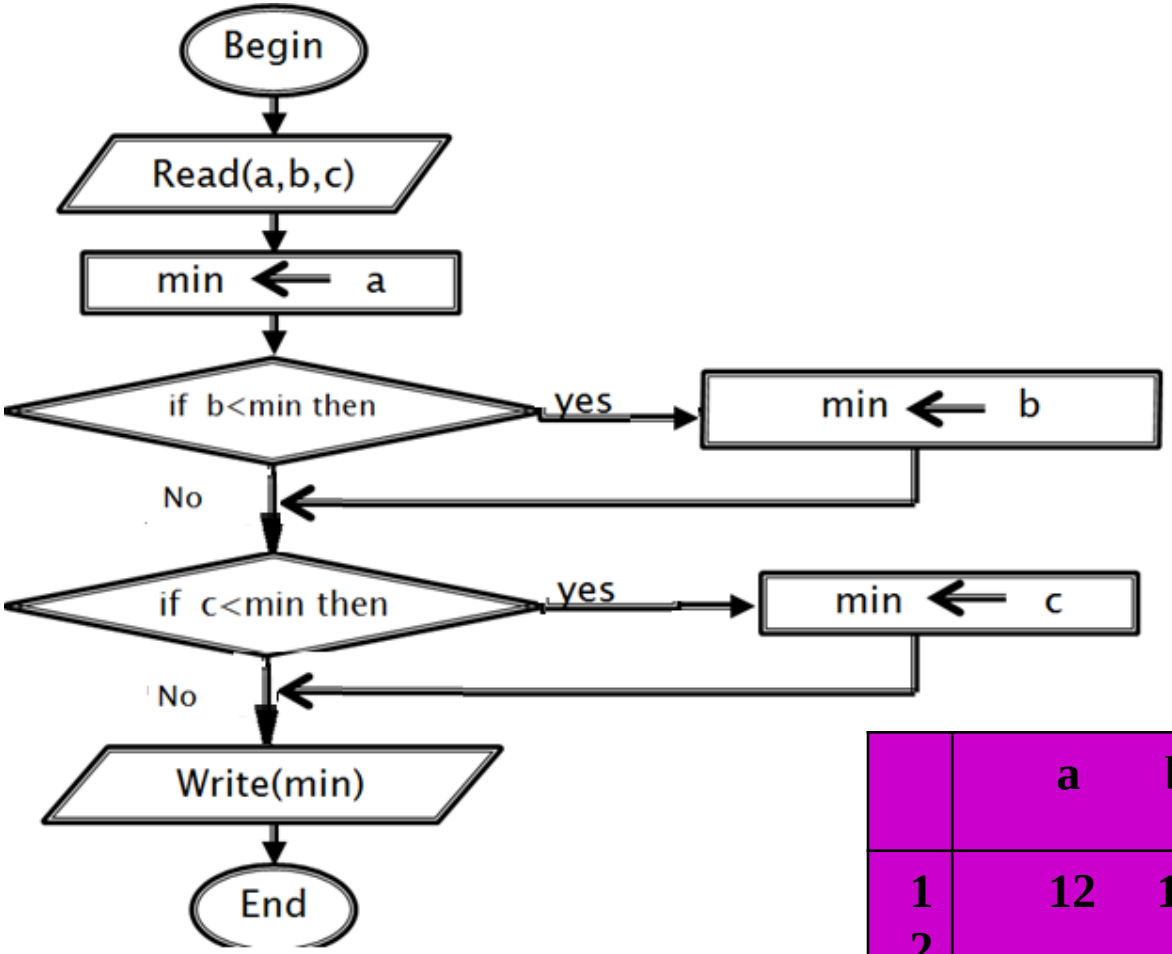
مثال : فلوچارتی رسم کنید که دو عدد از ورودی دریافت کرده بزرگترین عدد را پیدا کرده در خروجی چاپ نماید.



مثال : فلوچارتی رسم نمائید که سه عدد از ورودی دریافت کرده، کوچکترین عدد را یافته در خروجی چاپ نماید:



نمونه اجرای فلوچارت بالا بصورت زیر می‌باشد:



	a	b	c	Min	خروج
1	12	11	17		11
2				12	
3				11	
4				11	
5				11	

تمرین

۱- فلوچارتی رسم کنید که عددی را از ورودی دریافت کرده، قدر مطلق عدد را در خروجی چاپ کند.

۲- فلوچارتی رسم نمائید که عددی از ورودی دریافت کرده مثبت، منفی یا صفر بودن عدد را تشخیص داده، در خروجی با پیغام مناسب چاپ کند.

۳- فلوچارتی رسم نمائید که عددی را از ورودی دریافت کرده، بخش پذیری آن بر ۳ و ۵ را بررسی نماید.

۴ـ فلوچارتی رسم نمائید که ضرایب یک معادله درجه دوم را از ورودی دریافت کرده، ریشه‌های آن را محاسبه در خروجی چاپ کند.

حلقه‌ها

در حل بسیاری از مسائل با عملیاتی روبرو می‌شویم ، که نیاز به تکرار دارند و عمل تکرار آن ها به تعداد مشخصی انجام می‌گیرد. فرض کنید، بخواهیم میانگین ۱۰۰ عدد را محاسبه کنیم، در اینصورت منطقی بنظر نمی‌رسد که ۱۰۰ متغیر مختلف را از ورودی دریافت کنیم سپس آن ها را جمع کنیم.

انواع حلقه ها

حلقه های با تکرار مشخص

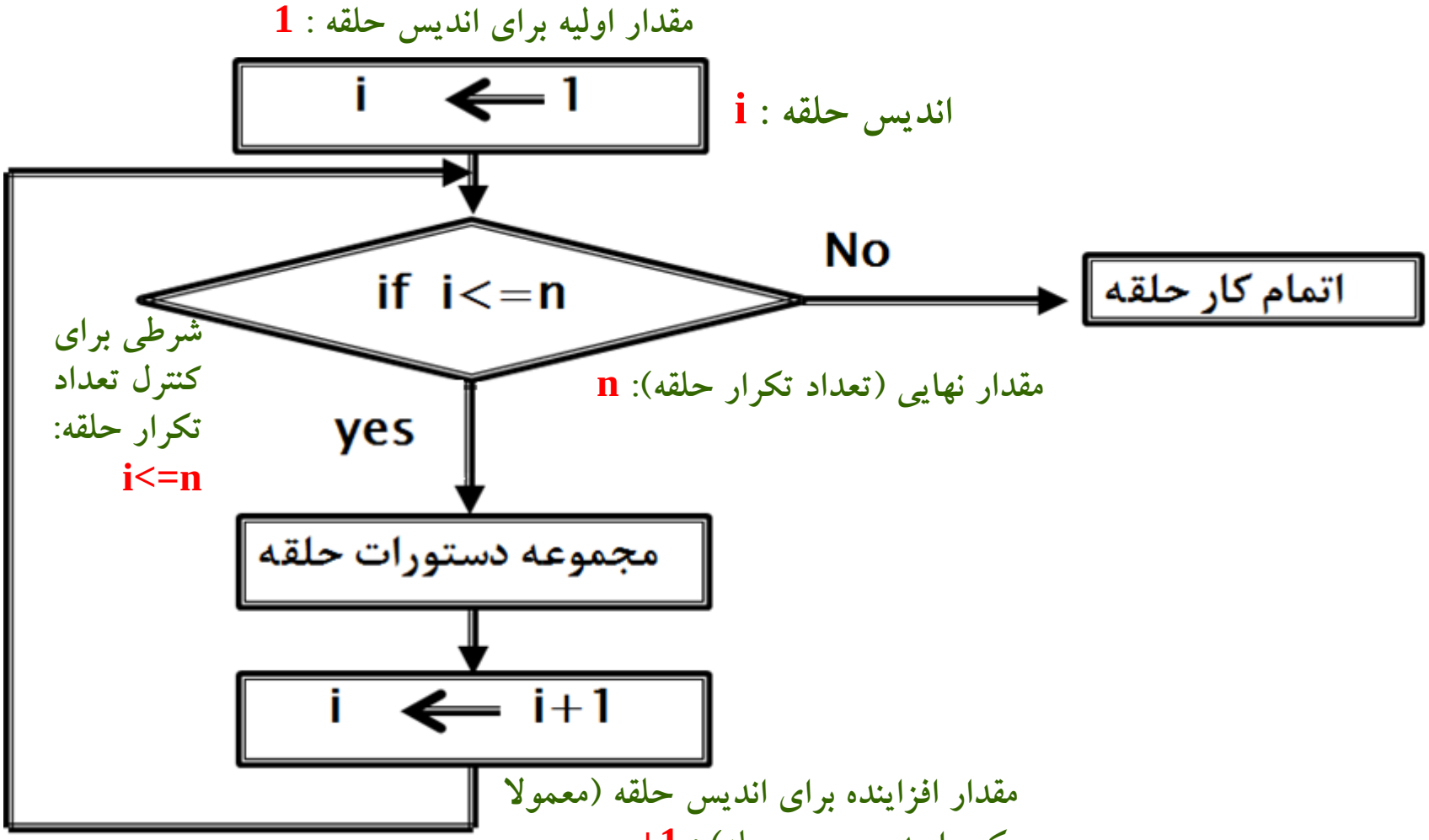
حلقه های با تکرار نامشخص

حلقه های با تکرار مشخص

در این نوع حلقه‌ها تعداد تکرار مشخص می‌باشد این حلقه از اجزاء زیر تشکیل می‌شود:

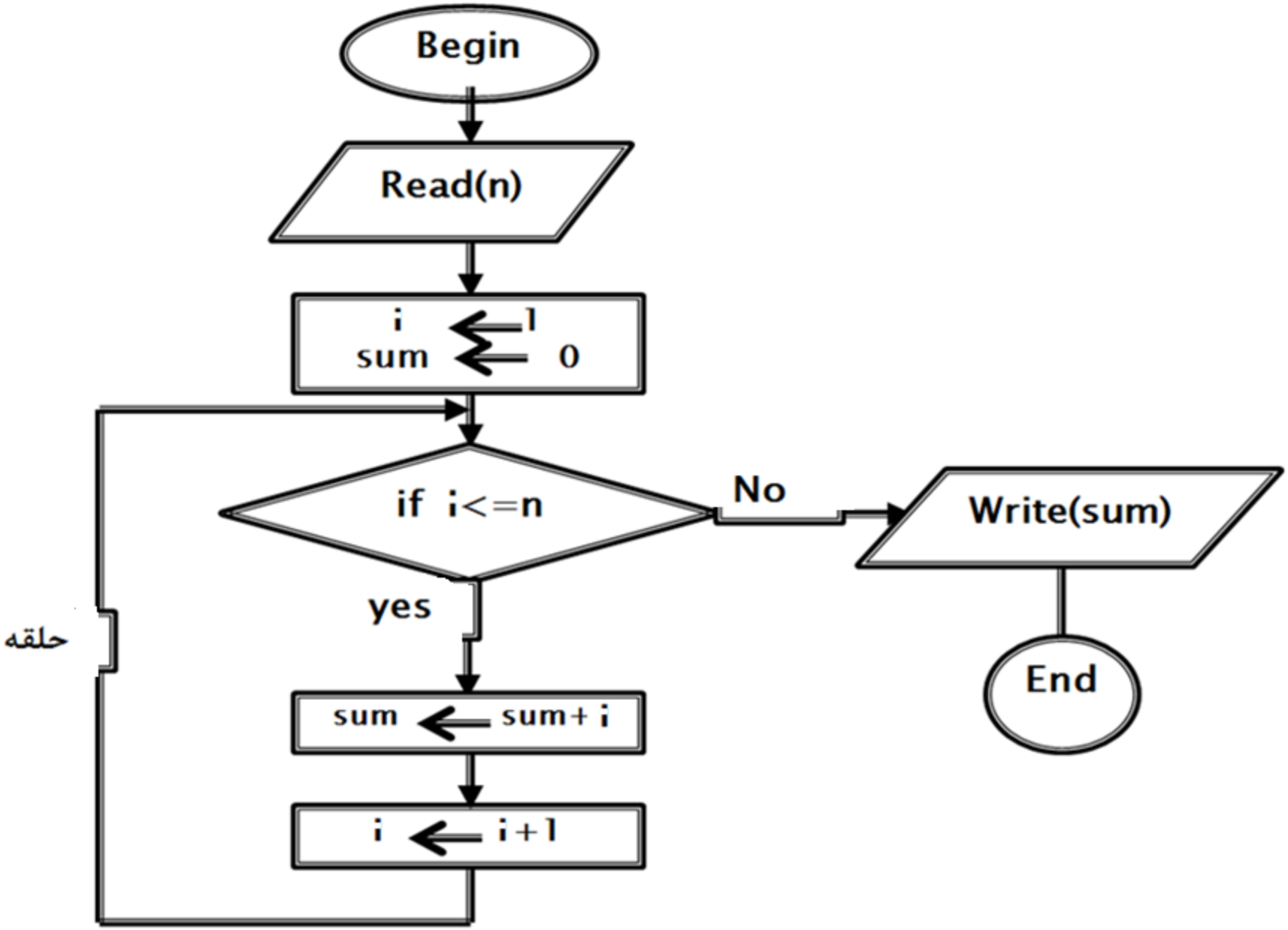
۱ـ اندیس حلقه
۲ـ مقدار اولیه برای اندیس حلقه
۳- مقدار افزاینده برای اندیس حلقه (معمولا یک واحد در هر مرحله)
٤ـ مقدار نهایی (تعداد تکرار حلقه)
٥ـ شرطی برای کنترل تعداد تکرار حلقه

این حلقه‌ها را غالباً با فلوچارت بصورت زیر نمایش می‌دهند:

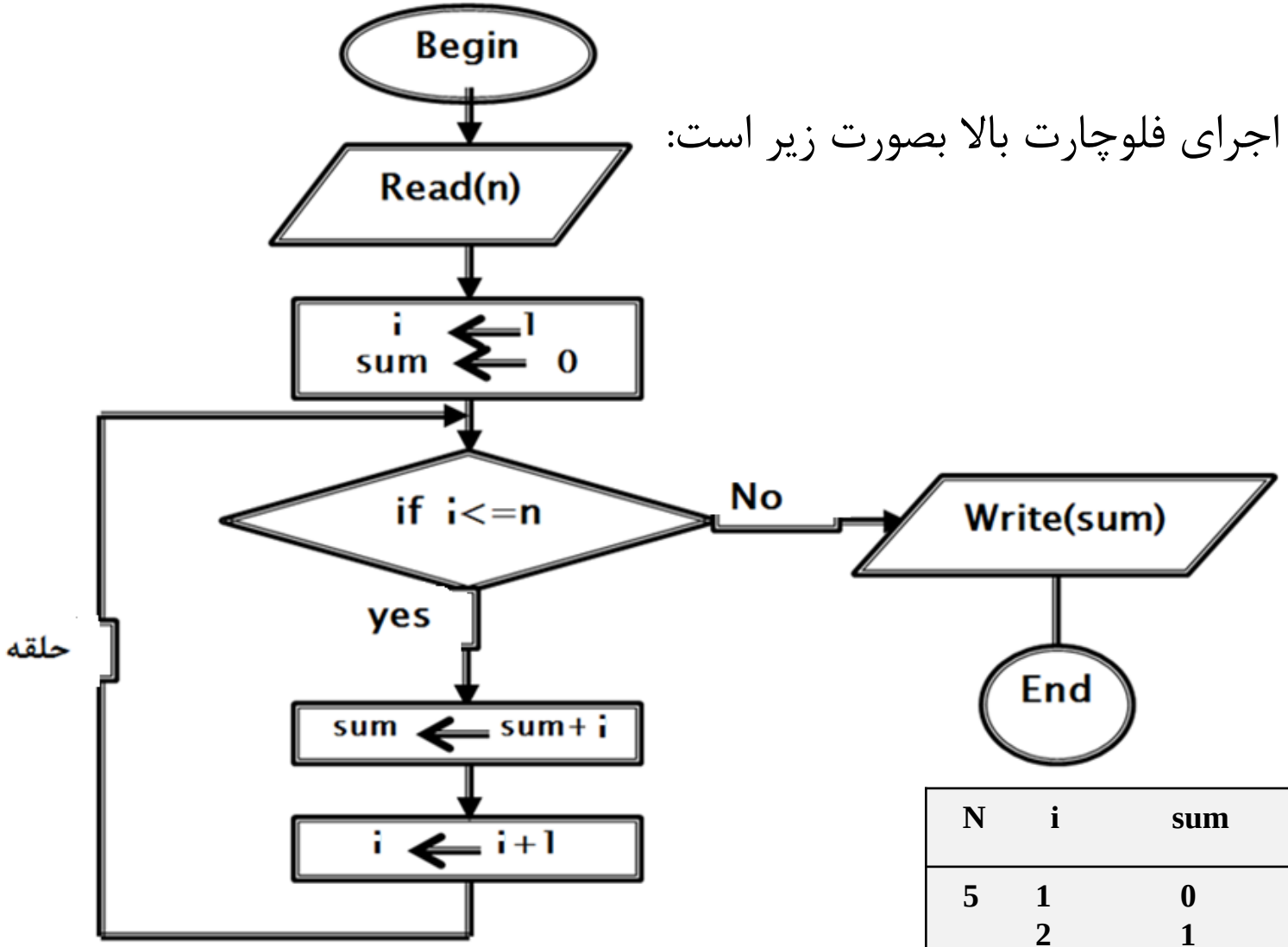


مثال : فلوچارتی رسم نمائید که عدد n را از ورودی دریافت کرده، مجموع اعداد از یک تا n را محاسبه کند.

n	مقدار نهایی
i	اندیس حلقه


Begin
Read(n)
i ← 1
sum ← 0
if i<=n
No
Write(sum)
End
yes
حلقه
sum ← sum+ i
i ← i+1

نمونه اجرای فلوچارت بالا بصورت زیر است:



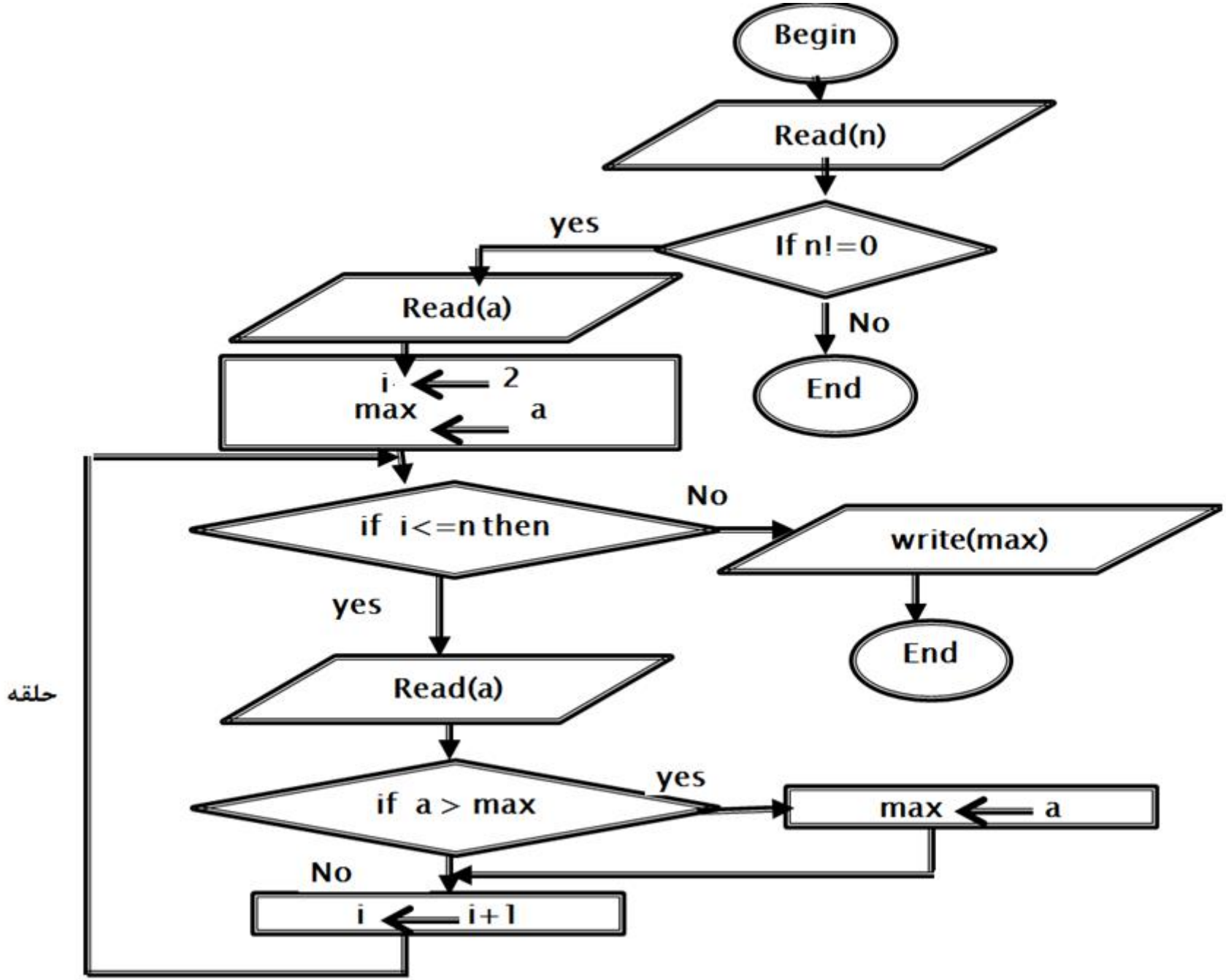
N	i	sum	خروجی
5	1	0	15
	2	1	
	3	3	
	4	6	
	5	10	
	6	15	

مثال : فلوچارتی رسم کنید که n عدد از ورودی دریافت کرده، بزرگترین مقدار از بین n عدد را پیدا کرده در خروجی چاپ نماید.

اندیس حلقه i

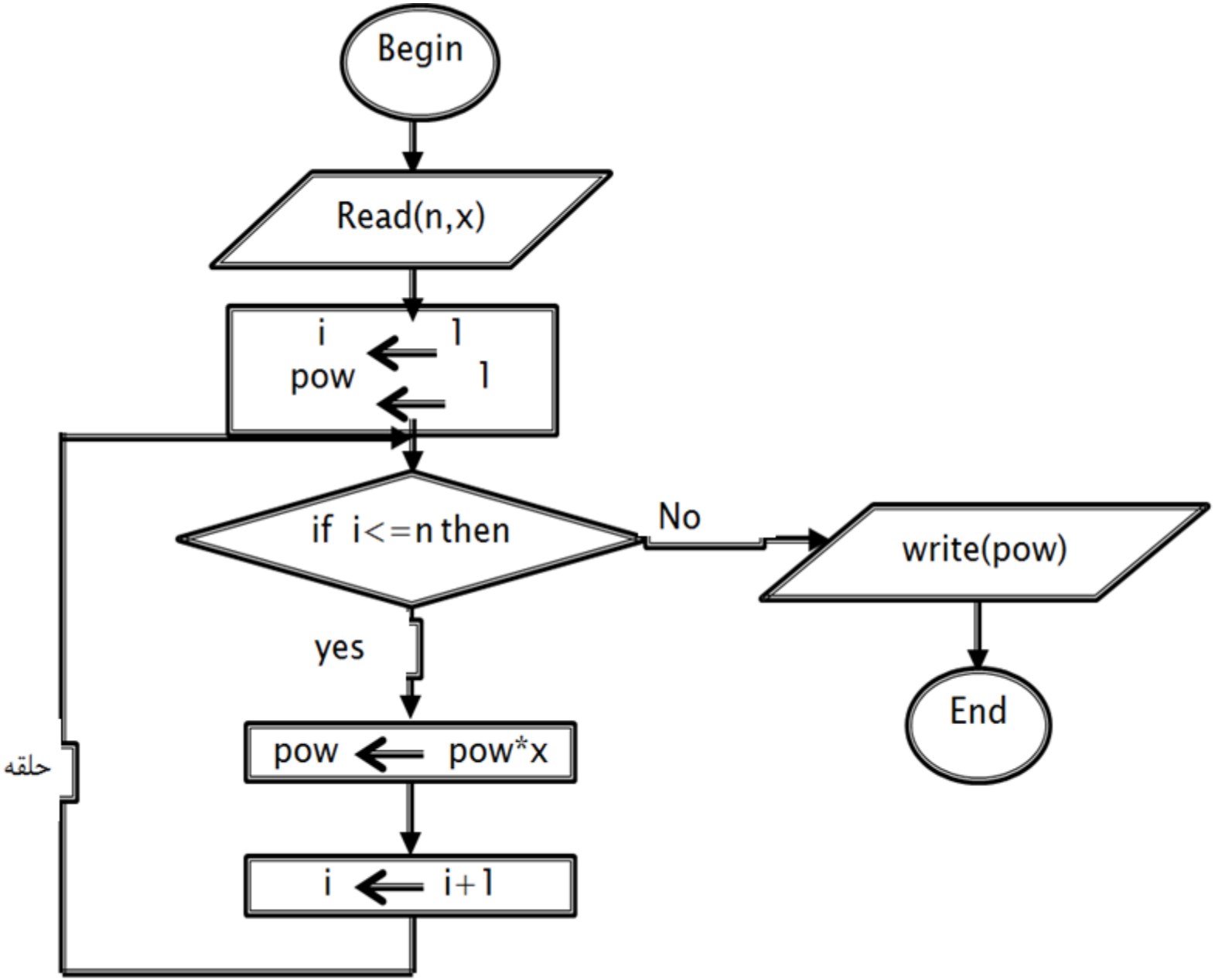
مقدار نهایی n

بزرگترین مقدار Max


Begin
Read(n)
If n!=0
yes
No
End
Read(a)
i ← 2
max ← a
حلقه
if i<=n then
No
write(max)
End
yes
Read(a)
if a > max
yes
max ← a
No
i ← i+1

مثال : فلوچارتی رسم نمائید که x , n، دو عـدد صـحیح مثبـت را از ورودی دریافت کرده سپس x به توان n را محاسبه کند.

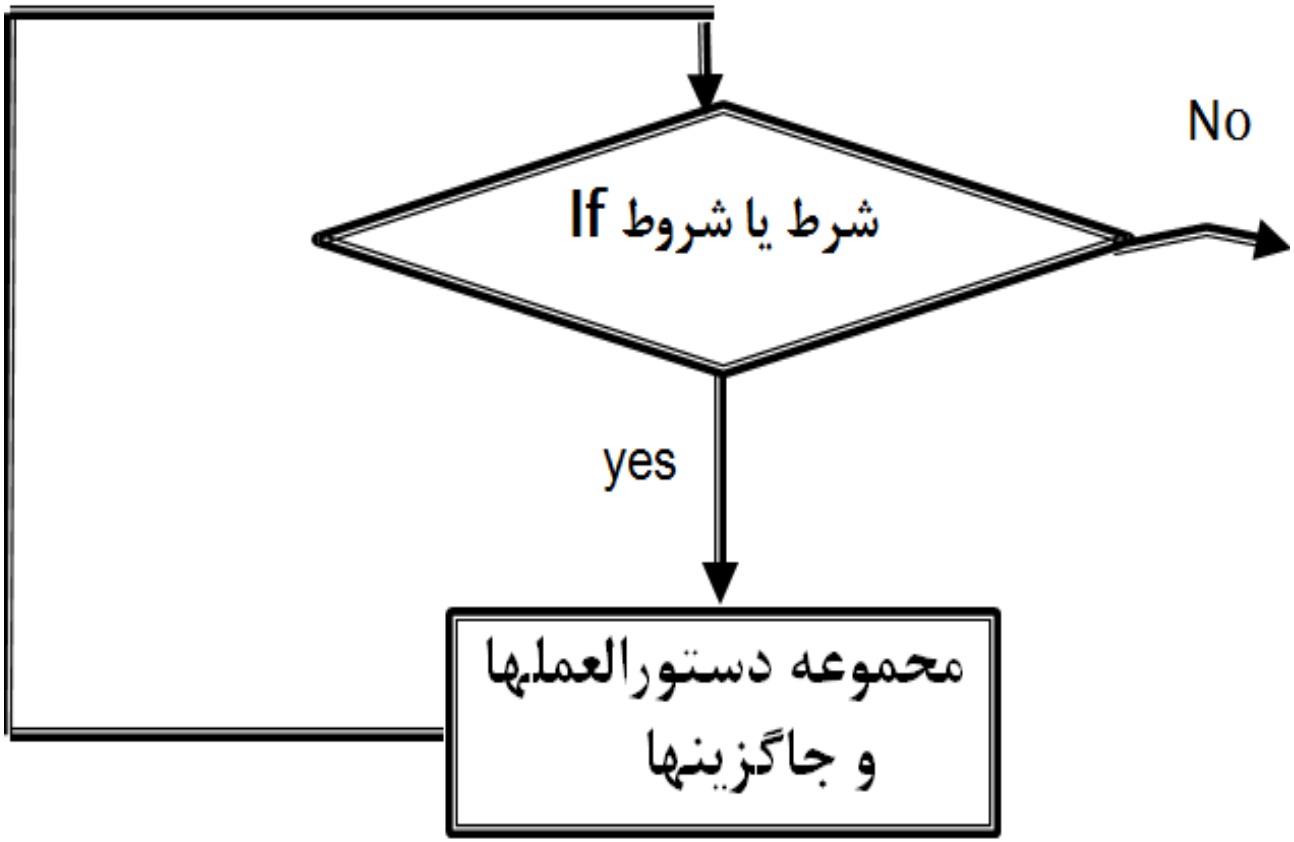
اندیس حلقه	i
مقدار نهایی	n
عدد به توان n	pow


Begin
Read(n,x)
i 1
pow 1
if i<=n then
No
write(pow)
End
yes
pow pow*x
i i+1
حلقه

حلقه‌هایی که تعداد تکرار آن ها مشخص نیست.

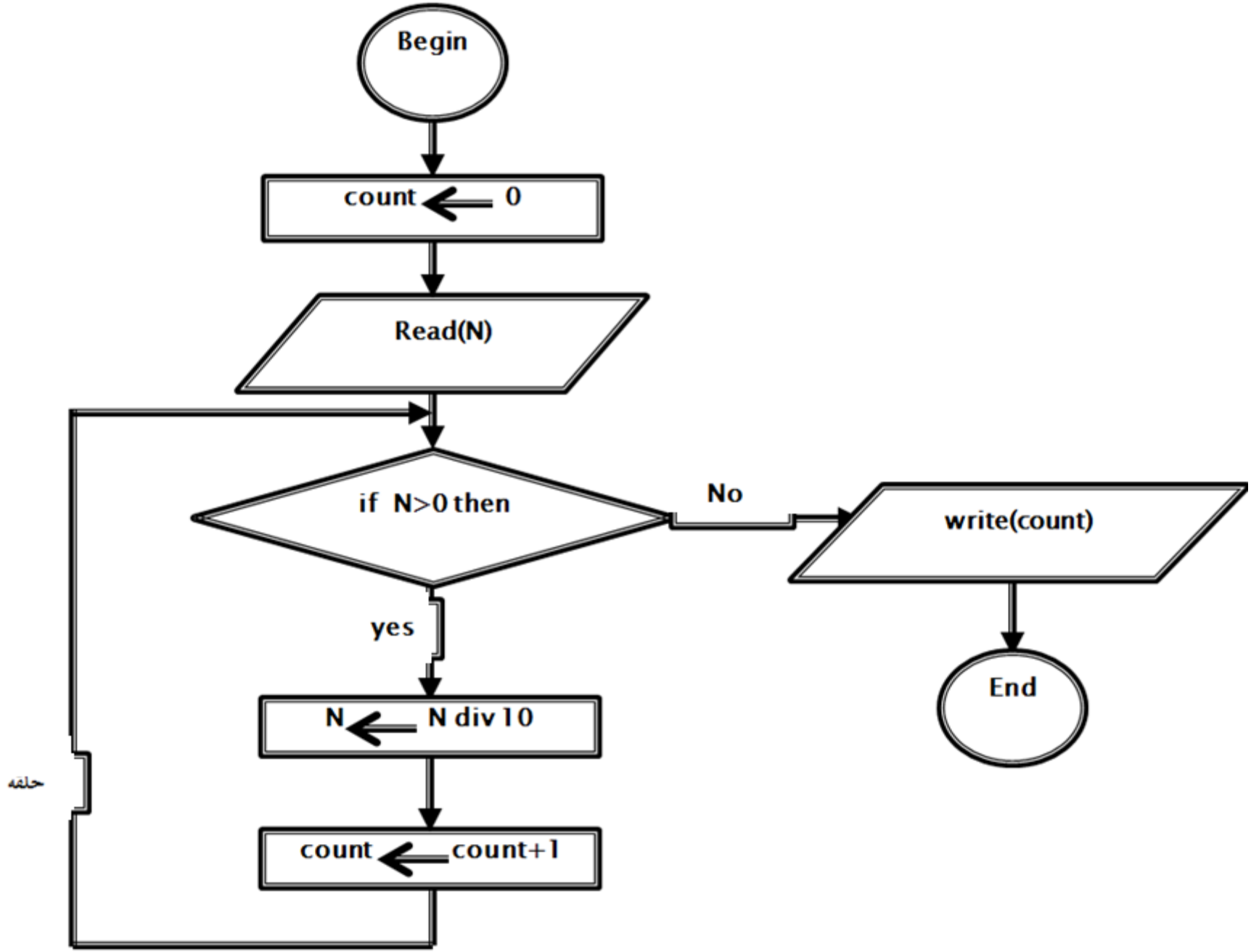
در این حلقه‌ها با توجه به ورودی، تعداد تکرار مشخص می‌شود. و دقیقاً نمی‌توان تعداد تکرار حلقه را بدون ورودی معین کرد. این حلقه ها فقط شامل شرطی هستند که تا زمانیکه برقرار باشد حلقه اجرا می‌شود.

در حالت کلی این نوع حلقه‌ها بصورت زیر نمایش داده می‌شوند:



مثال: فلوچارتی رسم کنید که عددی را از ورودی دریافت کرده سپس تعداد ارقام آن را شمرده در خروجی چاپ نماید.

عدد خوانده شده	N
تعداد ارقام	count

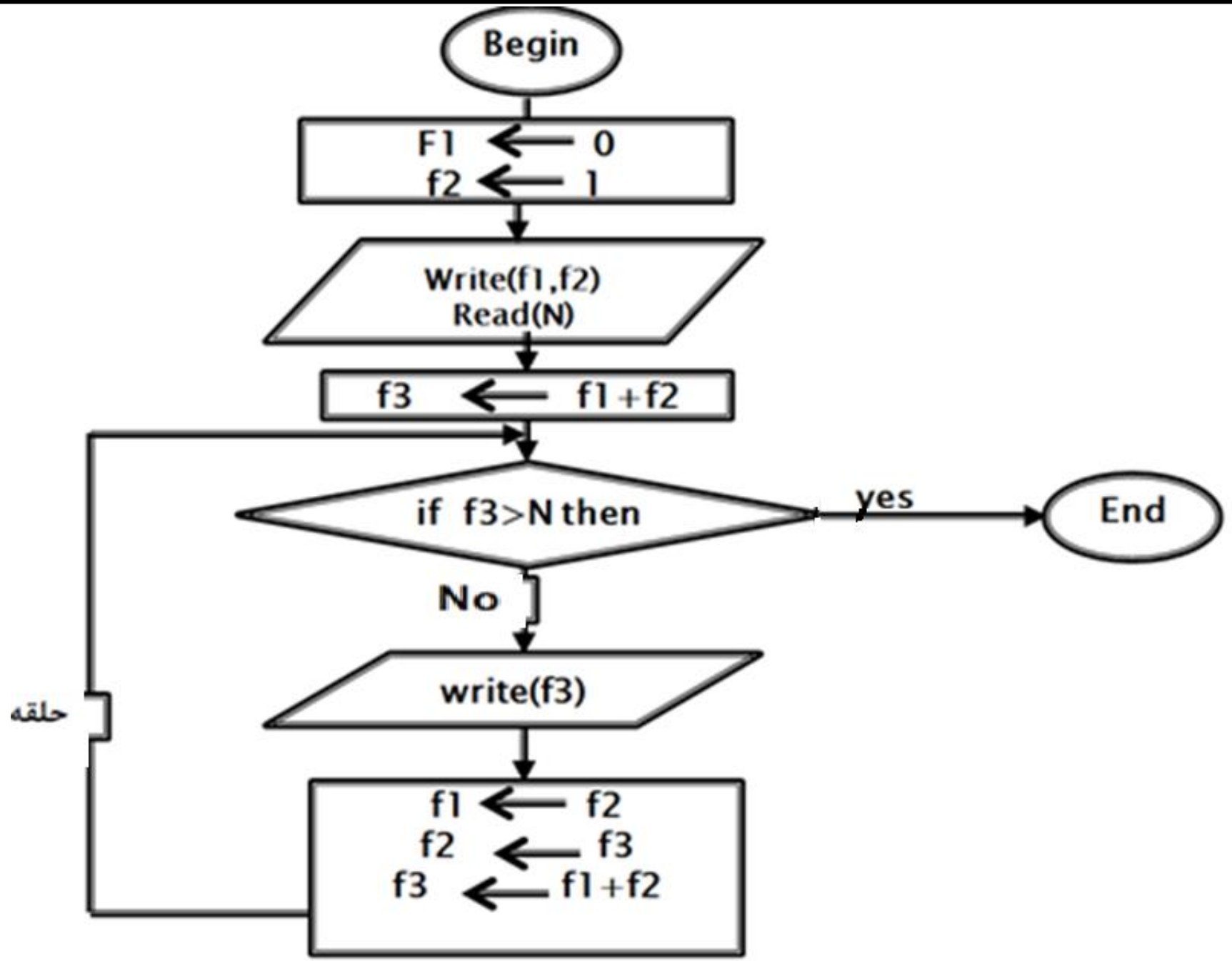

Begin
count ← 0
Read(N)
if N>0 then
No
write(count)
End
yes
N ← N div 10
count ← count+1
حلقه

مثال : فلوچارتی رسم نمائید که عددی از ورودی دریافت کرده، سری فیبوناچی قبل از آن را تولید نماید.

در حالت کلی جملات سری بصورت:

$f_k=f_{k-1}+f_{k-2}$

N	عدد خوانده شده
f_1	جمله اول سری
f_2	جمله دوم سری
f_3	جمله سوم سری


Begin
F1 ← 0
f2 ← 1
Write(f1,f2)
Read(N)
f3 ← f1+f2
if f3>N then
yes
End
No
write(f3)
حلقه
f1 ← f2
f2 ← f3
f3 ← f1+f2

تمرین

۱- فلوچارتی رسم نمائید که عددی از ورودی دریافت کرده، کامل بودن آن را بررسی نماید. (عدد کامل، عددی است که مجموع مقسوم‌علیه‌های آن با خودش برابر باشد.)

۲- فلوچارتی رسم کنید که N را از ورودی دریافت کرده، N جمله سری فیبوناچی را تولید نماید.

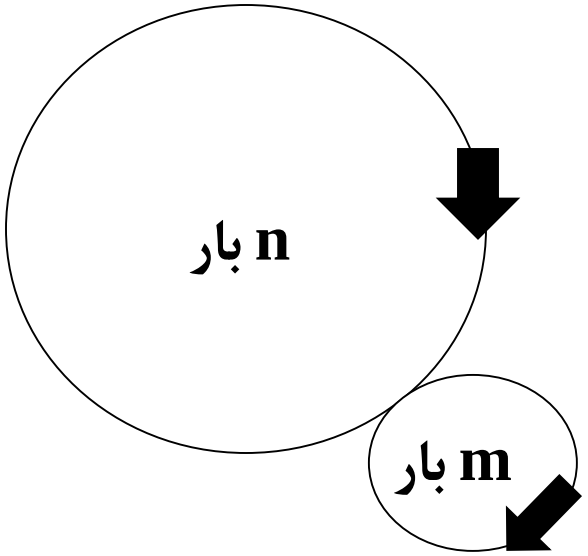
۳ـ فلوچارتی رسم نمائید که دو عدد N , M را از ورودی خوانده، بزرگترین مقسوم‌علیه مشترک دو عدد را محاسبه و چاپ کند.

حلقه‌های تودرتو

الگوریتم‌هایی که تا حال بکار بردیم، فقط شامل یک حلقه بودند. در صورتی که در بسیاری از مسائل ممکن است نیاز به استفاده از چند حلقه در داخل هم باشیم. در این نوع حلقه‌ها باید دقت بیشتری به خرج دهیم، تا مشکلی پیش نیاید. اگر از حلقه‌های نوع اول بصورت تودرتو استفاده کنیم در اینصورت برای هر حلقه شرط نهایی و اندیس اولیه جداگانه باید تعریف کنیم .

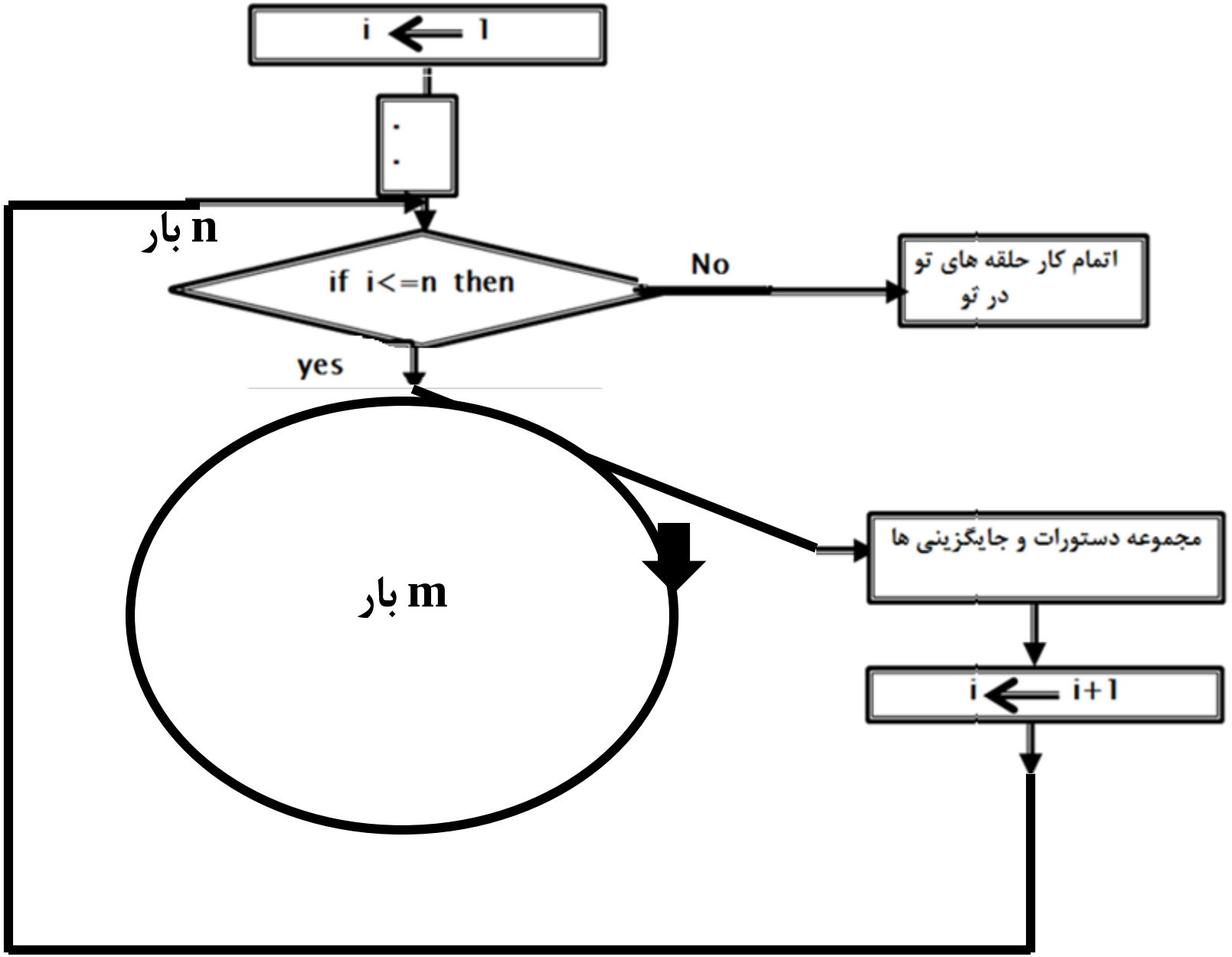
در حلقه‌های تودرتو به ازای یکبار تکرار حلقه اولیه، حلقه داخلی به اندازه مقدار نهایی خود تکرار می‌شود. در کل اگر حلقه اولیه **n** بار تکرار شود و حلقه داخلی **m** بار، در اینصورت کل حلقه :

$$n \times m$$



بار تکرار خواهد شد.

فلوچارت حلقه‌های تودرتو را می‌توان بصورت زیر نشان داد:

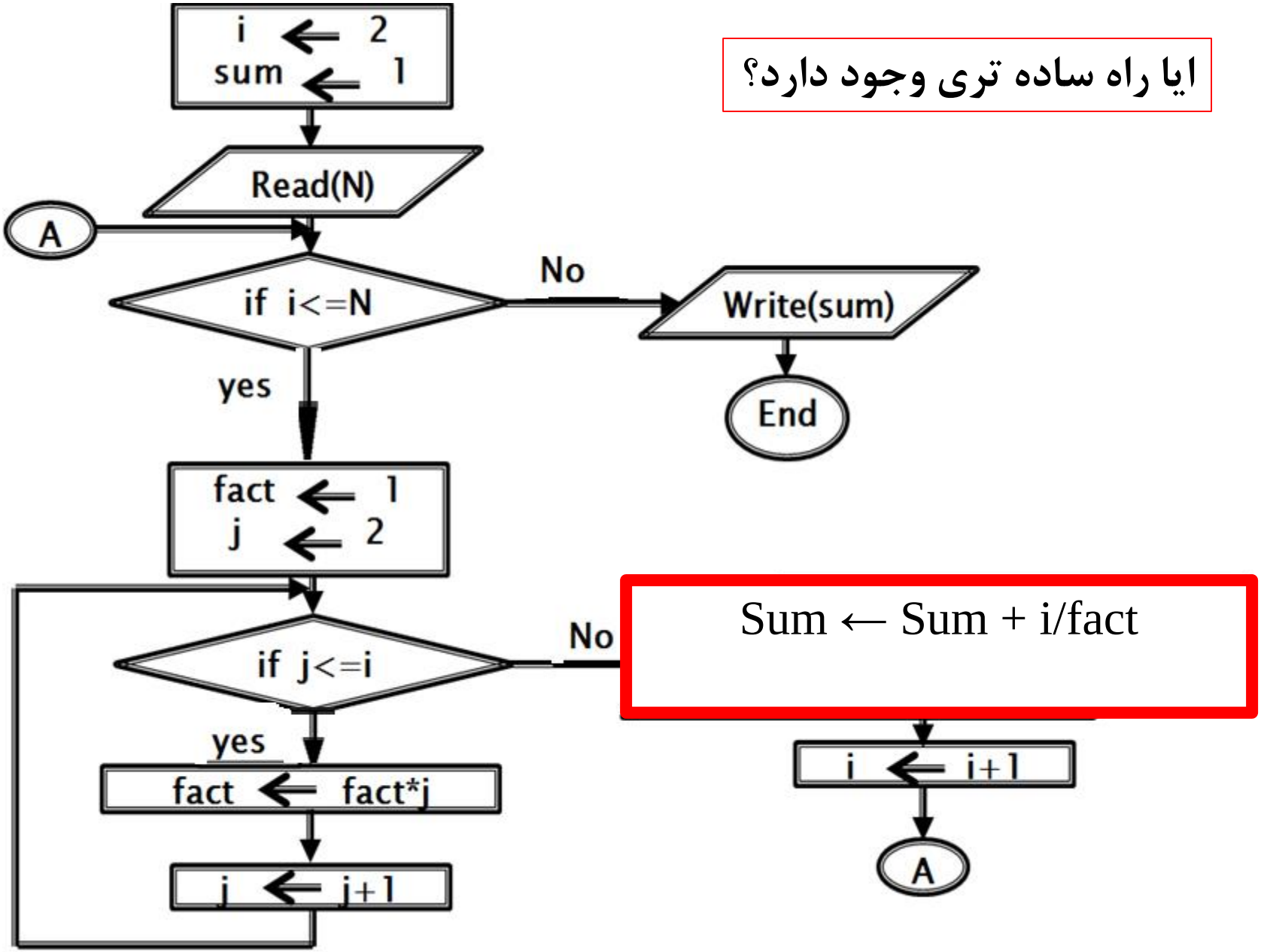


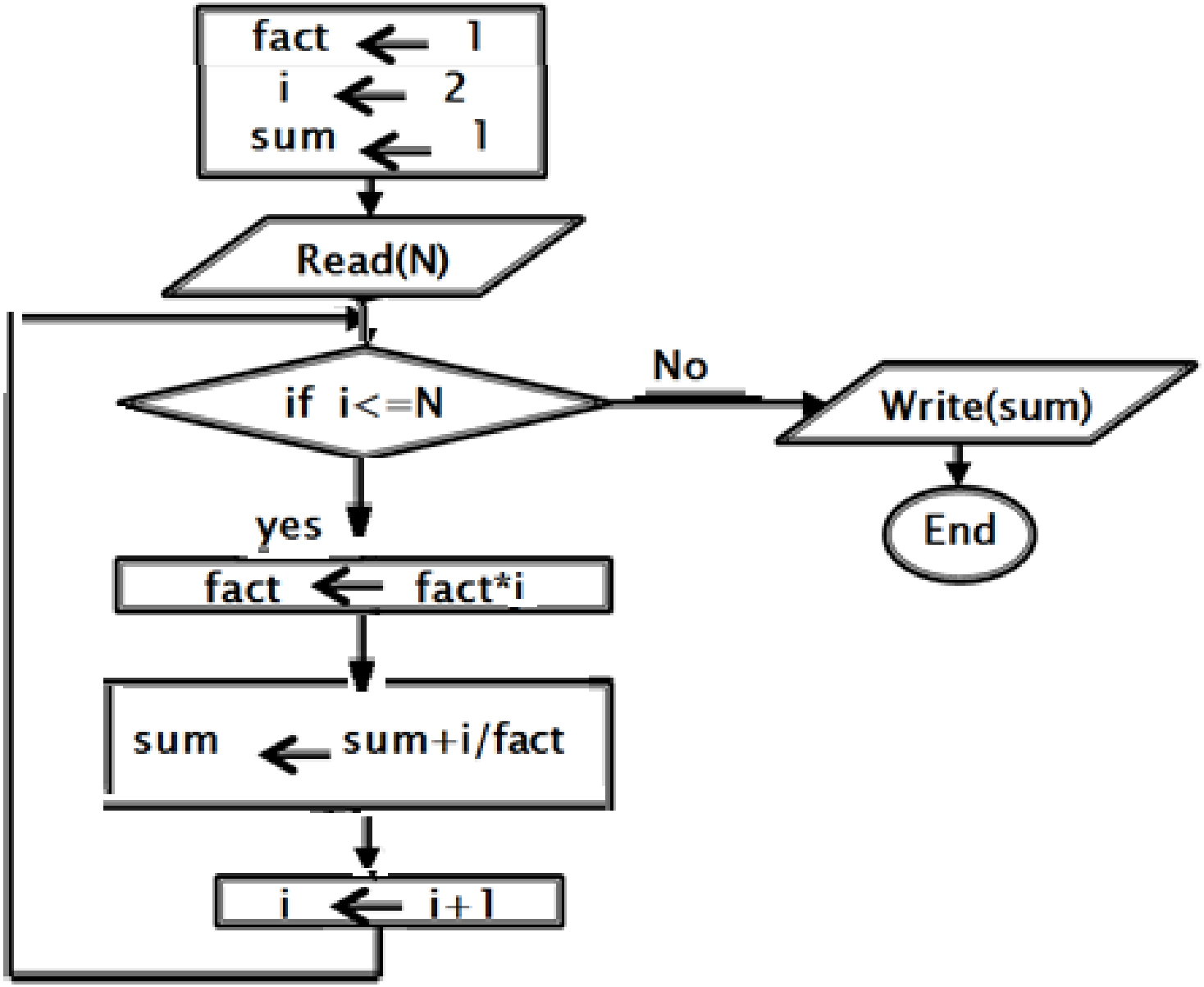
مثال : فلوچارتی رسم نمائید که N را از ورودی دریافت کرده، مجموع سری زیر را محاسبه نماید:

$$S = 1 + \frac{2}{2!} + \frac{3}{3!} + \ldots + \frac{N}{N!}$$

اندیس حلقه اول	i
ورودی	N
محاسبه فاکتوریل	fact
اندیس حلقه داخلی	j
مجموع	Sum

ایا راه ساده تری وجود دارد؟




fact ← 1
i ← 2
sum ← 1
Read(N)
if i<=N
No
Write(sum)
End
yes
fact ← fact*i
sum ← sum+i/fact
i ← i+1

تمرینات آخر فصل

۱- فلوچارتی رسم نمائید که N عدد از ورودی دریافت کرده تعداد اعداد اول و کامل را شمرده در خروجی چاپ نماید.

۲- فلوچارتی رسم نمائید که N , X را از ورودی خوانده مقدار سری زیر را محاسبه کند:

$$S = 1 + \frac{x^{2}}{2!} + \frac{x^{4}}{4!} + \ldots + \frac{x^{n}}{N!}$$

+2

۳- فلوچارتی رسم نمائید که عددی را از ورودی دریافت کرده مقلوب عدد را محاسبه و در خروجی چاپ کند.

۴ـ فلوچارتی رسم کنید که تاریخ تولد شخصی را از ورودی خوانده، سن شخص را با تاریخ روز، محاسبه نموده در خروجی چاپ کند.

۵- فلوچارتی رسم نمائید که (m>n) N ,M را از ورودی دریافت کرده سری فیبوناچی بین N ,M را تولید کرده، در خروجی چاپ کند.

برنامه نویسی به زبان C++

فهرست مطالب

- فصل اول : مقدمات زبان C++
- فصل دوم : ساختار های تصمیم گیری و تکرار
- فصل سوم : سایر ساختار های تکرار
- فصل چهارم : اعداد تصادفی
- فصل پنجم : آرایه ها
- فصل ششم : توابع
- فصل هفتم : ساختارها و اشاره گرها

فصل اول

مقدمات C++

فهرست مطالب فصل اول

۱. تاریخچه مختصر
۲. قانون نامگذاری شناسه ها
۳. متغیر ها
۴. اعلان متغیر
۵. تخصیص مقادیر به متغیر
۶. داده های از نوع کرکتر
۷. کرکتر های مخصوص
۸. رشته ها
۹. نمایش مقادیر داده ها
۱۰. دریافت مقادیر
۱۱. عملگر انتساب
۱۲. عملگر های محاسباتی
۱۳. عملگرهای افزایش و کاهش
۱۴. عملگر sizeof
۱۵. عملگرهای جایگزینی محاسباتی
۱۶. اولویت عملگرها
۱۷. توضیحات (Comments)
۱۸. توابع کتابخانه
۱۹. برنامه در C++

تاریخچه مختصر C++

این زبان در اوائل دهه ۱۹۸۰ توسط Bjarne stroustrup در آزمایشگاه بل طراحی شده. این زبان عملاً توسعه یافته زبان برنامه نویسی C می باشد که امکان نوشتن برنامه‌های ساخت یافته شئ گرا را می‌دهد.

قانون نامگذاری شناسه‌ها

۱) حروف کوچک و بزرگ در نامگذاری شناسه‌ها متفاوت می‌باشند.

بنابراین Xy ، XY ، xY ، xy چهار شناسه متفاوت از نظر C++ می‌باشد.

قانون نامگذاری شناسه‌ها

۲) در نامگذاری شناسه‌ها از حروف الفباء، ارقام وزیر خط (underscore) استفاده می‌شود و حداکثر طول شناسه ۳۱ می‌باشد و شناسه بایستی با یک رقم شروع نگردد.

قانون نامگذاری شناسه‌ها

۳) برای نامگذاری شناسه‌ها از کلمات کلیدی نبایستی استفاده نمود. در زیر بعضی از کلمات کلیدی داده شده است.

And	Sizeof	then	xor	Template
Float	False	Friend	While	continue
extern	Private	Switch	Default	Const
delete	typedef	if	this	Virtual

لیست کامل کلمات کلیدی

متغیرها

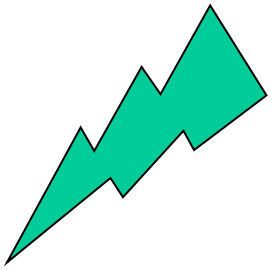
متغیر، مکانی در حافظه اصلی کامپیوتر می‌باشد که در آنجا یک مقدار را می‌توان ذخیره و در برنامه از آن استفاده نمود. قانون نامگذاری متغیرها همان قانون نامگذاری شناسه‌ها می‌باشد.

در اسلاید بعد به انواع داده ها اشاره می شود.

انواع داده ها

نوع داده	مقادیر	حافظه لازم
int	32767 تا 32768–	۲ بایت
unsigned int	0 تا 65535	۲بایت
long int	2147483647 تا 2147483648–	۴ بایت
unsigned long int	0 تا 4294967295	۴ بایت
char	یک کارکتر	۱بایت
unsigned char	127 تا 128–	۱ بایت
float	3.4e38 تا 1.2e-38	۴ بایت
double	1.8e308 تا 2.2e-308	۸ بایت

اعلان متغیرها



قبل از آنکه در برنامه به متغیرها مقـداری تخصـیص داده شود و از آنها استفاده گردد بایستی آنها را در برنامه اعلان نمود.

در اسلاید بعد مثال هایی از اعلان متغیر ذکر شده است.

چند مثال از اعلان متغیر ها :

- ✓ برای اعلان متغیرx از نوع int :

```
int   x;
```

- ✓ برای اعلان متغیرهای p و q را از نوع float که هر کدام چهار بایت از حافظه را اشغال می‌کنند :

```
float  p , q  ;
```

- ✓ برای اعلان متغیر next از نوع کرکتر که می‌توان یکی از ۲۵۶ کرکتر را به آن تخصیص داد و یک بایت را اشغال می‌کند.

```
char next ;
```

تخصیص مقادیر به متغیرها

با استفاده از عملگر = می‌توان به متغیرها مقدار اولیه تخصیص نمود.

در اسلاید بعد مثال هایی از اعلان متغیر ذکر شده است.

مثال :

- در دستورالعمل
```
int   x=26;
```
X را از نوع int با مقدار اولیه 26 اعلان نموده .

- در دستورالعمل
```
long int   a=67000 ,
b=260;
```
متغیرهای b و a را از نوع long int تعریف نموده با مقادیر بترتیب 260 و 67000.

داده‌های از نوع کرکتر

برای نمایش داده‌های از نوع **char** در حافظه کامپیوتر از جدول **ASCII** استفاده می‌شود. جدول اسکی به هر یک از ۲۵۶ کرکتر یک عدد منحصر بفرد بین ۰ تا ۲۵۵ تخصیص می‌دهد.

کرکترهای مخصوص

کامپیلر C++ بعضی از کرکترهای مخصوص که در برنامه می‌توان از آنها برای فرمت بندی استفاده کرد را تشخیص می‌دهد. تعدادی از این کرکترهای مخصوص به همراه کاربرد آنها در اسلاید بعد آورده شده است .

کرکترهای مخصوص

\n	Newline
\t	Tab
\b	Backspace
\a	Beep sound
\”	Double quote
\’	Single quote
\0	Null character
\?	Question mark
\\	Back slash

بعنوان مثال از کرکتر \a می‌توان برای ایجاد صدای beep استفاده نمود.

```
char        x = '\a' ;
```

رشته‌ها

رشته یا string عبارتست از دنباله‌ای از کرکترها که بین " " قرار داده می‌شود. در حافظه کامپیوتر انتهای رشته‌ها بوسیله 0\ ختم می‌گردد.

در اسلاید بعد به دو مثال دقت نمایید.

مثال ۱ :

"BOOK STORE" یک رشته ده کرکتـری می‌باشـد که با توجه به کرکتر 0\ که به انتهای آن در حافظه اضافه می شود جمعــاً یازده بایت را اشغال می‌کند.

مثال ۲ :

دقت نمایید که "w" یک رشته می‌باشد که دو بایت از حافظه را اشغال می‌کند در حالیکه 'w' یک کرکتر می‌باشد که یک بایت از حافظه را اشغال می‌نماید.

نمایش مقادیر داده‌ها

برای نمایش داده‌ها بر روی صفحه مانتور از cout که بدنبال آن عملگر درج یعنی >>قید شده باشد استفاده می‌گردد. بایستی توجه داشت که دوکرکتر >پشت سر هم توسط C++ بصورت یک کرکتر تلقی می‌گردد.

مثال :

- ✓ برای نمایش پیغام good morning بر روی صفحه نمایش :

```
cout << "good morning";
```

- ✓ برای نمایش مقدار متغیر x بر روی صفحه نمایش :

```
cout << x ;
```

دریافت مقادیر متغیرها

به منظور دریافت مقادیر برای متغیرها در ضمن اجرای برنامه از صفحه کلید، از cin که بدنبال آن عملگر استخراج یعنی >> قید شده باشد می‌توان استفاده نمود.

مثال :

```
int x;
cout << "Enter a number:" ;
cin >> x;
```

عملگر انتساب

عملگر انتساب = می‌باشد که باعث می‌گردد مقدار عبارت در طرف راست این عملگر ارزیابی شده و در متغیر طرف چپ آن قرار گیرد.

مثال :

```
x=a+b;
x=35 ;
x=y=z=26 ;
```

از عملگرهای انتساب چندگانه نیز می‌توان استفاده نمود. که مقدار سه متغیر z و y و x برابر با 26 می شود.

عملگرهای محاسباتی

در C++ پنج عملگر محاسباتی وجود دارد که عبارتند از :

+	جمع
–	تفریق
*	ضرب
/	تقسیم
٪	باقیمانده

این عملگرها دو تائی می‌باشند زیرا روی دو عملوند عمل می‌نمایند. از طرف دیگر عملگرهای + و – رامی‌توان بعنوان عملگرهای یکتائی نیز در نظر گرفت.

مثال ۱ :

در حالتی که هر دو عملوند عملگرهـای ٪ ، / ، * ، + ، – از نـوع صحیح باشد نتیجه عمل از نوع صحیح می‌باشد.

عبارت	نتیجه
5 + 2	7
5 * 2	10
5 – 2	3
5 % 2	1
5 / 2	2

مثال ۲ :

در صورتیکه حداقل یکی از عملوندهای عملگرهای / ، * ، – ، + از نوع اعشاری باشد نتیجه عمل از نوع اعشاری می‌باشد.

عبارت	نتیجه
5. 0 +2	7. 0
5* 2. 0	10.0
5. 0 / 2	2.5
5. 0 -2	3. 0
5. 0 / 2. 0	2.5

عملگرهای افزایش و کاهش

در C++ ، افزایش یک واحد به مقدار یک متغیر از نوع صحیح را افزایش و بطور مشابه کاهش یک واحد از مقدار یک متغیر از نوع صحیح را کاهش می‌نامند..

عملگرهای افزایش و کاهش

عملگر کاهش را با – – و عملگر افزایش را بـــا ++ نمـــایش می‌دهنـــد. چـــون عملگرهـــای ++ و – – فقـــط روی یـــک عملوند اثر دارند این دو عملگر نیز جـــزء عملگرهای یکتائی می‌باشند.

مثال :

سه دستور العمل :

```
++x;
x++;
x= x+1;
```

معادل می‌باشند و بطریق مشابه سه دستورالعمل زیر نیز معادل می باشند.

```
- - y ;
y= y-1;
y - - ;
```

از عملگرهای ++ و -- می‌توان بدو صورت پیشوندی و پسوندی استفاده نمود.
در دستورالعمل‌های پیچیده عملگر پیشوندی قبل از انتساب ارزیابی می شود و عملگر پسوندی بعد از انتساب ارزیابی می‌شود.

مثال :

```
int x=5;
y=++x * 2;
```

پس از اجرای دستورالعملهای فوق :

```
y=12
X=6
```

```
int  x=5;
y=x++ * 2;
```

پس از اجرای دستورالعملهای فوق :

```
y=10
X=6
```

عملگر sizeof

Sizeof ازعملگرهای یکتائی می باشد و مشخص کننده تعداد بایت هائی است که یک نوع داده اشغال می‌کند.

مثال :

```
int   x;
cout << sizeof x ;
```

مقدار ۲ نمایش داده می‌شود .

```
cout << sizeof(float) ;
```

مقدار ۴ نمایش داده می شود.

عملگرهای جایگزینی محاسباتی

برای ساده‌تر نوشتن عبارتها در C++ ، می‌توان از عملگرهای جایگزینی محاسبـــــاتی استفاده نمود.

+= -= *= /= %=

مثال:

```
int a=1;
a = a +1;
```

⟺

```
int a=1;
a += 1;
```

اولویت عملگرها

ارزیابی مقدار یک عبارت ریاضی براساس جدول اولویت عملگرها انجام می‌گردد. در ذیل جدول اولویت عملگرها براساس بترتیب از بیشترین اولویت به کمترین اولویت داده شده است.

چپ به راست	پرانتزها	( )
راست به چپ	عملگرهای یکتایی	sizeof ++ -- + -
چپ به راست	عملگرهای ضرب و تقسیم و باقیمانده	* / %
چپ به راست	عملگرهای جمع و تفریق	+ –
چپ به راست	عملگرهای درج و استخراج	>> <<
راست به چپ	عملگرهای جایگزینی و انتساب	= += -= *= /= %=

مثال ۱ :

(5+2) *(6+2*2)/2

با توجه به جدول اولویت عملگرها داریم که

چپ به راست	پرانتزها
راست به چپ	عملگرهای یکتایی
چپ به راست	عملگرهای ضرب و تقسیم و باقیمانده
چپ به راست	عملگرهای جمع و تفریق
چپ به راست	عملگرهای درج و استخراج
راست به چپ	عملگرهای جایگزینی و انتساب

7 *(6+2*2)/2

7*(6+4)/2

7* 10 /2

70 /2

35

مثال ۲ :

```
int a=6 , b=2, c=8, d=12;
d=a++ * b/c ++;
cout << d << c << b << a;
```

خروجی :

چپ به راست	پرانتزها
راست به چپ	عملگرهای یکتایی
چپ به راست	عملگرهای ضرب و تقسیم و باقیمانده
چپ به راست	عملگرهای جمع و تفریق
چپ به راست	عملگرهای درج و استخراج
راست به چپ	عملگرهای جایگزینی و انتساب

توضیحات (Comments)

توضیحات در برنامه باعث خوانائی بیشتر و درک بهتر برنامه می شود. بنابراین توصیه بر آن است که حتی الامکان در برنامه‌ها از توضیحات استفاده نمائیم. در C++، توضیحات بدو صورت انجام می‌گیرد که در اسلایدهای بعد به آن اشاره شده است.

توضیحات (Comments)

الف: این نوع توضیح بوسیله // انجام می‌شود. که کامپیوتر هر چیزی را کــه بعــد از // قــرار داده شود تا انتهای آن خط اغماض می‌نماید.

مثال :

```
c=a+b;      //c is equal to sum of a  and b
```

ب: توضیح نوع دوم با /* شروع شده و به */ ختم می‌شود و هر چیزی که بین* / و /* قــرار گیــرد اغماض می‌نماید .

مثال :

```
/ * this is a program
to calcufate sum of
n integer numbers */
```

توابع کتابخانه

زبان C++ مجهز به تعدادی توابع کتابخانه می‌باشد. بعنوان مثال تعدادی توابع کتابخانه برای عملیات ورودی و خروجی وجود دارند. معمولاً توابع کتابخانه مشابه ، بصورت برنامه‌های هدف (برنامه ترجمه شده بزبان ماشین) در قالب فایلهای کتابخانه دسته بندی و مورد استفاده قرا رمی‌گیرند. این فایلها را فایلهای header می‌نامند و دارای پسوند h. می‌باشند.

نحوه استفاده از توابع کتابخانه ای

برای استفاده از توابع کتابخانه خاصی بایستـــــی نام فایل header آنرا در ابتدای برنامه در دستور #include قرار دهیم.

#include < اسم فایل header >

تابع	نوع	شرح	فایل هیدر
abs(i)	*int*	قدرمطلق i	stdlib.h
cos(d)	*double*	کسینوس d	math.h
exp(d)	*double*	e^x	math.h
log(d)	*double*	$\log_e d$	math.h
log10(d)	*double*	$\mathrm{Log}_{10} d$	math.h
sin(d)	*double*	سینوس d	math.h
sqrt(d)	*double*	جذر d	math.h
strlen(s)	*int*	تعداد کرکترهای رشته s	string.h
tan(d)	*double*	تانژانت d	math.h
toascii(c)	*int*	کداسکی کرکترc	stdlib.h
tolower(c)	*int*	تبدیل به حروف کوچک	stdlib.h
toupper(c)	*int*	تبدیل به حرف بزرگ	stdlib.h

برنامه در C++

اکنون باتوجه به مطالب گفته شده قادر خواهیم بود که تعدادی برنامه ساده و کوچک به زبان C++ بنویسیم. برای نوشتن برنامه بایستی دستورالعملها را در تابع main( ) قرار دهیم و برای اینکار می‌توان به یکی از دو طریقـــی کـــه در اسلایدهای بعد آمده است ، عمل نمود.

روش اول :

```
#include      <       >
int main( )
{
۱ دستورالعمل ;
۲ دستورالعمل ;
.
.
.
n  دستورالعمل;
return 0 ;
}
```

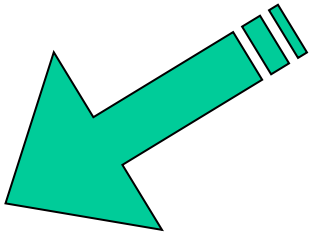
روش دوم :

```
#include     <       >
void main( )
{
۱ دستورالعمل ;
۲ دستورالعمل ;
.
.
.
n  دستورالعمل;
}
```

❑ error:

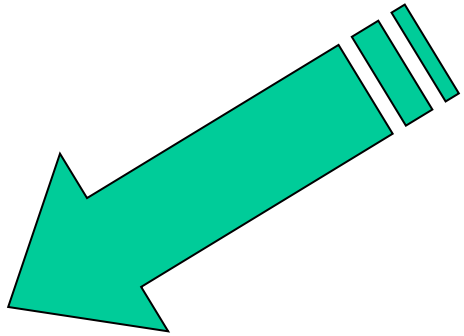
- به خطاهای برنامه نویسی error می گویند. ما در برنامه نویسی دو نوع خطا داریم:
- خطاهای دستوری (syntax error)
- خطاهای منطقی (logical error).

برنامه ای که پیغام C++ is an object oriented language را روی صفحه مانیتور نمایش می دهد.



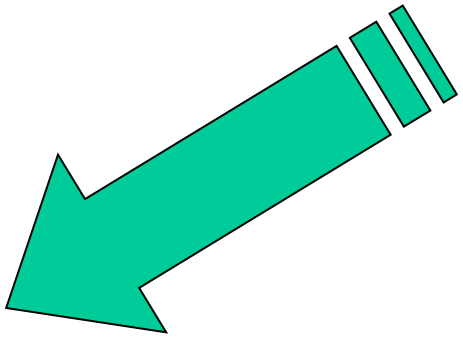
```
#include    <iostream.h>
int main( )
{
cout <<"C++ is an object oriented language \n" ;
return 0 ;
}
```

برنامه زیر یک حرف انگلیسی کوچک را گرفته به حرف بزرگ تبدیل می‌نماید.



```
#include    <iostream.h>
#include    <stdlib. h>
int main( )
{
 char c1 , c2;
cout << "Enter a lowercase letter:"
cin >> c1;
c2 = toupper(c1);
cout << c2 << endl;
return 0; }
```

دو عدد از نوع اعشاری را گرفته مجموع و حاصلضرب آنها را محاسبه و نمایش می‌دهد.



```
#include    <iostream.h>
int main( )
{
float         x,y,s,p ;
cin >> x >> y ;
s= x+y ;
p=x*y;
cout << s <<endl << p;
return 0 ;
}
```

فصل دوم

ساختارهای تصمیم گیری و تکرار

فهرست مطالب فصل دوم

۱. عملگر های رابطه ای
۲. عملگر شرطی
۳. دستورالعمل شرطی
۴. عملگر کاما
۵. عملگر های منطقی
۶. دستورالعمل For

عملگرهای رابطه ای

از این عملگرها برای تعیین اینکه آیا دو عدد با هم معادلند یا یکی از دیگری بزرگتر یا کوچکتر می‌باشد استفاده می‌گردد. عملگرهای رابطه‌ای عبارتند از:

مساوي	= =
مخالف	! =
بزرگتر	>
بزرگتر یا مساوي	> =
کوچکتر	<
کوچکتر یا مساوي	<=

عملگر شرطی

شکل کلی عملگر شرطی بصورت زیر می‌باشد:

```
expression _ test ? expression _ true : expression _ false
```

عملگر شرطی تنها عملگری در C++ می‌باشد که دارای سه عملوند می‌باشد.

مثال ۱ :

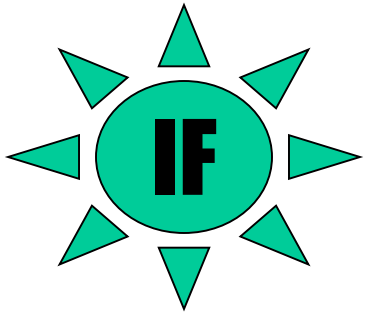
```
int x=10,y=20,b;
b=(x>y) ? x : y ;
```

این دو دستور العمل باعث می شوند که ماکزیمم مقادیر y و x در b قرار بگیرد.

مثال ۲ :

```
x>=10 ? cout << "passed" : cout << "failed" ;
```

اگر مقدار x بزرگتر یا مساوی ده باشد رشتهٔ passed در غیر اینصورت رشته failedنمایش داده می شود.



دستورالعمل شرطی

توسط این دستور شرطی را تست نموده و بسته به آنکه شرط درست یـا غلـط باشـد عکس‌العمـل خاصی را نشان دهیم.

```
if (عبارت)
{
        دستورالعمل 1 ;
            .
        دستورالعمل n;
}
 else
{
        دستورالعمل 1 ;
            .
        دستورالعمل n;
 }
```

مثال ١ :

```
if(x != y)
{
cout << x ;
++ x ;
}
else
        {
cout << y ;
- - y ;
         }
```

مثال ۲:

برنامه زیر یک عدد اعشاری را از ورودی گرفته جذر آن را محاسبه می‌نماید.

```
#include    <iostream.h>
#include    <math . h>
int main( )
{
float  x,s;
cin >> x ;
if ( x < 0 )
cout << " x is negative" << endl ;
else
{
s = sqrt(x) ;
cout << s << endl ;
                    }
return 0;
}
```

عملگر کاما

تعدادی عبارت را می‌توان با کاما بهم متصل نمود و تشکیل یک عبارت پیچیده‌تری را داد. این عبارتها به ترتیب از چپ به راست ارزیابی شده و مقدار عبارت معادل عبارت n می‌باشد.

(عبارت 1 , عبارت 2 , عبارت 3 , ... , عبارت n)

مثال :

اگر داشته باشیم int a=2 , b=4 , c=5 ; عبارت زیر را در نظر بگیرید:

(++ a , a+b, ++ c, c+b)

مقدار عبارت برابر است با b+c که معادل 10 می‌باشد.

عملگرهای منطقی

با استفاده از عملگرهای منطقی می‌توان شرط های ترکیبی در برنامه ایجاد نمود.
عملگرهای منطقی عبارتست از :

AND

OR

NOT

که در C++ به ترتیب بصورت زیر نشان داده می شود.

&&

||

!

جدول درستی سه عملگر شرطی

And

a	b	a && b
true	true	True
true	false	False
false	true	False
false	false	False

Or

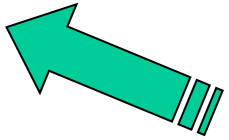
a	b	a || b
true	true	True
true	false	True
false	true	True
false	false	False

Not

a	!a
true	False
false	True

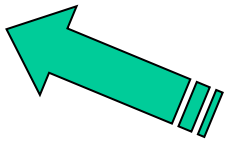
چند مثال :

```
if ((x= = 5) ||(y != 0))
    cout << x << endl ;
```



اگر x برابر با 5 یا y مخالف صفر باشد مقدار x نمایش داده شود .

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



اگر مقدار x مخالف صفر باشد، آنگاه x برابر با صفر شود .

برنامه زیر طول سه پاره‌خط را از ورودی گرفته مشخص می‌نماید که آیا تشکیل یک مثلث می دهد یا خیر؟

```
#include       < iostream.h >
int main( )
{
float  a, b, c;
cout << "Enter three real numbers" << endl ;
cin >> a >> b >> c;
if(( a < b + c) &&(b < a+c) &&(c < a+b))
cout << "It is a triangle" ;
else
cout << "Not a triangle" ;
return  0 ;
}
```

دستورالعمل For

از دستور العمل for برای تکرار دستورالعمل ها استفاده می شود. شکل کلی دستور for بصورت زیر می‌باشد:

```
for (عبارت 1  ;  عبارت 2  ;  عبارت 3)
{
 دستورالعمل 1 ;
 دستورالعمل 2 ;
 .
 .
 .
 دستورالعمل n;
}
```

ساختار for

معرفی کنترل گر حلقه;

for (مقداردهی اولیه کنترل گر حلقه; شرط حلقه; گام حرکت)

{

مجموعه دستورات بدنه حلقه ;

}

```
int i;

for  (i=1;   i<=3;    i++)
    {
   cout << “hello \n”;
     }
```

مثال

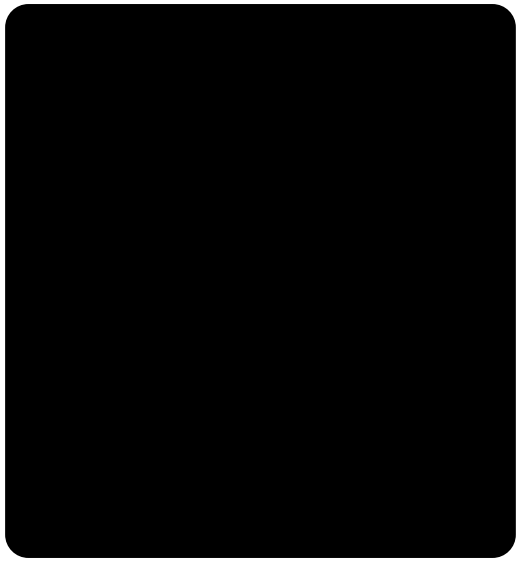
```
void main()
{
cout << “hello \n”;
cout << “hello \n”;
cout << “hello \n”;
}
```

```
void main()
{
int i;
for  (i=1;   i<=3;    i++)
     {
     cout << “hello \n”;
    }
}
```

نحوه اجرای دستور for

```
int i;

for  (i=1;   i<=3;    i++)
     {
     cout << “hello \n”;
    }
```

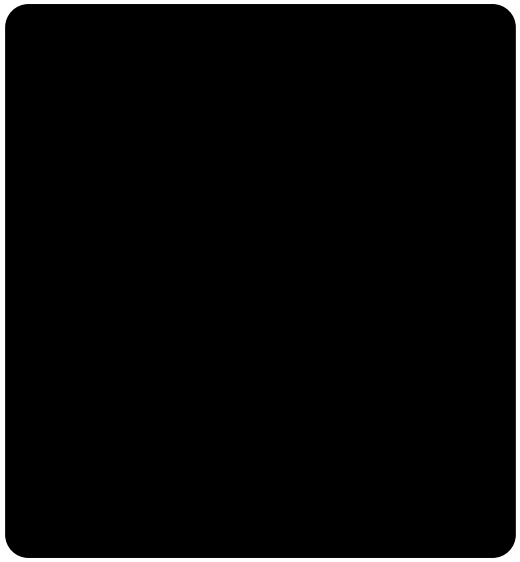
i =



نحوه اجرای دستور for

```
int i;

for  (i=1;   i<=3;    i++)
     {
     cout << “hello \n”;
    }
```

i = 1



نحوه اجرای دستور for

```
int i;

for  (i=1;   i<=3;    i++)
     {
      cout << “hello \n”;
     }
```

i = 2

```
hello
```

نحوه اجرای دستور for

```
int i;

for  (i=1;   i<=3;    i++)
     {
     cout << “hello \n”;
    }
```

i = 3

```
hello
hello
```

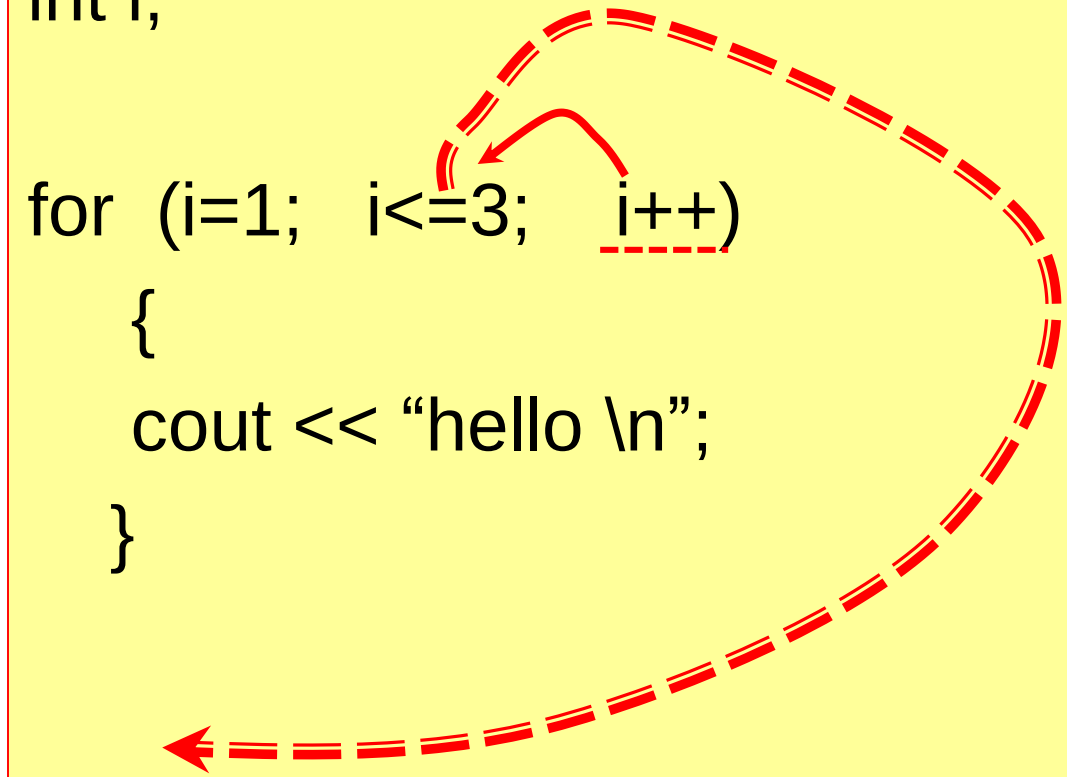
نحوه اجرای دستور for

```
int i;

for  (i=1;   i<=3;    i++)
     {
      cout << “hello \n”;
     }
```

```
hello
hello
hello
```

نحوه اجرای دستور for

```
int i;

for  (i=1;   i<=3;    i++)
     {
     cout << “hello \n”;
    }
```



i = 4

```
hello
hello
hello
```

نکته

- لزومی ندارد که کنترل گر حلقه حتماً از ۱ شروع شود.

```
int i;

for  (i=۵;   i<=۷;    i++)
     {
     cout << “hello \n”;
    }
```

نکته

- مقدار دهی اولیه کنترل گر حلقه می تواند خارج از دستور for باشد.

```
int i =1;

for  ( i  ;   i<=3;    i++)
     {
     cout << “hello \n”;
    }
```

نکته

- مقدار دهی اولیه کنترل گر حلقه می تواند خارج از دستور for باشد. در این صورت می توان جمله اول موجود در عبارت for را خالی گذاشت.

```
int i=1;

for  (  i;   i<=3;    i++)
     {
     cout << “hello \n”;
    }
```

=

```
int i=1;

for  (    ;   i<=3;    i++)
     {
     cout << “hello \n”;
    }
```

نکته

- گام حرکت می تواند در بدنه دستور for تعریف شود.

```
int i=1;

for  (  i;   i<=3;      )
     {
     cout << “hello \n”;
     i++;
    }
```

نکته

- معرفی کنترل گر حلقه می تواند در داخل دستور for باشد.

```
for  (int i= 1;   i<=3;    i++)
     {
     cout << “hello \n”;
    }
```

نکته

- در دستور for اگر قسمت شرط خالی باشد، حلقه همیشه اجرا خواهد شد. به عبارتی هیچ شرطی برای توقف نداریم

```
for  (int i= 1;          ;    i++)
    {
    cout << “hello \n”;
   }
```

- وقتی هیچ شرطی نداریم، دو قسمت دیگر دستور for نیز می توانند خالی باشند.

```
for  (          ;          ;          )
    {
    cout << “hello \n”;
   }
```

نکته

- لزومی ندارد که گام حرکت بصورت افزایشی باشد بلکه می تواند بصورت کاهشی نیز باشد.

```
for  (int i= ۳;   i >= 1;    i--)
    {
    cout << “hello \n”;
   }
```

- تمرین: اعداد ۱۰۰ تا ۱ را به صورت نزولی چاپ نمایید. (با گام حرکت افزایشی و کاهشی جداگانه بنویسید)

نکته

- گام حرکت می تواند افزایش یا کاهش بیش از ۱ واحد را داشته باشد.
- مثال: چاپ اعداد فرد بین ۱ تا ۱۰۰

```
for  (int i= ۱;   i <= 100;    k=k+2)
    {
    cout << k << “\n”;
   }
```

نکته

- کنترل گر حلقه می تواند اعشاری یا کاراکتری باشد، لزومی ندارد که حتما عدد صحیح در نظر بگیریم.

```
for  (char ch= ‘a’;   ch <= ‘z’;    ch++)
     {
     cout << ch << “\n”;
    }
```

برنامه زیر عدد صحیح و مثبت n را از ورودی گرفته فاکتوریل آن را محاسبه ونمایش می‌دهد.

```
#include          <iostream.h>
int      main( )
{
int  n, i ;
long fact = 1 ;
cout << "Enter a positive integer number";
cin >> n;
for( i=1; i<=n; ++i)
     fact *= i;
 cout << fact << endl;
return 0 ;
}
```

برنامه زیر مجموع اعداد صحیح و متوالی بین ۱ تا n را محاسبه نموده و نمایش می‌دهد.

```
#include        <iostream.h>
int      main( )
{
int n, i=1 ;
long s = 0 ;
cin >> n ;
for(    ; i<=n; i++)
      s += i;
cout << s ;
return  0 ; }
```

برنامه زیر ارقام 0 تا 9 را نمایش می‌دهد.

```
#include    <iostream.h>
int main( )
{
int  j=0 ;
for( ; j <= 9 ; )
      cout << j++ << endl;
return 0 ;
}
```

برنامه زیر کلیه اعداد سه رقمی که با ارقام 1 ، 2 ، 3 ایجاد می‌شوند را نمایش می‌دهد.

```
#include    <iostream.h>
int main( )
{
int  i,j,k,n;
for(i=1; i<=3; ++i)
     for(j=1; j<=3; ++j)
               for(k=1; k<=3; ++k)
               {
               n=i*100 + j*10+k;
               cout << n << '\n' ;
               }
return 0 ;
}
```

تمرین

- برنامه ای که اعداد ذوج بین ۱ تا ۱۰۰۰ را چاپ کند.
- برنامه ای که ۱۰۰ عدد را خوانده، مجموع را محاسبه و چاپ کند.
- برنامه ای که حاصلضرب اعداد ۱ تا ۵۰ را چاپ کند.
- برنامه ای که ۵۰ کاراکتر از صفحه کلید خوانده و تعیین کند که کدام یک حرف کوچک است.
- برنامه ای که ۱۰۰ عدد صحیح را خوانده max و min را چاپ کند.
- برنامه ای که کاراکتر هایی که کد آن ها بین ۱۰۰ تا ۲۰۰ هست، را چاپ نماید.

کاربرد دستور break در دستور for

- اگر در بدنه for از جمله ی break; استفاده شود، ادامه ی اجرای حلقه متوقف شده و حلقه خاتمه می یابد.

```
int i , x;
for(i=1;i<=100;i++)
{cin>>x;
if(x==50) break;
}
```

- قطعه کد فوق حداکثر ۱۰۰ عدد صحیح از ورودی می گیرد، ولی اگر در بین اعداد ورودی عدد ۵۰ وارد شود بدون بررسی شرط حلقه از ادامه اجرای دستورات for اجتناب کرده و از حلقه خارج می شود.

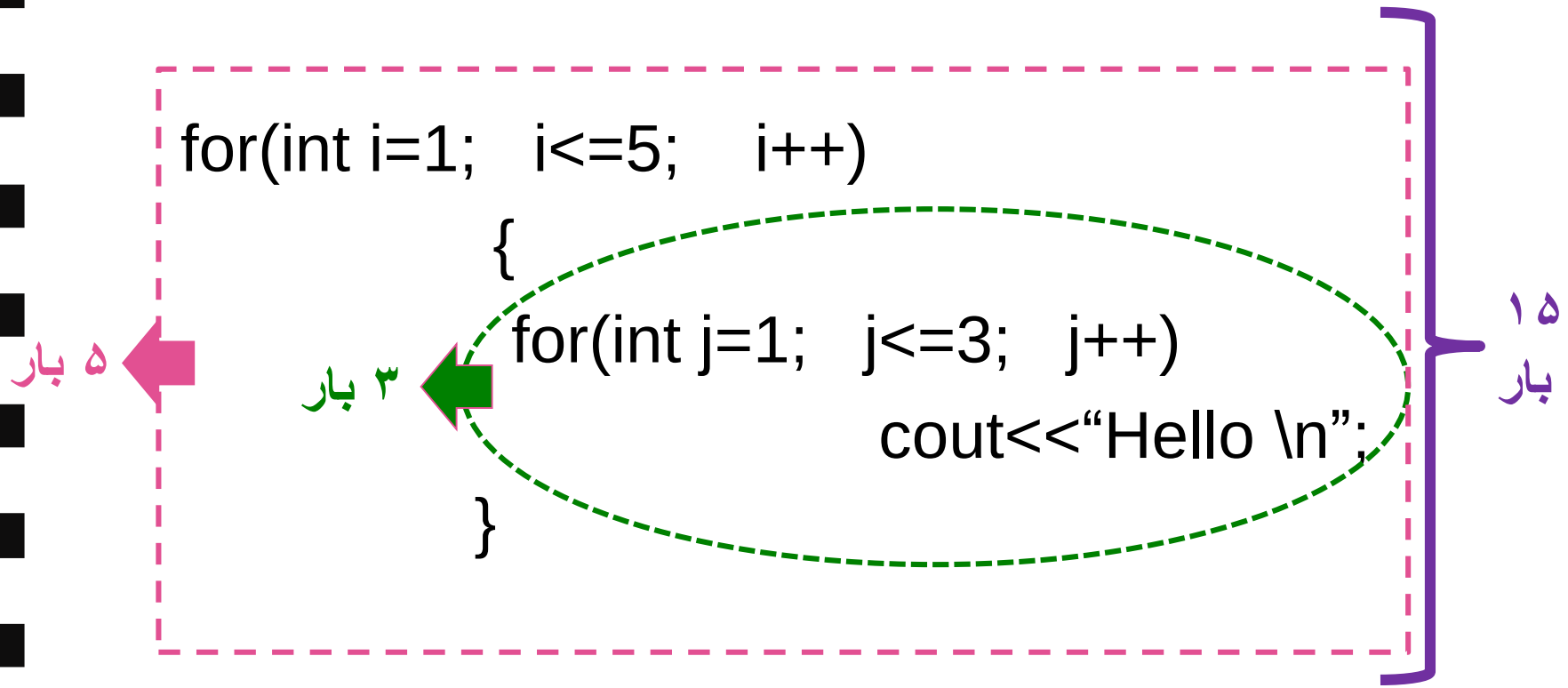
مثال

- قطعه کدی که تعدادی کاراکتر از صفحه کلید خوانده، بعد از فشردن دکمه ی E تعداد آن ها را مشخص کند:

```
char ch;
int i;
for(i=0;     ;i++)
            {cin>>ch;
            if (ch==‘E’) break;
            }
cout<<i;
```

حلقه for تودرتو

- می توان داخل بدنه ی دستور for هر دستور دلخواه دیگری نوشت. به عنوان مثال می توان از یک دستور for در بدنه دستور for استفاده کرد.
- قطعه کد زیر عبارت Hello را ۱۵ بار اجرا می کند:



مثال: چاپ جدول ضرب اعداد

```
int main()
{int i,j;
for(int i=1;   i<=10;   i ++)
                    for(int j=1;  j<=10;  j++)
                                { cout<<i*j<<“     “;
                                  if (j==10)   cout<<‘\n’;
                                  }
return 0;
}
```

مثال: برنامه ای که ۲۰ عدد را خوانده و برای هرکدام مجموع اعداد ۱ تا آن عدد را محاسبه کند.

```
int main()
{
int i,j,x,sum;
for(int i=1;   i<=20;  i++)
        {sum=0;   cin>>x;
        for(int j=1;  j<=x;   j++)
                        sum+=j;
         cout<<“sum of 1 to “<<x<<“ :“<<sum<<endl;
        }
return 0;
}
```

کاربرد حلقه for با دو اندیس

برنامه ای بنویسید که ستون اعداد زیر را چاپ کند:

روش اول:

```
for(int i=1;  i<=20;   i++)
    cout<<i<<“, “<<21-i<<endl;
```

روش دوم:

```
for(int i=1, j=20;    i<=20;    i++, j--)
    cout<<i<<“, “<<j<<endl;
```

1, 20
2, 19
3, 18
4, 17
.
.
.
20, 1

دستور continue; در for

- اگر دستور continue; در حلقه for استفاده شود، جملاتی از حلقه که هنوز اجرا نشده اند، بدون اجرا مانده و ادامه اجرا از انتهای حلقه اغاز خواهد شد.

```
int main()
{
int x, p=1;
for(int i=1;    i<=20;     i++)
            {cin>>x;
             if (!x) continue;
              p*=x;
            }
cout<<p;
return 0;
}
```

برنامه فوق ۲۰ عدد از ورودی خوانده و حاصلضرب اعداد غیر صفر را در متغیر p محاسبه کرده و در نهایت نمایش می دهد.

فصل سوم

سایر ساختارهای تکرار

فهرست مطالب فصل سوم

۱. دستورالعمل while
۲. دستورالعمل do while
۳. دستورالعمل break
۴. دستورالعمل continue
۵. دستورالعمل switch
۶. تابع cin.get()
۷. عملگر static_cast<>()
۸. جدول اولویت عملگرها

دستورالعمل while

از این دستور العمل مانند دستورالعمل for برای تکرار یک دستورالعمل ساده یا ترکیبی استفاده می‌گردد. شکل کلی این دستور العمل بصورت زیر می‌باشد.

```
معرفی کنترل گر حلقه;
مقدار دهی اولیه کنترل گر حلقه;

while( شرط)
        {
        دستورالعمل ۱ ;
        دستورالعمل ۲ ;
        .
        .
        دستورالعمل n ;
        گام حرکت;
        }
```

تفاوت دستورهای while و for

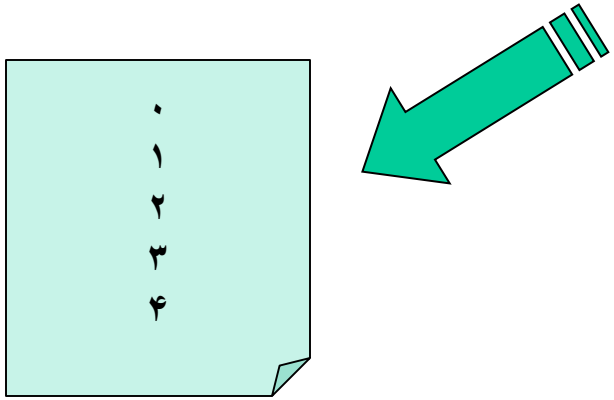
دستورالعمل **for** زمانی استفاده می شود که تعداد دفعات تکرار از قبل مشخص و معین باشد. در صورتیکه تعداد دفعات تکرار مشخص نباشد بایستی از دستورالعمل **while** استفاده نمود.

دقت کنید با پیاده سازی مکانیزم های خاصی می توان بدون توجه به دفعات تکرار از هر دو دستور بصورت مشابه استفاده کرد.

مثال :

```
int x=0
while(x<5)
cout << x ++<< endl;
```

با اجرای قطعه برنامه فوق مقادیر زیر نمایش داده میشود :



برنامهٔ فوق n مقدار از نوع اعشاری را گرفته میانگین آن ها را محاسبه و در متغیر avg قرار می‌دهد.

```
#inclode   <iostream.h>
int main( )
{
int count = 0 , n;
float  x, sum = 0 , avg ;
cin >> n ; /*  تعداد مقادیر ورودی n*/
while(count < n){
                        cin >> x ;
                        sum += x ;
                       ++ count ; }
avg = sum / n ;
cout << avg << endl;
return 0 ; }
```

نکته

- همانند دستور for اگر داخل بدنه دستور while فقط یک جمله باشد می توان } و { را حذف کرد.

- مثال: نمایش اعداد ۰ تا ۱۰۰:

```
int main( )
{
int j = 0;
while(j<=100)
                cout<<j++<<endl;

return 0 ;
}
```

برنامه زیر را تفسیر کنید:

```
int main( )
{
int j = 0;
while(j<10)
        J++;
        cout<<j;
return 0 ;
}
```

دستورالعمل do while

این دستور العمل نیز برای تکرار یک دستورالعمل ساده یا ترکیبی استفاده می‌شود. شکل کلی این دستورالعمل بصورت زیر می‌باشد.

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

```
معرفی کنترل گر حلقه;
مقدار دهی اولیه کنترل گر حلقه;

do
     { دستورالعمل ۱ ;
       دستورالعمل ۲ ;
        .
        .
       دستورالعمل n ;
       گام حرکت;
     } while( شرط);
```

تفاوت دستورهای while وdo while

در دستورالعمل while ابتدا مقدار شرط ارزیابی شده اما در دستورالعمل do while ابتدا دستورالعمل اجرا شده سپس مقدار شرط ارزیابی می‌گردد. بنابراین دستورالعمل do while حداقل یک بار انجام می شود .

مثال :

```
#include   <iostream.h>
int main( )
{
int count = 0;
do
cout << count ++<<endl ;
while(count <= 9);
return 0 ; }
```

ارقام 0 تا 9 را روی ده خط نمایش می‌دهد

دستورالعمل break

این دستورالعمل باعث توقف دستورالعمل های تکرار(for , while ,do while) شده و کنترل به خارج از این دستورالعمل ها منتقل می‌نماید.

مثال ۱ :

```
#include   <iostream.h>

int main( )
{
float x, s=0.0 ;
cin >> x ;
while(x <= 1000.0) {
                        if(x < 0.0){
                                cout << "Error-Negative Value" ;
                                 break;
                                 }
                        s += x ;
                        cin >> x ;}
cout << s << endl ;
return 0 ;
}
```

جمع تعدادی عدد که بین ۰ و 1000 هستند. اگر بین اعداد وارد شده عدد منفی وارد شود، بواسطه دستور break; حلقه خاتمه می یابد.
اگر عدد وارد شده بزرگتر از 1000 باشد نیز شرط حلقه برآورده نشده و حلقه خاتمه می یابد

مثال ٢:

```
#include   <iostream.h>
int main( )
{
int count = 0 ;
while( 1 )
            {
             count ++ ;
             if(count > 10 )
                          break ;
            }
cout << "counter :   " << count << "\n";
return 0 ;
}
```

```
counter : 11
```

مثال ۳:

```
#include   <iostream.h>
void  main( )
{
int count;
float x, sum = 0;
cin >> x ;
for(count = 1; x < 1000 . 0; ++ count )
                    {
                    cin >> x ;
                    if(x < 0.0) {
                                cout << "Error – Negative value " <<endl;
                                break ;
                                }
                    sum += x ;
                    cin >> x ;
                    }
cout << sum << '\n' ;
 }
```

جمع تعدادی عدد اعشاری که بیشتر از ۰ و کمتر از ۱۰۰۰ هستند را محاسبه می کند. اگر بین اعداد وارد شده عدد منفی وارد شود، بواسطه دستور break; حلقه خاتمه می یابد.
اگر عدد وارد شده بزرگتر یا مساوی 1000 باشد نیز شرط حلقه برآورده نشده و حلقه خاتمه می یابد

مثال ٤:

```
#include   <iostream.h>
int main( )
{
float x , sum = 0.0 ;

do  {
      cin >> x ;
      if(x < 0.0)
                        {
                        cout << "Error – Negative Value" << endl ;
                        break ;
                        }
      sum += x ;
       } while(x <= 1000.0);

cout << sum << endl ;
return 0 ;
 }
```

جمع تعدادی عدد که بین ۰ و 1000 هستند. اگر بین اعداد وارد شده عدد منفی وارد شود، بواسطه دستور break; حلقه خاتمه می یابد.

اگر عدد وارد شده بزرگتر از 1000 باشد نیز شرط حلقه برآورده نشده و حلقه خاتمه می یابد.

نکته: این حلقه حداقل یک بار اجرا می شود.... حتی اگر عدد اول بزرگتر از ۱۰۰۰ باشد نیز در مجموع شرکت داده خواهد شد. سپس از حلقه خارج خواهیم شد.

دستورالعمل continue

از دستورالعمل **continue** می‌توان در دستورالعمل هـای تکـرار **do while** ، **while** ، **for** استفاده نمود. این دستورالعمل باعث می‌شود که کنترل به ابتـدای دستورالعمل های تکرار منتقل گردد.

(تذکر: در ابتدای حلقه بایستی شرط حلقه نیز بررسی شود)

پس بهتر است بگوییم به انتهای بدنه حلقه تکرار رفته و از آن جا ادامه می دهد.

مثال ۱:

```
#include   <iostream.h>
int main( )
{
float  x, sum = 0.0 ;
Do {
      cin >> x ;
       if(x < 0 . 0)
                      {
                      cout  << "Error" << endl ;
                      continue ;
                       }
       sum += x ;
      } while(x <= 1000.0 );
cout << sum ;
return 0 ; }
```

جمع تعدادی عدد که کوچکتر و مساوی 1000 هستند را بدست می اورد. اگر بین اعداد وارد شده عدد منفی وارد شود، در حاصل جمع شرکت نخواهد کرد و اجرای دستورالعمل ها به انتهای حلقه هدایت خواهد شد.

اگر عدد وارد شده بزرگتر از 1000 باشد نیز شرط حلقه برآورده نشده و حلقه خاتمه می یابد.

نکته: این حلقه حداقل یک بار اجرا می شود.... حتی اگر عدد اول بزرگتر از ۱۰۰۰ باشد نیز در مجموع شرکت داده خواهد شد سپس از حلقه خارج خواهیم شد.

مثال ۲:

n عدد از ورودی خوانده. در صورتیکه صفر نباشد ان را
در مجموع اعداد شرکت می دهد.
در نهایت میانگین اعداد غیر صفر را محاسبه می کند

```
#include   <iostream.h>
int main( )
{
int n , navg = 0 ;
float  x, avg, sum = 0 ;
cin >> n ; / * عبارت از تعداد اعداد ورودی n * /
for(int count = 1 ; count <=n; ++ count )
                          {
                           cin >> x ;
                           if(x == 0 ) continue ;
                            sum += x ;
                            ++ navg ;
                           }
avg = sum / navg;
cout << avg << endl ;
return 0 ;
 }
```

دستورالعمل switch

همانطور که می دانید از دستورالعمل شرطی(if else) می توان بصورت تودرتو استفاده نمود ولی از طرفی اگر عمق استفادهٔ تو در تو از این دستورالعمل زیاد گردد، درک آن ها مشکل می شود . برای حل این مشکل C++ ، دستورالعمل switch که عملاً یک دستورالعمل چند انتخابی می‌باشد را ارائه نموده است.

شکل کلی دستور العمل Switch

```
switch(عبارت)
{
case     valueone : statement;
                          break;
case      valuetwo: statement;
                         break;



case       valuen : statement;
                         break;
default: statement ;
}
```

مثال ١ :

```
#include   <iostream.h>
void main( )
{
unsigned int n ;
cin >> n;
switch(n)
{
     case   0:
               cout << "ZERO" << endl ;
               break;
     case   1:
               cout << "one" << endl ;
               break ;
     case 2:
               cout << "two" << endl ;
               break;
     default :
               cout << "default" << endl;
}        / * end of switch statement * /
}
```

مثال ٢ :

```
#include   <iostream.h>
void main( )
{
unsigned  int  n;
cin >> n ;
switch(n) {
                case 0:
                case 1:
                case 2:
                        cout << "Less Than Three" << endl;
                        break;
                case 3:
                        cout << "Equal To Three" << endl ;
                        break;
                default:
                        cout << "Greater Than Three" << endl;
            }
}
```

تابع cin.get() :

این تابع یک کرکتر را از صفحه کلید می‌گیرد. برای استفاده از این تابع در ابتدای برنامه بایستی داشته باشیم:
#include <iostream.h>

قطعه برنامه ذیل یک کرکتر را از صفحه کلید گرفته و نمایش می‌دهد.

```
char        x;
x = cin.get( );
cout << x ;
```

برنامهٔ ذیل یک سطر متن انگلیسی که به CTRL Z ختم می شود را گرفته دقیقاً نمایش می‌دهد.

```
#include      <iostream.h>
int main( )
{
char  x;
while((x = cin.get( )) !=EOF)
cout << x ;
return 0 ;
 }
```

EOF به معنی End of File می‌باشد که در iostream.h تعریف شده و مقدار آن برابر با ۱- می‌باشد. مقدار آن در سیستم عامل DOS عبارتست از ctrl z .

در قطعه برنامه ذیل از تابع cin.get() و دستور switch استفاده شده است.

```
char        x;
x = cin.get( );
switch(x)  {
case  ′ r ′ :
case  ′ R ′ :
            cout << ″RED″ << ″\n″ ;
            break ;
case  ′ b ′ :
case  ′ B ′ :
             cout << ″BLUE″ << endl ;
             break ;
case  ′ y ′ :
case  ′ Y ′ :
            cout <<  ″YELLOW″ << endl; break;
  }
```

برنامهٔ ذیل یک سطر متن انگلیسی را گرفته کرکترهای خالی (blank) آن را حذف نموده و نمایش می دهد.

```
#include      <iostream.h>
int main( )
{
char next;
while((next = cin.get( ) ) !=EOF)
if(next != ' ' )
cout  << next ;
return 0 ;
 }
```

تمرین : برنامه ای با استفاده از ساختار switch case که یک عملگر و دو عملوند را از ورودی خوانده ، عملگر را بر روی عملوند اجرا نمایید.

عملگر static_cast

از این عملگر برای تبدیل موقت یک نوع **data** به نوع دیگر استفاده می‌شود. این عملگر یک عملگر یکتائی می‌باشد.

مثال ۱:

```
int   x = 25 ;
float  y ;
y = static_cast < float >(x) ;
```

مقدار x موقتاً بصورت اعشاری در می آید و در نتیجه مقدار y برابر با 25.0 می شود. بایستی توجه داشت که نوع متغیر x عوض نمی شود بلکه موقتاً مقدار آن بصورت اعشاری در آمده است.

مثال ۲:

```
float x = 14.75 ;
cout << static_cast < int >(x) << endl;
cout << x ;
```

ابتدا مقدار ۱۴ نمایش داده می شود و سپس مقدار ۱۴,۷۵ نمایش داده میشود.

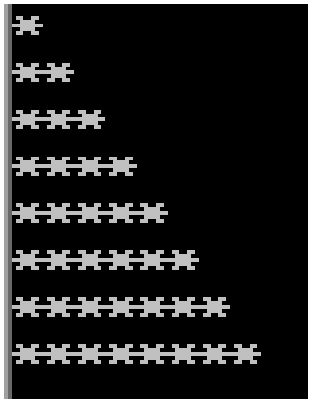
جدول اولویت عملگرها

چپ به راست	( )
راست به چپ	Static_cast < >( ) ++ -- + - sizeof
چپ به راست	* / %
چپ به راست	+ -
چپ به راست	<< >>
چپ به راست	< <= > >=
چپ به راست	= = !=
راست به چپ	? :
راست به چپ	= += -= *= /= %=
چپ به راست	,

تمرین

۱. برنامه ای که یک سکه ۱۰۰ ریالی را به سکه های ۲و۵ و۱۰و۲۰ و ۵۰ ریالی خرد کند.

۲. برنامه ای که خروجی زیر را چاپ کند:



۳. برنامه ای که حاصل عبارت زیر را بدست اورد:

$$S = 1 + \frac{2}{2!} + \frac{3}{3!} + \ldots + \frac{N}{N!}$$

فصل چهارم

اعداد تصادفی

فهرست مطالب فصل چهارم

۱. تولید اعداد تصادفی
۲. تعریف نوع داده (typedef)
۳. داده های از نوع شمارشی

اعداد تصادفی

مقادیر تصادفی یا شانسی در اکثر برنامه‌های کاربردی در زمینه شبیه سازی و بازیهای کامپیوتری نقش مهمی را ایفا می‌نمایند. برای ایجاد یک عدد تصادفی صحیح بین ۰ و ۳۲۷۶۷ بایستی از تابع **rand()** استفاده نمائیم.

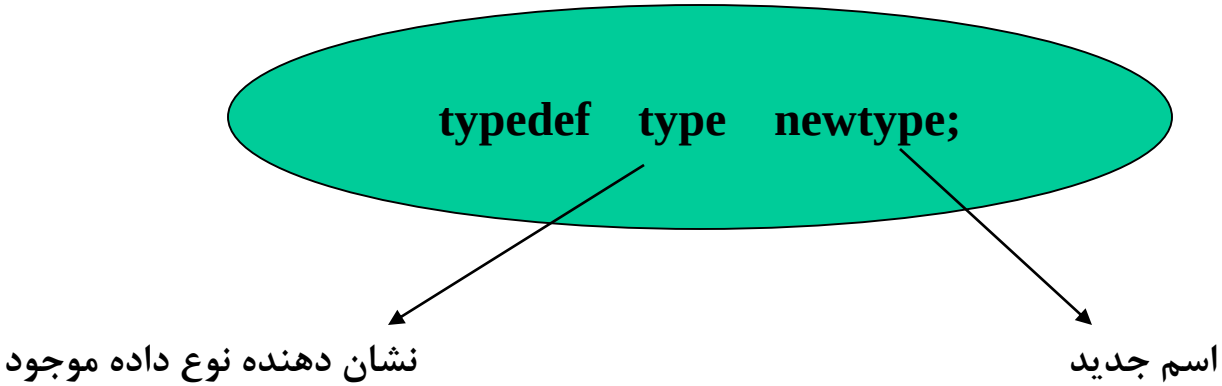
rand( )

برنامه زیر 10 عدد تصادفی بین 0 و 32767 را ایجاد می‌نماید.

```
#include        <stdlib.h>
#include        < iostream.h>
int main( )
{
for(int j=1; j<=10; ++j)
cout << rand( ) << '\n' ;
return 0 ;
}
```

تعریف نوع داده (typedef)

از **typedef** می‌توان برای تعریف نوع داده‌های جدید که معادل نوع داده‌های موجود باشد استفاده نمود. شکل کلی عبارتست از :



مثال :

```
typedef  int  integer;
```

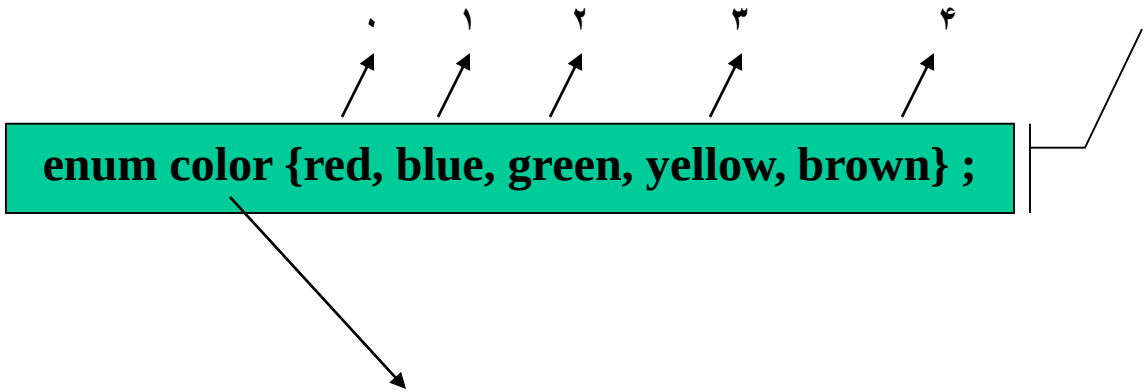
حال می‌توان y و x را بصورت زیر تعریف نمود :

```
integer  x,y;
```

داده‌های از نوع شمارشی

بمنظور معرفی داده‌های از نوع شمارشی از کلمه **enum** استفاده می‌گردد.

مثال



مثال

```
emum status {married, devorced, vidow, single};
  status  a ;
   a= single ;
```

:

توجه :

بایستی در نظر داشت که داده‌های از نوع شمارشی در عملیات ورودی و خروجی شرکت نمی‌نمایند. بعبارت دیگر مقادیر داده‌های از نوع شمارشی بایستی در برنامه تعیین نمود. دستورالعمل های ورودی و خروجی مانند **cin** و **cout** در مورد داده‌های شمارشی نمی‌توان استفاده نمود.

فصل پنجم

آرایه ها

فهرست مطالب فصل پنجم

۱. آرایه یک بعدی
۲. آرایه دو بعدی (ماتریس ها)

آرایه یک بعدی

آرایه یک فضای پیوسته از حافظه اصلی کامپیوتر می‌باشد که می‌تواند چندین مقدا را در خود جای دهد.

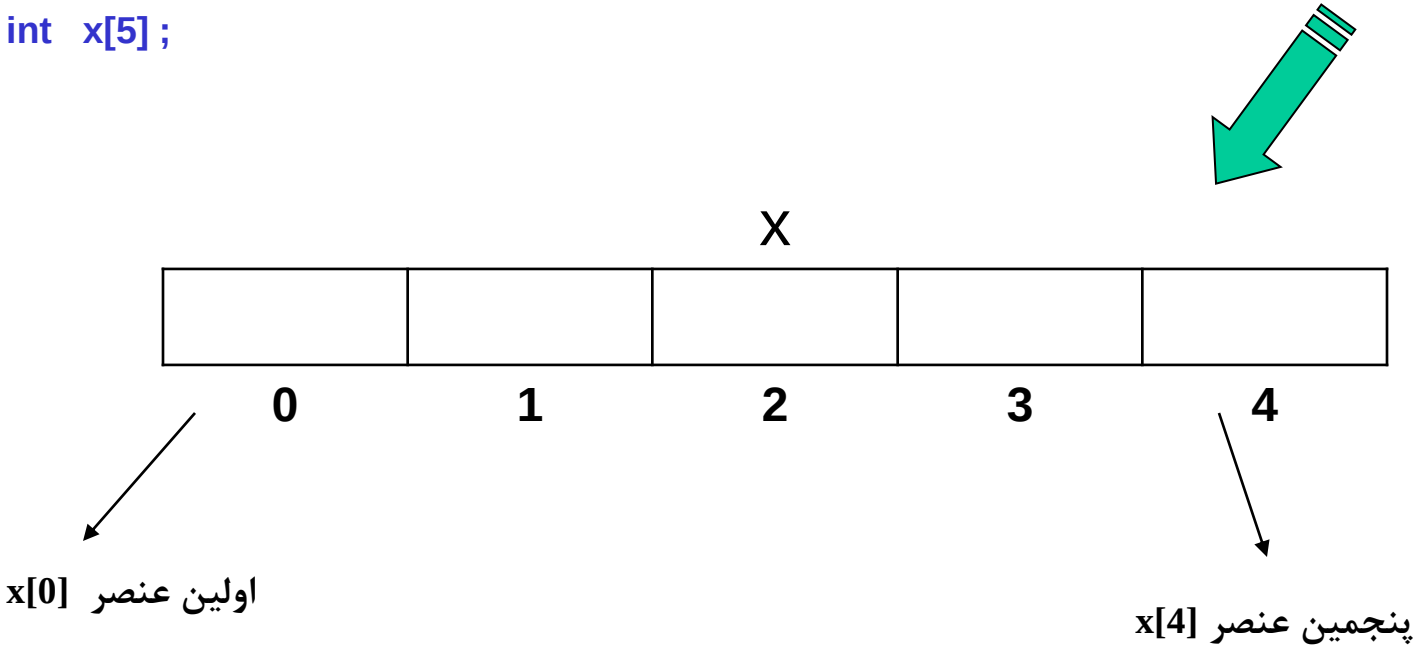
- کلیه عناصر یک آرایه از یک نوع می‌باشند.
- عناصر آرایه بوسیله اندیس آن ها مشخص می‌شوند.
- در C++ ، اندیس آرایه از صفر شروع می‌شود.

کاربرد آرایه ها

آرایه‌ها در برنامه‌نویسی در مواردی کاربرد دارند که بخواهیم اطلاعات و داده‌ها را در طول اجرای برنامه حفظ نمائیم.

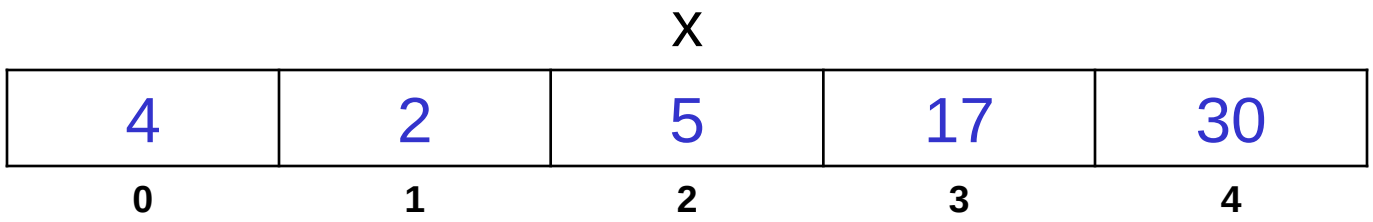
آرایه یک بعدی از نوع int

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



تخصیص مقادیر اولیه به عناصر آرایه :

```
int  x[5]= {4, 2, 5, 17, 30};
```



دریافت مقادیر عناصر آرایه :

```
int  x[5];
for(int    i=0; i<=4; ++i)
              cin >> x[ i ] ;
```

نمایش مقادیر عناصر آرایه :

```
 for(int   i=0; i<5; ++i)
              cout << x[ i ] ;
```

اگر تعداد مقادیر اولیه کمتر از تعداد عضوهای آرایه باشد عضوهای باقیمانده بطور اتوماتیک، مقدار اولیه صفر می‌گیرند.

```
int   x[5] = {12, 5, 7};
```

X

12	5	7	0	0
0	1	2	3	4

بایستی توجه داشت که آرایه‌ها به صورت ضمنی مقدار اولیه صفر نمی‌گیرند. برنامه نویس باید به عضو اول آرایه، مقدار اولیه صفر تخصیص دهد تا عضوهای باقی‌مانده بطور اتوماتیک، مقدار اولیه صفر بگیرند.

```
int   x[5] = {0} ;
```

X

0	0	0	0	0
0	1	2	3	4

دستور زیر یک آرایه یک بعدی شش عنصری از نوع float ایجاد می‌نماید.

```
float x[  ] = {2.4, 6.3, -17.1, 14.2, 5.9, 16.5} ;
```

X

2.4	6.3	-17.1	14.2	5.9	16.5
0	1	2	3	4	5

برنامه ذیل 100 عدد اعشاری و مثبت را گرفته تشکیل یک آرایه می دهد سپس مجموع عناصر آرایه را مشخص نموده نمایش می‌دهد.

```
#include  <iostream.h>
int main( )
{
const int arrsize = 100 ;
float x[ arrsize], tot = 0.0 ;
for(int   j=0; j<arrsize; j++)
                              cin >> x[ j ];
for(j=0; j<arrsize; j++)
                           tot += x[ j ] ;
cout << tot ;
return 0 ;
}
```

برنامه ذیل 20 عدد اعشاری را گرفته تشکیل یک آرایه داده سپس کوچکترین عنصر آرایه را مشخص و نمایش می‌دهد.

```
#include  <iostream.h>
#include   <conio.h>
int main( )
{
float   x[20], s;
int  j ;
for(j=0; j<20 ; ++j)
                    cin >> x[ j ];
s = x[0 ] ;
for(j=1; j<20; ++j)
               if(x[ j] <s)
                         s = x[ j ];
cout << s << endl;
return 0;
}
```

برنامه زیر 100 عدد اعشاری را گرفته بروش حبابی (Bubble sort) بصورت صعودی مرتب می‌نماید.

```
#include  <iostream.h>
#include  <conio.h>
int main ( )
{
float  x[100] , temp;
int   i,j ;
for(i=0; i<100; ++i)
     cin >> x[i ];
for(i=0; i<99; i++)
     for(j=i+1 ; j<100; j++)
                         if(x[ j ] < x[i ]
                         {
                                   temp = x[ j ] ;
                                   x[ j ] = x[ i ];
                                   x[ i ] = temp ;
                         }
for(i=0; i<=99; i++)
     cout << x[ i ] << endl;
return 0 ;
}
```

آرایه‌های دوبعدی (ماتریس‌ها)

ماتریس ها بوسیله آرایه‌های دوبعدی در کامپیوترنمایش داده می شوند.

```
int a[3][4];
```

	ستون 0	ستون 1	ستون 2	ستون 3
سطر 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
سطر 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
سطر 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

تخصیص مقادیر اولیه به عناصر آرایه :

```
int a[3][4]={ {1,2,3,4}, {5,6,7,8}, {9,10,11,12} } ;
```

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

نكتة ١ :

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

	0	1	2	3
0	1	0	0	0
1	2	3	0	0
2	4	5	6	0

نكتة ٢ :

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

	0	1	2	3
0	1	2	3	4
1	5	0	0	0
2	0	0	0	0

نکته ۳ :

در یک آرایهٔ دواندیسی، هر سطر، در حقیقت آرایه‌ای یک اندیسی است. در اعلان آرایه‌های دواندیسی ذکر تعداد ستون ها الزامی است.

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

	0	1	2	3
0	1	2	3	4
1	5	0	0	0

برنامه زیر یک ماتریس 3*4 را گرفته مجموع عناصر آن را مشخص نموده و نمایش می‌دهد.

```
#include  <iostream.h>
#include  <conio.h>
int main( )
{
float   x[3][4], total= 0.0;
int   i, j ;
// generate matrix x.
for(i=0; i<3; ++i)
    for (j=0; j<4; j++)
          cin >> x[ i ][ j ];
// calculate the sum of elements.
for(i=0; i<3; ++i)
        for(j=0; j<4; j++)
                total + = x [ i ][ j ];
cout << "total =  " << total << endl;
return 0 ;
}
```

برنامه که عناصر ماتریس ۲*۳ را از ورودی خوانده و بزرگترین عنصر هر سطر را پیدا کرده و به همراه شماره سطر آن چاپ نماید.

ضدحال...!

1. برنامه ای بنویسید که دو ماتریس را از ورودی خوانده و حاصلضرب آن را محاسبه و چاپ نماید.

- توجه کنید که دو ما زمانی قابل ضرب هستند که تعداد ستون های ماتریس اول با تعداد سطرهای ماتریس دوم برابر باشد. پس ابعاد ماتریس را می توانید تحت عنوان سه عدد از کاربر بپرسید. بصورت شکل زیر:

$$[\ddots]_{m\times n} \times [\ddots]_{n\times p} = [\ddots]_{m\times p}$$

- ابعاد ماتریس را از کاربر بپرسید. یعنی m و n و p

فصل ششم

توابع

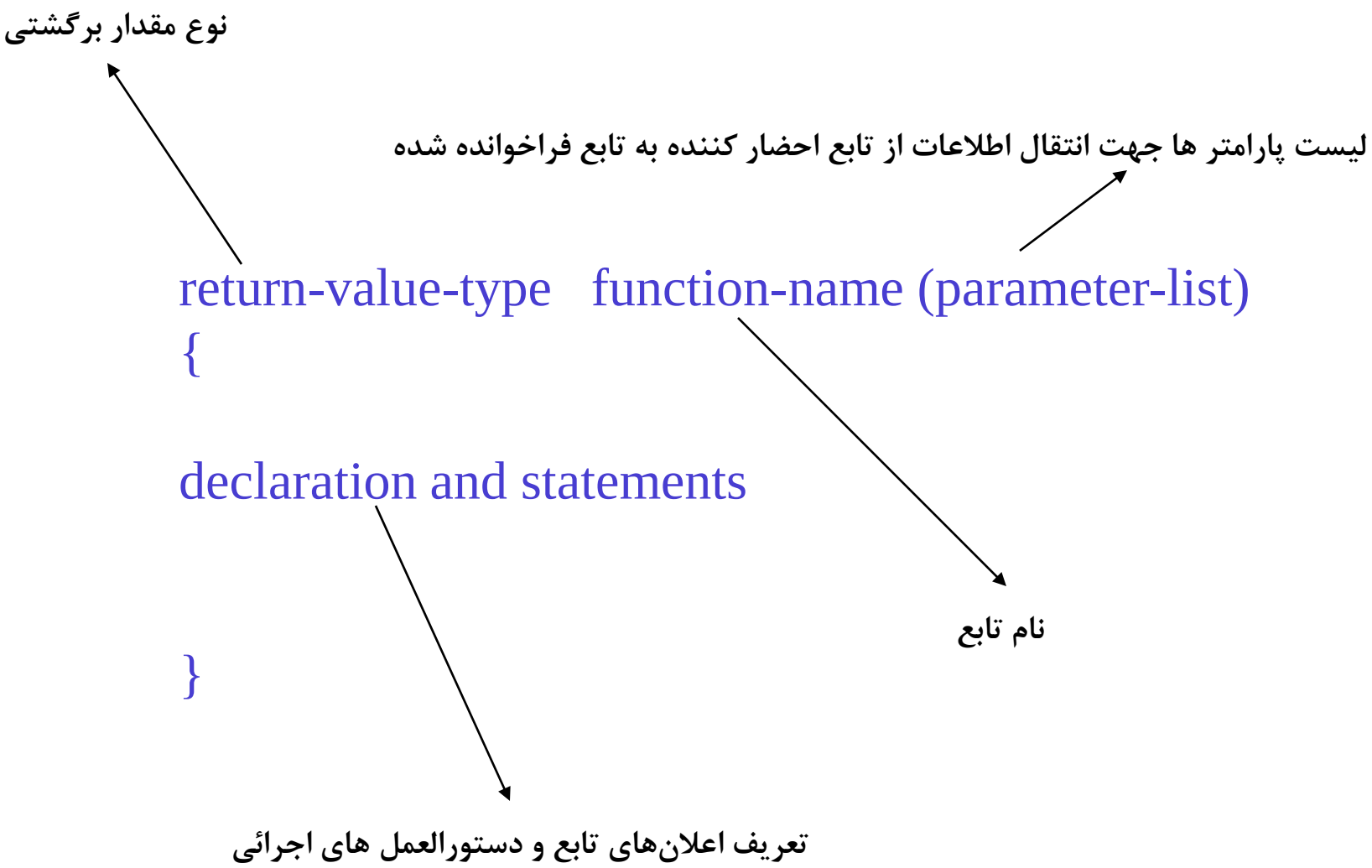
فهرست مطالب فصل ششم

۱. تعریف تابع
۲. تابع بازگشتی
۳. توابع درون خطی
۴. انتقال پارامترها از طریق ارجاع
۵. کلاس های حافظه (storage classes)
۶. سربارگذاری توابع

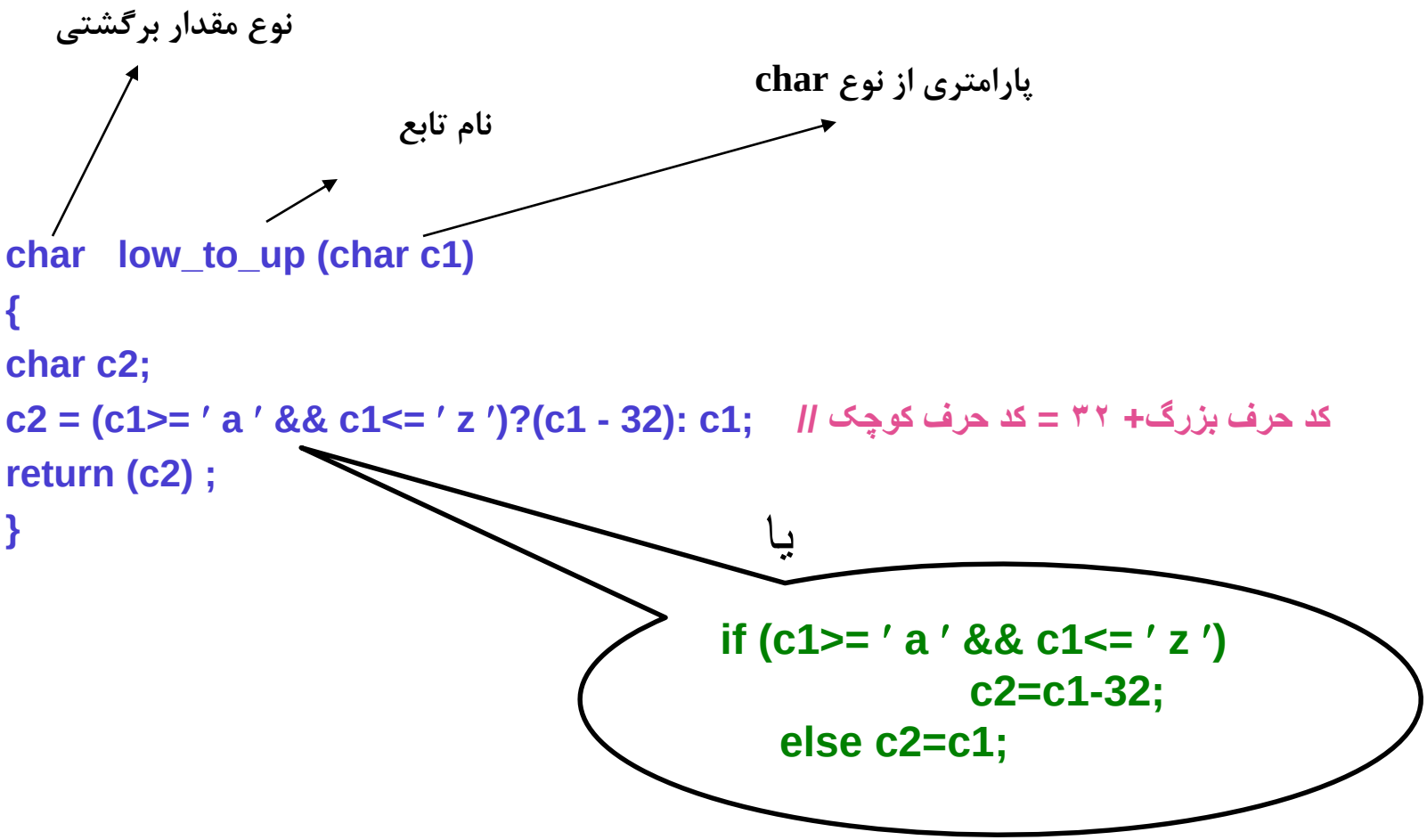
تعریف توابع

استفاده از توابع در برنامه‌ها به برنامه‌نویس این امکان را می‌دهد که بتواند برنامه‌های خود را به صورت قطعه قطعه برنامه بنویسد. تا کنون کلیه برنامه‌هائی که نوشته‌ایم فقط از تابع ( )main استفاده نموده‌ایم .

شکل کلی توابع بصورت زیر می‌باشند :



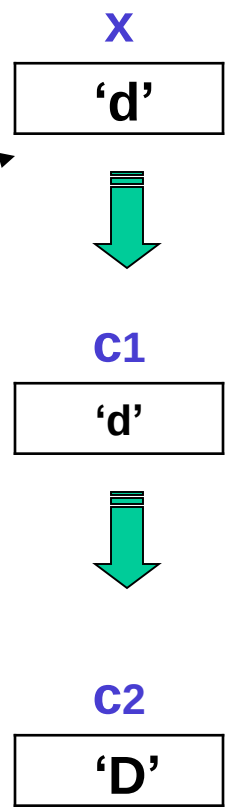
تابع زیر یک حرف کوچک را به بزرگ تبدیل می‌نماید.



برنامه کامل که از تابع قبل جهت تبدیل یک حرف کوچک به بزرگ استفاده می‌نماید.

```
#include  <iostream.h>
char low_to_up(char c1)
{
char  c2;
c2=(c1 >= ′ a ′ && c1 <= ′ z ′)?(′ A ′ +c1 -′ a ′) : c1;
return c2;
}
int main( )
{
char x;
x=cin.get( );
cout << low_to_up(x) ;
return 0;
 }
```

آرگومان



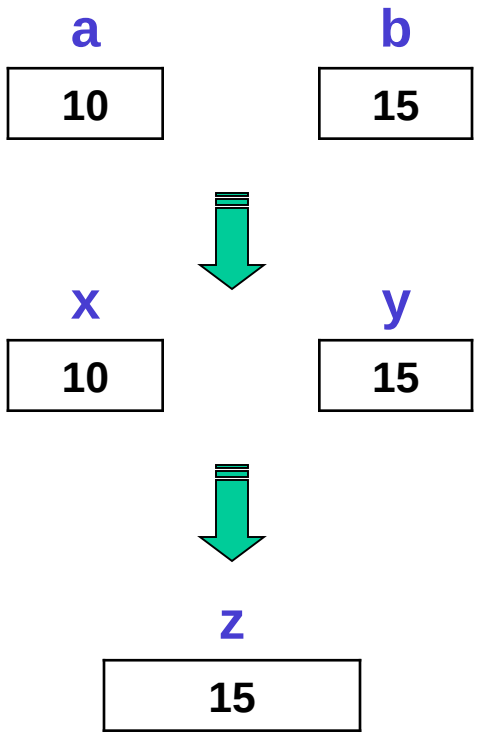
تابع maximum دو مقدار صحیح را گرفته بزرگترین آنها را برمی گرداند.

```
int   maximum(int x, int y)
{
int z ;
z=(x >= y)? x : y;      // if (x>=y) z=x; else z=y;
return z;
}
```

برنامه کامل که از تابع maximum جهت یافتن ماکزیمم دو مقدار صحیح استفاده می نماید.

```
#include  <iostream.h>
int   maximum(int x , int y)
{
int z ;
z=(x > y)? x : y ;
return z;
}

int main( )
{
int a, b ;
cin >> a >> b ;
cout << maximum(a,b);
return 0;
}
```

آرگومانهای تابع maximum a, b



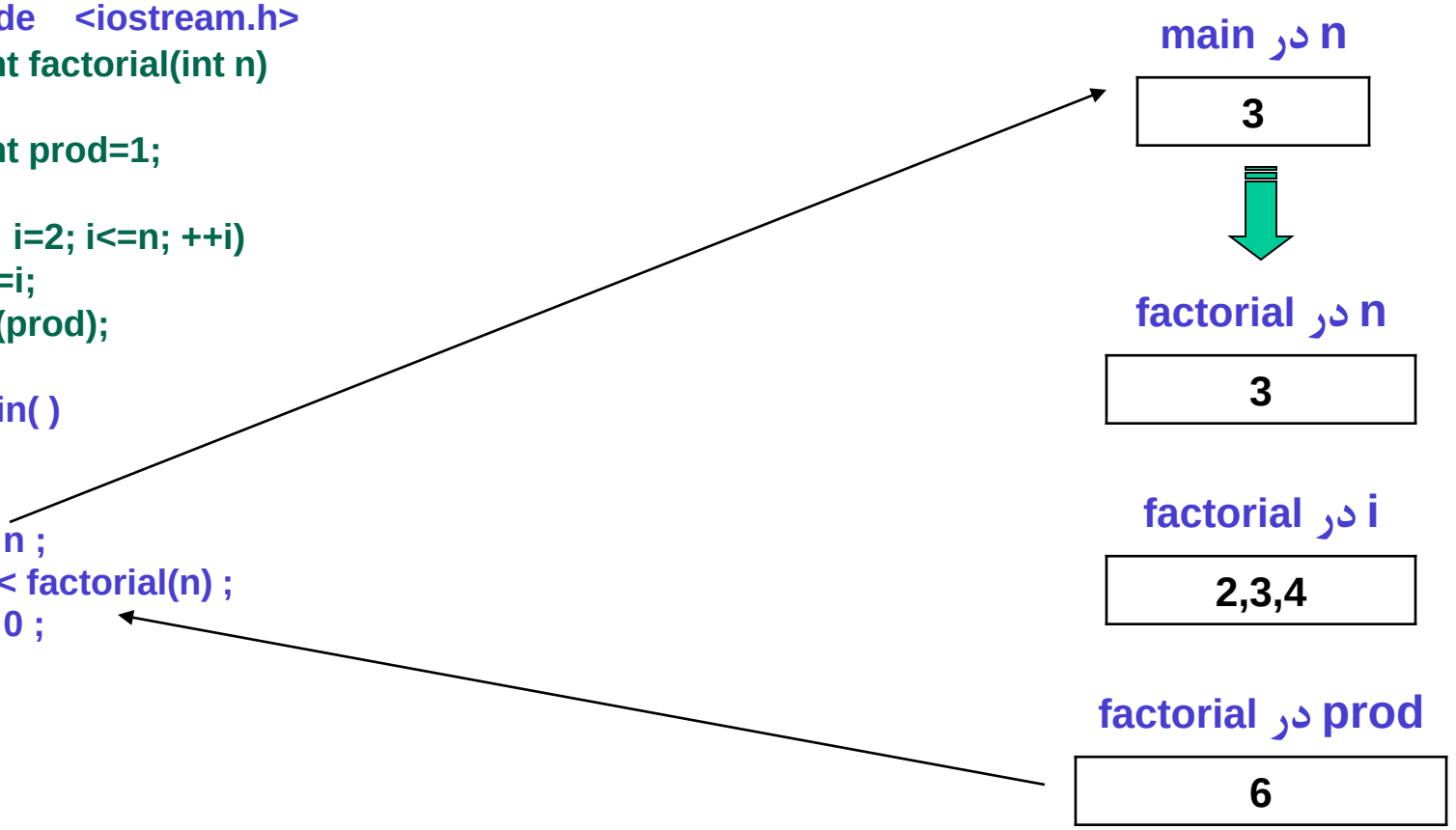
نکته ۱ :

اسامی پارامترها و آرگومان های یک تابع
می‌توانند همنام باشند.

برنامه زیر یک مقدار مثبت را گرفته فاکتوریل آنرا محاسبه نموده نمایش می‌دهد.

$$x!=1*2*3*4*...*(x-1)*x$$

```
#include    <iostream.h>
long int factorial(int n)
{
long int prod=1;
if(n>1)
for(int  i=2; i<=n; ++i)
prod *=i;
return(prod);
}
int main( )
{
int n;
cin >> n ;
cout << factorial(n) ;
return 0 ;
}
```



نکته ۲ :

وقتی در تابعی، تابع دیگر احضار می‌گردد
بایستی تعریف تابع احضار شونده قبل از
تعریف تابع احضار کننده در برنامه ظاهر گردد.

نکته ۳ :

اگر بخواهیم در برنامه‌ها ابتدا تابع **main** ظاهر گردد بایستی **prototype** تابع یعنی پیش نمونهٔ تابع که شامل نام تابع، نوع مقدار برگشتی تابع، تعداد پارامترهائی را که تابع انتظار دریافت آنرا دارد و انواع پارامترها و ترتیب قرارگرفتن این پارامترها را به اطلاع کامپایلر برساند.

در اسلاید بعد مثالی در این زمینه آورده شده است.

```
#include    <iostream.h>
#include   <conio.h>
long int factorial(int); // function prototype
int main( )
{
int n;
cout << "Enter a positive integer" << endl;
cin >> n;
cout << factorial(n) << endl;
return 0 ;
}
long int factorial(int n)
{
long int prod = 1;
if(n>1)
for(int i=2; i<=n; ++i)
prod *= i;
return(prod);
}
```

نکته ۲ :

در صورتی که تابع مقداری بر نگرداند نوع مقدار برگشتی تابع را **void** اعلان می‌کنیم. و در صورتیکه تابع مقداری را دریافت نکند بجای **parameter- list** از **void** یا ( ) استفاده می گردد.

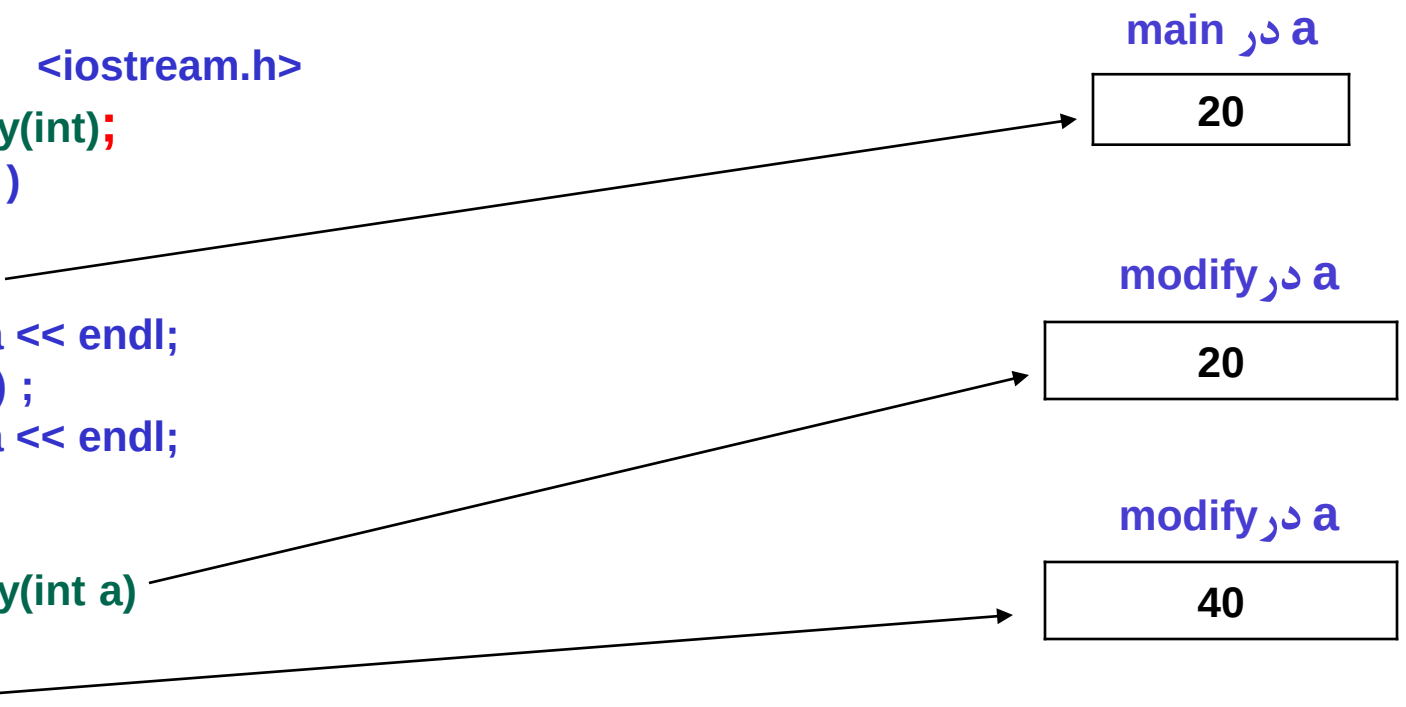
در اسلاید بعد مثالی در این زمینه آورده شده است.

```
#include    <iostream.h>
#include   <conio.h>
void maximum(int , int) ;
int main( )
{ int x, y;
cin >> x >> y;
maximum(x,y);
return 0;
}
void maximum(int x, int y)
{
int z ;
z=(x>=y) ? x : y ;
cout << "max value \n" << z<< endl;
return ;
}
```

تابع مقداری بر نمی گرداند.

احضار بوسیله مقدار (Call By Value)

```
#include    <iostream.h>
int modify(int);
int main( )
{
int a=20;
cout << a << endl;
modify(a) ;
cout << a << endl;
return 0 ;
}
int modify(int a)
{
a *= 2;
cout << a << endl;
return 0;
}
```



خروجی برنامه :

20

40

20

احضار بوسیله مقدار (Call By Value)

```
#include   <iostream.h>
int modify(int);
int main( )
{
int a=20;
cout << a << endl;
modify(a) ;
cout << a << endl;
return 0 ;
}
int modify(int a)
{
a *= 2;
cout << a << endl;
return 0;
}
```

a در main

20

a در modify

20

a در modify

40

در این نوع احضار تابع حافظه‌های مورد استفاده آرگومان ها و پارامترها از هم متمایزند و هرگونه تغییر در پارامترها باعث تغییر در آرگومان های متناظر نمی‌گردد.

تابع بازگشتی (recursive functions)

توابع بازگشتی یا **recursive** توابعی هستند که
وقتی احضار شوند باعث می‌شوند که خود را احضار نمایند.

نحوه محاسبه فاکتوریل از طریق تابع بازگشتی

$n != 1*2*3*\ldots*(n-1)*n$

$f(n) = n !$

$$f(n) = \begin{cases} 1 & \text{اگر } n=1/\ n=0 \\ n * f(n-1) & \text{در غیر اینصورت} \end{cases}$$

$n!= 1*\ 2*\ 3*\ \ldots\ *(n-2)\ *(n-1)\ *n$

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

در اسلاید بعد تابع بازگشتی مورد نظر پیاده سازی شده است.

تابع بازگشتی محاسبه فاکتوریل

```
#include    <iostream.h>
long int factorial(int) ;
int main( )
{
     int n ;
     cout << " n= " ;
     cin >> n ;
     cout << endl << " factorial = " << factorial(n) << endl;
     return 0 ;
}
long int factorial(int n)
{
     if(n<=1)
          return(1);
     else
          return(n *factorical(n-1) ) ;
}
```

نحوه محاسبه n امین مقدار دنباله فیبوناچی از طریق تابع بازگشتی

دنباله فیبوناچی : 0 , 1, 1, 2, 3, 5, 8, 13, 21 , 34, …

جمله n ام = fib(n)=

- اگر n=1 : 1
- اگر n=2 : 1
- در غیر اینصورت : fib(n-1)+fib(n-2)

در اسلاید بعد تابع بازگشتی مورد نظر پیاده سازی شده است.

برنامه‌زیر n امین مقدار دنبالهٔ فیبوناچی (fibonacci) را مشخص و نمایش می‌دهد.

```
#include    <iostream.h>
long int fib(long int); // forward declaration
int main( )
{
      long int r ;
      int n ;
      cout << ″ Enter an integer value ″ << endl ;
      cin >> n ;
      r = fib(n) ;
      cout << r << endl ;
      return 0 ;
}
long int fib(long int n)
{ if (n==0) return 0;
else
      if(n = = 1  ||   n= = 2)
            return 1 ;
      else
            return(fib(n-1) + fib(n-2) ) ;
}
```

برنامه زیر یک خط متن انگلیسی را گرفته آن را وارون نموده نمایش می‌دهد.

```
# include <iostream>
# include <conio.h>
using namespace std;
void reverse(void) ; // forward declaration
int main( )
{
reverse( ) ;
getch();
return 0 ;
}
void reverse(void)   // read  a line of characters and reverse it
{
char c ;
if( (c=cin.get()) != '\n')
reverse( );
cout << c ;
return ;
}
```

انتقال پارامترها از طریق ارجاع

تاکنون وقتی تابعی را احضار می‌کردیم یک کپی از مقادیر آرگومان ها درپارامترهای متناظر قرار می‌گرفت . این روش احضار بوسیله مقدار یا **call by value** نامیده شد.
در انتقال پارامترها از طریق ارجاع در حقیقت حافظه مربوط به آرگومان ها و پارامترهای متناظر بصورت اشتراکی مورد استفاده قرار می‌گیرد. این روش **call by reference** نامیده می‌شود .

انتقال پارامترها از طریق ارجاع

در این روش پارامترهائی که از طریق call by reference عمل می‌نمایند در پیش نمونه تابع قبل از نام چنین پارامترهائی از & استفاده می‌شود.واضح است که در تعریف تابع نیز بهمین طریق عمل می‌شود.

```
#include    <iostream.h>
int vfunct(int);
void rfunct (int &) ;
int main( )
{
      int x=5, y=10;
      cout << x << endl << vfunct(x) << endl << x << endl ;
      cout  << y << endl ;
      rfunct(y) ;
      cout << y << endl ;
      return 0 ;
}
int vfunct(int a)
{
      return a *= a ;
}
void rfunct(int &b)
{
      b *= b ;
}
```

مثال :

x	y
5	10

خروجی :

5
25
5
۱۰

مقدار آرگومان x تغییر نمی کند.

x	y b
5	10 100

ادامه خروجی :

100

نکته :

وقتی پارامتری بصورت **call by reference** اعلان می‌گردد این بدان معنی است که با تغییر مقدار این پارامتر در تابع احضار شده مقدار آرگومان متناظر نیز تغییر می‌نماید.

برنامه‌زیر با استفاده از fswap دو مقدار اعشاری را مبادله می‌نماید.

```
#include    <iostream.h>
void fswap(float & , float & );
int main( )
{
      float a=5.2, b=4.3;
      cout << a << endl << b ;
      fswap( a , b) ;
      cout << a << endl << b ;
      return 0 ;
}
void fswap(float &x , float & y)
{
      float t;
      t = x ;
      x = y ;
      y = t ;
}
```

توابع درون خطی (inline)

کلمه **inline** بدین معنی است که به کامپایلر دستور می‌دهد که یک کپی از دستورالعمل های تابع در همان جا (در زمان مقتضی) تولید نماید تا از احضار تابع ممانعت بعمل آورد.

اشکال توابع inline

بجای داشتن تنها یک کپی از تابع ، چند کپی از دستورالعمل های تابع در برنامه اضافه می‌شود که باعث بزرگ شدن اندازه یا طول برنامه می‌شود.
بنابراین از **inline** برای توابع کوچک استفاده می‌گردد.

مثالی از توابع درون خطی

```
#include    <iostream.h>
inline float cube(const float s) {return s*s*s; }
int main( )
{
float side ;
cin >> side ;
cout << side << cube(side) << endl ;
return 0 ;
}
```

کلاس‌های حافظه (storage classes)

متغیرها بدو طریق متمایز مشخص می‌شوند یکی بوسیله نوع (type) آن ها و دیگری بوسیله کلاس حافظه آن ها. نوع متغیر قبلاً اشاره شده بعنوان مثال int ، float ، double ، . . . ولی کلاس حافظهٔ یک متغیر در مورد طول عمر و وسعت و دامنهٔ متغیر بحث می‌نماید.

در اسلاید بعد به انواع کلاس حافظه می پردازیم.

بطور کلی کلاس حافظه متغیرها به چهار دستهٔ تقسیم می‌گردد :

۱. automatic
۲. static
۳. external
۴. register

automatic

متغیرهای automatic در درون یک تابع تعریف می‌شوند و در تابعی که اعلان می‌شود بصورت متغیرهای محلی برای آن تابع می‌باشند. حافظه تخصیص داده شده به متغیرهای automatic پس از اتمام اجرای تابع از بین می رود بعبارت دیگر وسعت و دامنهٔ متغیرهای از نوع automatic تابعی می‌باشد که متغیر در آن اعلان گردیده است.

static

متغیرهای **static** نیز در درون توابع تعریف می شوند و از نظر وسعت و دامنه شبیه متغیرهای **automatic** هستند ولی در خاتمهٔ اجرای تابع، حافظه وابسته به این نوع متغیرهـــــا از بین نمی‌رود بلکه برای فراخوانی بعدی تابع باقی می‌ماند.

و به دودسته تقسیم می شوند

1- متغیر های استاتیک محلی

2- متغیر های استاتیک عمومی

مقدار اولیه متغیرهای استاتیک محلی و عمومی صفر است.

در اسلاید بعد به یک مثال از کاربرد این نوع کلاس حافظه می پردازیم.

استاتیک محلی

- فقط در همان تابعی که تعریف می شوند قابل استفاده اند.
- هنگام فراخوانی تابع ایجاد می شوند و به هنگام خروج تابع ، آخرین مقدار خودشان را حفظ می کنند.
- فقط یک با مقدار اولیه می گیرند.

در اسلاید بعد به یک مثال از کاربرد این نوع کلاس حافظه می پردازیم.

مثال :

```
#include    <iostream.h>
#include     <conio.h>
Void   test(void);
int main( )
{
Int    i;
For (i=0 ; i<5; i++)
      test();
return 0 ;
}
Void  test(void)
{
Static  int y=0;
Int    x=0;
Cout<<“\n outo x= “<<x<<“, static y=“<<y;
X++;
Y++;
}
```

مثال :

```
#include    <iostream.h>
// program to calculate successive fibonacci numbers
long int fib(int) ;
int main( )
{
int n ;
cout << ″ how many fibonacci numbers?″ ;
cin >> n ;
cout << endl ;
for(int j=1; j<=n; ++j )
cout << j << ″   ″ << fib(j) << endl ;
return 0 ;
}
long int fib(int count)
{
static long int t1 = 1, t2=1;
long int  t ;
t =(count <3) ?1 : t1 + t2 ;
t2 = t1 ;
t1 = t ;
return(t) ;
}
```

بایستی توجه داشت که اگر در توابع به متغیرهای از نوع static مقدار اولیه تخصیص ندهیم مقدار صفر بصورت اتوماتیک برای آنها در نظر گرفته می‌شود.

external

متغیرهای از نوع external متغیرهائی هستند که در بیرون از توابع اعلان می شوند و وسعت و دامنه فعالیت آنها کلیه توابعی می‌باشد که در زیر دستور اعلان متغیر قرار دارد.

در اسلاید بعد به یک مثال از کاربرد این نوع کلاس حافظه می پردازیم.

مثال :

```
#include    <iostream.h>
External int w; // external variable
functa(int x, int y)
{
cout << w ;
w = x + y ;
cout << endl << w << enld;
return x%y ;
}
int main( )
{
int a, b, c, d;
cin >> a >> b ;
c=functa(a, b) ;
d=functa(w, b+1);
cout << endl << c << endl << d << endl << w ;
return 0 ;
}
```

بایستی توجه داشت که اگر در توابع به متغیرهای از نوع external مقدار اولیه تخصیص ندهیم مقدار صفر بصورت اتوماتیک برای آنها در نظر گرفته می‌شود.

register

وقتی متغیری از نوع **register** اعلان می‌شود از کامپیوتر عملاً درخواست می‌شود که به جای حافظه از یکی از رجیسترهای موجود استفاده نماید.

کاربرد کلاس register

معمولاً از نوع رجیستر برای شاخص‌های دستور تکرار و یا اندیس های آرایه‌ها استفاده می‌شود. بایستی توجه داشت که متغیرهای از نوع رجیستر قابل استفاده دردستور **cin** نمی‌باشند

```
register int i;
for(i=1;i<=10;i++)
    cout<<“hi”;
```

سربارگذاری توابع (function overloading)

در C++ این امکان وجود دارد که دریک برنامه بتوانیم از چند توابع هم نام استفاده نمائیم مشروط بر این که پارامترهای این توابع متفاوت باشند. (از نظر تعداد پارامتر و یا نوع پارامترها و ترتیب آنها)

مثال :

```
#include    <iostream.h>
float addf(float , int);
int addf(int , int);
int main( )
{
int a=5, b=10 ;
float d=14.75 ;
cout << addf(a , b) << endl;
cout << addf(d ,  b) << endl;
return 0 ;
}
int addf(int x, int y)
{
return x+y ;
}
float addf(float x, int y)
{
return x+y ;
}
```

برنامه ای که با استفاده ار تابعی به نام Square() ، مربع یک عدد را محاسبه می کند

دقت شود این برنامه باید بتواند مربع یک عدد صحیح و اعشاری را با استفاده از توابع سرباگذاری محاسبه نمایید.

تمرین

۱. برنامه ای بنویسید که ۳ عدد اعشاری را خوانده و به تابعی ارسال کند و تابع میانگین آن ها را محاسبه کرده و برگرداند.

۲. برنامه ای که ضرایب معادله ی درجه دومی را خوانده، آن ها را به تابعی ارسال کند. تابع معادله را حل کند و جواب ها را در خروجی چاپ کند. (تابع جواب ها را به تابع اصلی برنمی گرداند.)

۳. برنامه ای بنویسید که عددی را از ورودی خوانده و هریک از ارقام آن را در یک سطر چاپ کند. تفکیک و چاپ ارقام توسط تابع بازگشتی صورت گیرد.

۴. برنامه ای بنویسید که حاصلضرب دو عدد صحیح را به کمک جمع کردن محاسبه کند. برای این کار از تابع بازگشتی استفاده کنید.

$$a \times b = \begin{cases} a & \text{اگر } b=1 \\ a \times (b-1)+a & \text{اگر } b>1 \end{cases}$$

1

```
# include <iostream>
# include <conio.h>
using namespace std;
float average(float a, float b, float c){return ((a+b+c)/3);}

void main()
{
float x,  y,  z;
cin>>x>>y>>z;
cout<<average(x,y,z);
getch();
}
```

2

```
# include <iostream>
# include <conio.h>
# include <math.h>
using namespace std;
void f(int a, int b, int c){
float  delta = (b*b)-(4*a*c);
    if(delta == 0)
                  cout<<"The only answer:n"<<"x= "<<(-b)/(2*a);
else if(delta > 0)
    {
        cout<<"Two answers:n"<<"x1= "<<(-b+sqrt(delta))/(2*a);
        cout<<"x2= "<<(-b-sqrt(delta))/(2*a);
    }
}
void main() {

float x,  y,  z;
cin>>x>>y>>z;
f(x,y,z);
getch();
}
```

3

```
# include <iostream>
# include <conio.h>
# include <math.h>
using namespace std;
void f(int a)
{if (a>10) f(a/10);
cout<<(a%10)<<endl;
}
void main()
{
int n;
cin>>n;
f(n);
getch();
}
```

4

```
# include <iostream>
# include <conio.h>
# include <math.h>
using namespace std;
int f(int a,int b)
{if (b==1) return a;
return (f(a,b-1)+a);
}
void main()
{
int n,m;
cin>>n>>m;
cout<<f(n,m);
getch();
}
```

فصل هفتم

ساختار ها و اشاره گرها

فهرست مطالب فصل هفتم

۱. ساختارها
۲. Union ها
۳. اشاره گرها (Pointer)
۴. تعریف آرایه
۵. آرایه های دو بعدی و اشاره گرها
۶. تخصیص حافظه بصورت پویا (عملگر new)
۷. رشته ها و توابع مربوطه

ساختارها

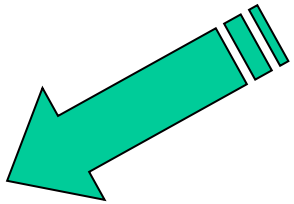
ساختارها شبیه آرایه‌ها بوده بدین صورت که یک نوع داده گروهی است که فضای پیوسته از حافظه اصلی را اشغال می‌نماید. اما عناصر ساختار الزاماً از یک نوع نمی‌باشند بلکه اعضای یک ساختار می‌توانند از نوع‌های مختلف از قبیل **char** ، **int** ، **float** ، **...** باشند.

تعریف ساختار

نام ساختار

```
struct time
{
int hour ; // 0 – 23
int minute ; // 0 – 59
int second; //
} ;
```

اعضا ساختار



مثال :

```
struct account {
int acc_no ;
char acc_type;
char name[80] ;
float balance ;
};
```

ساختار account دارای چهار عضو می‌باشد.

acc_no شماره حساب از نوع int

acc_type نوع حساب از نوع char

name مشخصات صاحب حساب از نوع رشتهٔ 80 کرکتری

balance مانده حساب از نوع float

به دو صورت می توان اعلان یک متغیر از نوع ساختار را نمایش داد :

روش اول :

```
struct account    {
                  int acc_no;
                  char acc_type;
                  char name[80];
                  float balance;
                  } cust1, cust2, cust3;
```

روش دوم :

```
struct account    {
                  int acc_no ;
                  char acc_type;
                  char name[80];
                  float balance;
                  };
account cust1, cust2, cust3;
```

به ساختارها می‌توان مقدار اولیه نیز تخصیص داد

```
account cust = {4236, 'r', "Nader Naderi" , 7252.5};
```

دسترسی به عناصر یک ساختار

بمنظور دسترسی به عناصر یک ساختار از عملگر • استفاده می‌گردد . عملگر. جزء عملگرهای یکتائی می‌باشد.

مثال :

```
cust .acc_no = 4236;
cust .acc_type = 'r';
cust . name = "Nader Naderi";
cust . balance = 7252.5;
```

نکته :

عضو یک ساختار خود می‌تواند یک ساختار دیگر باشد.

```
struct date {
        int month;
        int day;
        int year;
        };
struct account {
        int acc_no ;
        char acc_typer;
        char name[80];
        float balance ;
        date lastpay ;
        };
```

اگرداشته باشیم

```
account x, y ;
```

آنگاه عضو lastpay بوسیله

```
x.lastpay.day
x.lastpay.month
x.lastpay.year
```

مشخص می‌گردد.

نکته :

می‌توان آرایه‌ای تعریف نمود که هر عضو آن یک ساختار باشد و حتی به آنها مقادیر اولیه تخصیص نمود.

```
struct struc1 {
        char name[40];
        int pay1;
        int pay2;
        } ;
struc1 cust[  ]= {"nader", 3000 , 40000,
                  "sara", 4200, 6000,
                  "susan", 3700, 25000,
                   "saman", 4800 , 2000, };
```

برنامه زیر هر عدد مختلط را بصورت یک ساختار در نظر گرفته، دو عدد مختلط را می‌گیرد و مجموع آن ها را مشخص و نمایش می‌دهد.

```
#include   <iostream.h>
int main( )
{
struct complex{
                        float  a;
                        float  b; } x, y, z;
cout << "enter 2 complex numbers" << endl ;
cin >> x.a>>x.b ;
cout << endl;
cin >> y.a >> y.b;
z.a = x.a + y.a ;
z.b = x.b + y.b ;
cout << endl << z.a << "   "<< z.b;
return 0 ;
}
```

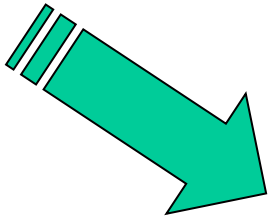
---

union

union از نظر ساختاری شبیه **struct** می‌باشد . با این تفاوت که عضوهائی که تشکیل **union** میدهد همگی از حافظه مشترکی در کامپیوتر استفاده می‌نمایند. بنابراین استفاده از **union** باعث صرفه‌جوئی در حافظه می‌گردد.

مثال :

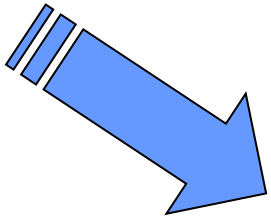
```
union id
{
char color [10];
int size;
} x , y;
```



هر کدام از متغیرهای x و y یک رشته 10 کرکتری یا یک مقدار از نوع int می‌باشد وکامپیوتر یک بلوک حافظه که بتواند رشته ۱۰ کرکتری رادر خود جای دهد ، برایcolor و size در نظر می‌گیرد.

مثال :

```
union xpq
 {
int x ;
char y[2] ;
} p ;
```



x

بایت دوم	بایت اول
y[1]	y[0]

تمرین:

برنامه ای بنویسیم که یک دفترچه تلفن ۲۰ تایی (ظرفیت پذیرش ۲۰ نام و شماره تلفن را داشته باشد)و امکان جستجو در آن را بر اساس نام فرد ایجاد نماید.

اشاره‌گرها (Pointers)

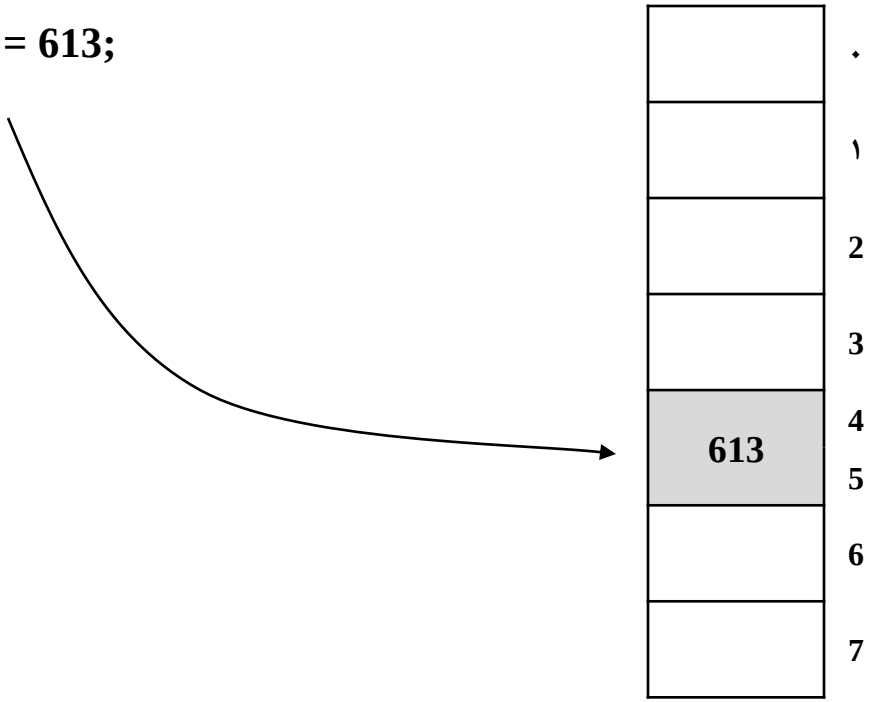
داده‌هائی که در کامپیوتر در حافظه اصلی ذخیره میشوند
بایت‌های متوالی از حافظه بسته به نوع data اشغال می‌کنند.

نوع داده	مقادیر	حافظه لازم
int	32767 تا –32768	۲ بایت
long int	2147483647 تا –2147483648	۴ بایت
char	یک کارکتر	۱بایت
float	3.4e38 تا 1.2e-38	۴ بایت
double	1.8e308 تا 2.2e-308	۸ بایت

اشاره‌گرها (Pointers)

با داشتن آدرس داده در حافظهٔ اصلی می‌توان براحتی به آن داده دسترسی پیدا نمود و از طرف دیگر آدرس هر داده در حافظه آدرس بایت شروع آن داده می‌باشد.

```
int  x = 613;
```



نکته :

در کامپیوتر آدرس‌ها معمولاً دو بایت اشغال می‌نمایند. اگر آدرس x را در px قرار دهیم آنگاه می‌گوئیم که px به x اشاره می‌نماید.

آدرس متغیر x را بوسیله &x نشان میدهیم و عملگر & را عملگر آدرس می‌نامند.

```
int  x , *px
X=54;
px = &x ;
```

مثال :

```
int y , x , *px ;
x = 26 ;
px = &x ;
```

حال اگر دستور العمل ; 10 =+ x را بدهیم :

حال اگر دستورالعمل ; 7 + *px = *px بدهیم.

آرایه یک بعدی و اشاره گرها

0	26.5	x
1	24.7	
2	5.8	
3	-73.2	
4	69.0	
5	100.5	
6	-13.24	
7	424.3	
8	187.8	
9	358.2	

اولین عنصر آرایه بوسیله x[0] مشخص می‌شود.

آدرس اولین عنصر آرایه بوسیله &x[0] یا بوسیله x مشخص می‌شود.

آدرس i امین عنصر آرایه بوسیله &x[i] یا بوسیله (x+i) مشخص می‌شود.

دو دستورالعمل زیر با هم معادلند .

```
x[ i ] = 82.5 ;
*(x + i) = 82.5 ;
```

از طرف دیگر اگر داشته باشیم

```
float x[10];
float *p;
```

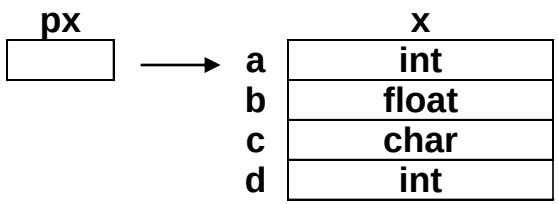
دو دستورالعمل زیر معادلند.

```
p = &x[2] ;
p= x + 2 ;
```

ساختارها و اشاره گرها

می‌توان اشاره‌گری را تعریف نمود که به اولین بایت یک ساختار (struct) اشاره نماید.

```
struct    struc1
{
        int a ;
        float b ;
        char  c;
        int d ;
} x, *px ;

px = &x ;
```



عبارت x.a معادل px–>a معادل (*px).a می‌باشد.

نکته :

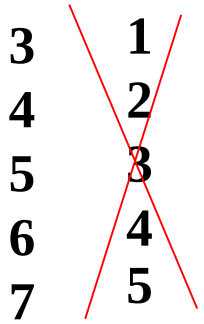
استفاده از آرایه‌ها بعنوان پارامتر تابع مجاز است.

در اسلاید بعد به یک مثال توجه نمایید.

در برنامه زیر تابع modify آرایه a را بعنوان پارامتر می‌گیرد.

```
#include    <iostream.h>
void  modify(int [  ] ); // forward declaration
int main( )
{
int  a[5] ;
for(int j=0; j<=4; ++j)
a[ j ] = j+1 ;
modify(a) ;
for(int j=0; j<5; ++j)
cout << a[ j ] << endl ;
return 0 ;
}
void modify(int a[  ]) // function definition
{
for(int j=0; j<5; ++j)
a[ j ] += 2 ;
for(int j=0; j<5; ++j)
cout << a[ j ] << endl ;
return ;
}
```

خروجی :



3
4
5
6
7

نکته: در این مثال از فراخوانی با مقدار استفاده نشده است. برای همین تغییرات انجام شده در تابع modify() باعث تغییر در مقادیر ارایه در تابع اصلی یا main() شده است. این مبحث، تحت عنوان فراخوانی از طریق ارجاع در ادامه توضیح داده می شود.

نکته :

در صورتیکه آرایه بیش از یک بعد داشته باشد بعدهای دوم به بعد بایستی در تعریف تابع و پیش نمونه تابع ذکر گردد.

در اسلاید بعد به یک مثال توجه نمایید.

```
#include    <iostream.h>
void printarr(int [   ][ 3 ]);
int main( )
{
      int  arr1 [2][3] = { {1,2,3}, {4,5,6} } ,arr2 [2][3]= {1,2,3,4,5} ,  arr3 [2][3]={ {1,2}, {4} };
      printarr(arr1);
      cout << endl ;
      printarr(arr2);
      cout << endl ;
      printarr(arr3);
      return 0 ;
}
void printarr(int a[  ][3] )
{
      for(int i=0; i<2; i++)
      {
            for(int j=0; j<3; j++)
                  cout << a[ i ][ j ] << ' ';
      cout << endl ;
      }
}
```

خروجی :

```
1   2   3
4   5   6

1   2   3
4   5   0

1   2   0
4   0   0
```

آرایه‌های دوبعدی و اشاره‌گرها

یک آرایه دوبعدی بصورت تعدادی آرایه یک بعدی می‌توان تعریف نمود.
اگر x یک ماتریس 5 سطری و 4 ستونی از نوع اعشاری باشد قبلاً این ماتریس را با

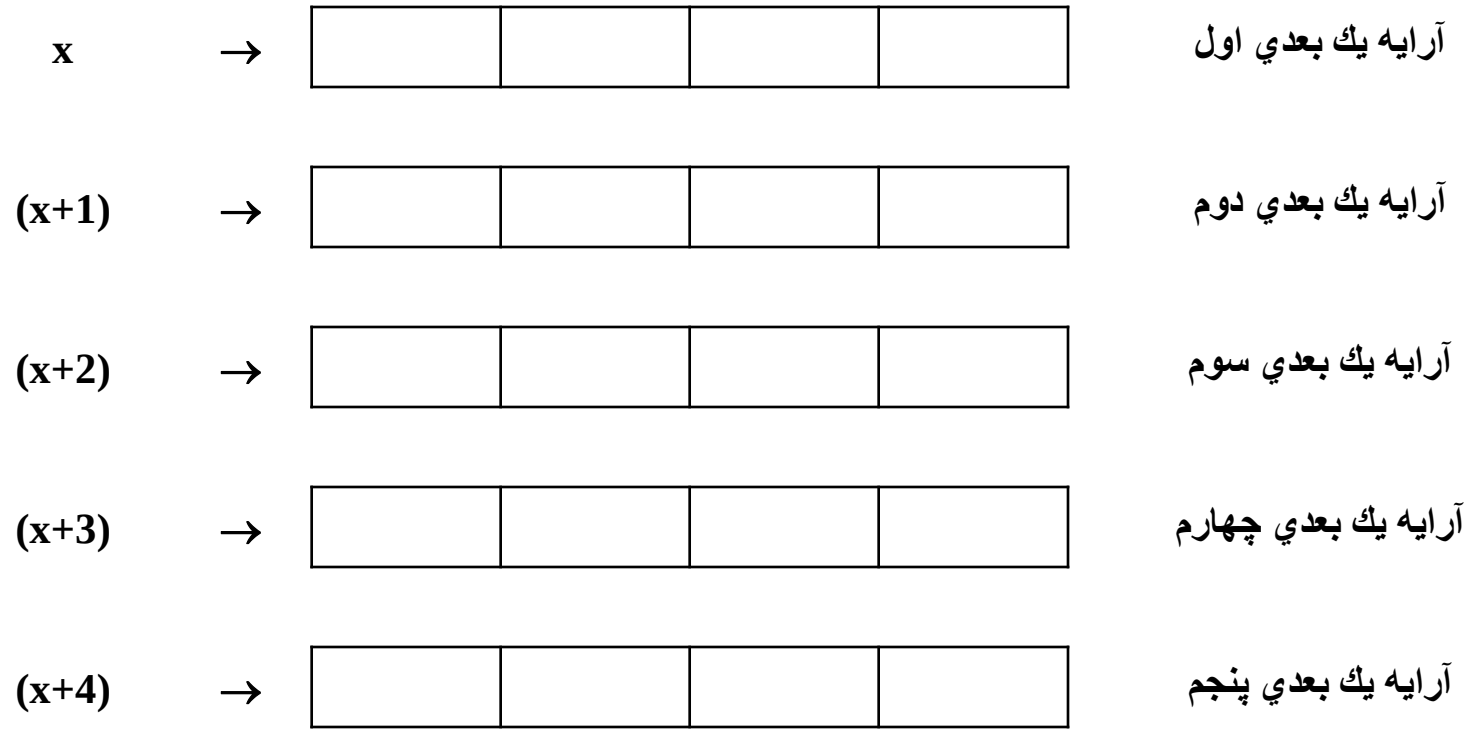
```
float x[5][4];
```

معرفی کردیم. حال با استفاده از اشاره‌گرها بصورت زیر معرفی نمائیم:

```
float  (*x)[4];
```

آرایه‌های دوبعدی و اشاره‌گرها

```
float  (*x)[4];
```



در برنامه زیر یک آرایه 5 عنصری از نوع int ایجاد شده و مقادیر عناصر آرایه را به چهار طریق نمایش می‌دهد.

```
#include      <iostream.h>
int main( )
{
int x[ ]={12, 25, 6, 19, 100}, i;
int *px=x; //  نام آرایه بدون اندیس، اشاره به عنصر اول آرایه می‌نماید

for(i=0; i<=4; i++)
cout << *(x+i) << endl;

//the second method
for(i=0; i<5; i++)
cout << x[ i ] << ′ \n′;

//the third method
for(i=0; i<=4; i++)
cout << px[ i ]<<endl;

//the forth method
for(i=0; i<=4; i++)
cout << *(px+i)<<endl;

return 0; }
```

تخصیص حافظه به صورت پویا یا (عملگر new)

از عملگر **new** برای تخیصیص حافظه به صورت پویا می توان استفاده نمود ، در ضمن می‌توان برای بلوکی از حافظه که تخصیص یافته مقدار اولیه تعیین نمود.

new

delete

dynamic memory allocation

برای تخصیص حافظه باندازه 20 مقدار از نوع int که اشاره‌گر ptx به آن اشاره نماید بصورت زیر عمل می‌شود.

```
int  *ptx;
ptx = new   int [20];
```

ptx به اولین داده از نوع int اشاره می‌نماید .

ptx+i به i+1 امین عنصر از فضای پیوسته اشاره می‌نماید.

بازگرداندن فضای یک آرایهٔ یک بعدی پویا

اگر در برنامهٔ خود با استفاده از عملگر new ، مقدار زیادی حافظه از سیستم اخذ کنید آنگاه تمام فضای امکانپذیر موجود درRAM ، در اختیار یک برنامه قرار می‌گیرد و ممکن است سیستم از کار بیفتد (به اصطلاح سیستم Hang می‌کند). برای اطمینان از کاربرد مطمئن عملگر new ، باید بین عملگر new و عملگر delete متناظر با آن هماهنگی لازم را به‌وجود آورید تا حافظهٔ اخذ شده را به سیستم عامل برگردانید. به‌طور کلی هرگاه که با مقداری حافظه که به‌طور پویا از سیستم اخذ کرده‌اید کارتان به اتمام رسید، بهتر است آن را بی‌درنگ به سیستم‌عامل برگردانید. نحوهٔ بازگرداندن حافظهٔ تخصیص یافته به یک آرایهٔ یک‌بعدی‌پویا به‌صورت زیراست.

```
delete[]   نام اشاره‌گر ;
```

برنامه زیر یک فضای n عنصری از نوع اعشاری در حافظه ایجاد نموده، سپس آن را مقدار داده و مجموع مقادیر رامشخص و نمایش می‌دهد.

```
#include      <iostream.h>
int main( )
{
int n;
float *ptr, tot = 0.0;
cout << ″enter a value for n  ″ << endl;
cin >> n;
ptr=new float [n];
for(int j=0; j<n; ++j)
      {
            cin >> *(ptr + j);
            cout << ′ \n ′;
       }
for(j=0; j<=n-1; ++j)
                  tot += *(ptr + j);
cout << tot ;
// in order to free the space use
delete[ ]    ptr ;
return 0;
 }
```

برنامه زیر آرایه‌های n عنصری از ساختار را ایجاد می‌نماید.

```
#include      <iostream.h>
#include      <conio.h>
int main( )
{
struct rec
 {
              float  a;
              int  b;
 } ;
int n;
rec  *ptr;
cout << "how many records? \n";
cin >> n ;
ptr = new  rec[n];
for(int  i=0; i<n; ++i) {
                          cout <<((ptr+i) ->b=i) << "    ";
                         cout <<((*(ptr +i)).a = i+0.5) << endl ;
                         }
delete  [ ] ptr;
getch();
return 0;
}
```

برنامه زیر دو مقدار اعشاری را گرفته مقادیر آن ها را به کمک تابع swap جابه‌جا می‌نماید.

```
#include      <iostream.h>
#include     <conio.h>
void swap(float *, float *);
int main( )
{
float a,b,*ptb,*pta;
cin >> a >> b;
pta=&a;
ptb=&b;
swap(pta,ptb);  ///////swap(a,b);
cout << a << endl << b << endl ;
return 0;
}
void swap(float *px , float *py)
{
   float t;
   t = *px;
   *px = *py;
   *py = t ;
    return;
 }
```

برنامه ای که تعداد n عدد را از ورودی خوانده ، در آرایه ای قرار می دهد و سپس آرایه را به تابعی ارسال می کند .تابع تعداد اعداد زوج و فرد را نمایش دهد.

رشته‌ها و توابع مربوطه

رشته‌ها در C++ ، آرایه‌ای از کرکترها می‌باشند که با کرکتر '\0' ختم می شوند.

```
char name[ ]= " sara";
```

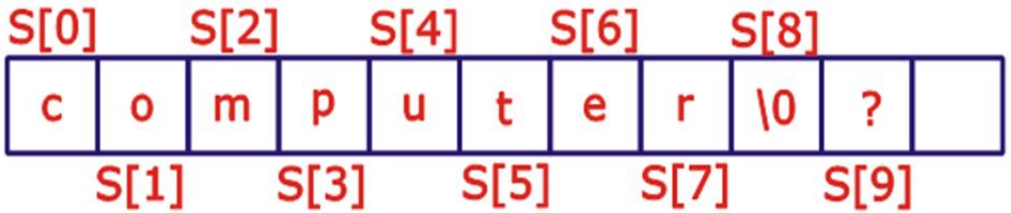
s
a
r
a
\0

نحوه رفتار با رشته‌ها همانند رفتار با آرایه‌ها می‌باشد ولی یک تفاوت اساسی در انتهای رشته‌ها می‌باشد که یک علامت نشان‌دهنده انتهای رشته‌ها می‌باشد ولی در آرایه‌ها هیچ علامتی انتهای آرایه‌ها را مشخص نمی‌کرد.

علامت '\0' انتهای رشته‌ها مشخص می‌کند. مثلاً در دستور

```
char S[10];
```

و اگر رشته را با 'computer' مشخص نماییم به صورت زیر در حافظه جا می‌گیرد.



مقداردهی اولیه به رشته

$$char\ \ S1[\ ] = \text{“}IT\text{”};$$

$$char\ \ S2[10] = \text{“}is\text{”};$$

$$char\ \ S3[\ ] = \{\text{‘}N\text{’}, \text{‘}e\text{’}, \text{‘}w\text{’}, \text{‘ ’}, \text{‘}L\text{’}, \text{‘}i\text{’}, \text{‘}v\text{’}, \text{‘}e\text{’}, \text{‘}\backslash 0\text{’}\};$$

اگر همانند دو دستور بالا عمل کنیم خود کامپایلر اتوماتیک در آخر رشته ‘\0’ قرار می‌دهد ولی در حالت سوم باید حتماً خودمان به رشته ‘\0’ را بدهیم.

نکته قابل توجه این است که رشته‌ها همیشه یک کارکتر بیشتر بخاطر ‘\0’ فضا می‌گیرند.

خواندن و نوشتن رشته:

برای خواندن رشته‌ها از ورودی طبق رویه زیر عمل می‌نماییم.

```
char  str[51];

cin >> str;

cout <<"string  is :" << str;
```

در این مثال یک رشته با طول ۵۰ فرض کردیم (یک کارکتر بخاطر '\0' در نظر نگرفتیم) و بوسیله *cin* از ورودی خواندیم و با تابع *cout* در خروجی چاپ نمودیم.

کاربر اگر کلید *Enter* را در مثال فشار دهد رشته *str* مقدار می‌گیرد یا اگر *space* یا *tab* را فشار دهد دوباره *cin* مقدار می‌گیرد.

استفاده از متد *get* :

تابع *get* را می‌توان به شکل زیر بکار برد.

cin.get (نام رشته و طول رشته [,'جداکننده'])

قسمت 'جداکننده' اختیاری می‌باشد و می‌تواند نباشد. این قسمت یک کارکتر را به عنوان انتهای رشته مشخص می‌کند.

طول رشته:

حداکثر مکانی را مشخص می‌کند که رشته می‌تواند بگیرد.

مثلاً:

```
char str[30];

cin.get(str,10);

cin.get(str , 10 , '.');
```

در دستور دوم حداکثر می‌توانیم با طول ۱۰ به *str* از ورودی مقدار دهیم و دستور سوم بجای *Enter* که آخر رشته باشد با ' . ' مشخص می‌شود.

تفاوت get( ) و cin

همانگونه که گفته شد انتهای ورودی دستور cin را یا Enter مشخص می‌کند یا space ولی در cin

می‌توانیم فاصله و tab را نیز به رشته بدهیم. مثلاً

```
char S[101];

cin.get(S,50);

cin >> S;
```

اگر کاربر رشته "IT is now live" را به برنامه بدهد تابع get ( ) همه را در C جا می‌دهد ولی cin فقط "IT" را

در S قرار می‌دهد.

رشته و اشاره گر

هر رشته از طریق اشاره‌گری به اولین کرکتر آن در دسترس قرار می‌گیرد. آدرس یک رشته، آدرس کرکتر اول آن می‌باشد. به رشته‌ها می‌توان مقدار اولیه تخصیص داد.

```
char    *name = " sara";
```

انتساب رشته‌ها

دو رشته را به صورت مستقیم نمی‌توانیم در داخل یکدیگر بریزیم.

دستور $strcpy(S1,S2)$ مقدار $S2$ را در داخل $S1$ کپی می‌نماید.

```
char  S1[ ]="computer";

char  S2[20];

strcpy(S2,S1);
```

مقایسه رشته‌ها

برای مقایسه رشته نمی‌توانیم به این صورت عمل نماییم. $if\ (S1==S2)$

چون $S2, S1$ اشاره‌گر به رشته می‌باشند ولی دستوری بنام $strcmp(S1, S2)$ وجود دارد که دو رشته $S2, S1$ را مقایسه می‌نماید.

$S_1 \mid S_2$	$S_1 = S_2$	$S_1 > S_2$	$S_1 < S_2$
$strcmp(S_1, S_2)$	۰	عدد مثبت	عدد منفی

$S_1 \mid S_2$	$S_1 = S_2$	$S_1 > S_2$	$S_1 < S_2$
$strcmp(S_1, S_2)$	۰	عدد مثبت	عدد منفی

عدد برگردانده شده از تابع *strcpy* طبق جدول بالا می‌باشد. منظور از اینکه دو رشته از هم کوچکتر باشد یا بزرگتر به این صورت می‌باشد که سمت چپ‌ترین کارکتر را با کد اسکی مقایسه می‌کند اگر بزرگتر بود که آن رشته از رشته دوم بزرگتر می‌باشد. اگر کوچکتر بود، آن رشته کوچکتر می‌باشد و اگر مساوی بود کارکتر بعدی را نگاه می‌کند.

الحاق دو رشته:

برای الحاق دو رشته از تابع $strcat\,(S1,S2)$ استفاده می‌نماییم. این تابع $S2$ را به انتهای $S1$ وصل می‌نماید.

تابع strcmpi(s1, s2)

رشته‌های s1 و s2 را با هم مقایسه نموده (بدون توجه به حروف کوچک و بزرگ) اگر رشته s1 برابر با رشته s2 باشد مقدار صفر و اگر رشته s1 کوچکتر از رشته s2 باشد یک مقدار منفی در غیر اینصورت یک مقدار مثبت بر می‌گرداند.

```
char *s1= "ALI"; // char s1[10] = "ALI";
char *s2="ali"; //char s2[10]="ali";
cout << strcmpi(s1, s2) << endl;
```

```
0
```

تابع strcmp(s1, s2)

رشته‌های s1 و s2 را با هم مقایسه نموده اگر s1 برابر با s2 باشد مقدار صفر و اگر رشته s1 کوچکتر از رشته s2 باشد یک مقدار منفی در غیر اینصورت یک مقدار مثبت برمی‌گرداند.

```
char *s1= "ALI";
char *s2="ali";

char s1[10]= "ALI";
char s2[10]="ali";

cout << strcmp(s1, s2) << endl;
```

-1

تابع strncmp(s1, s2,n)

حداکثر n کرکتر از رشتهٔ s1 را با n کرکتر از رشتهٔ s2 مقایسه نموده در صورتیکه s1 کوچکتر از s2 باشد یک مقدار منفی، اگر s1 مساوی با s2 باشد مقدار صفر در غیر اینصورت یک مقدار مثبت برمیگرداند.

```
char *s1= "ali reza";
char *s2="ali";
char s1[10]= "ali reza";
char s2[10]="ali";
cout << strncmp(s1, s2,3) << endl;
```

```
0
```

```
char *s1= "ali reza";
char *s2="ali";
char s1[10]= "ali reza";
char s2[10]="ali";
cout << strncmp(s1, s2,4) << endl;
```

```
32
```

تابع strcat(s1, s2)

دو رشتهٔ s1 و s2 را بعنوان آرگومان گرفته رشتهٔ s2 را به انتهای رشتهٔ s1 اضافه می‌نماید. کرکتر اول رشتهٔ s2 روی کرکتر پایانی '\0' رشتهٔ s1 نوشته می‌شود ونهایتاً رشتهٔ s1 را برمیگرداند.

```
char s1[20]= "ali ";
char *s2="reza";
char s2[20]="reza";
cout << strcat(s1, s2) << endl;
```

ali reza

تابع strncat(s1, s2,n)

دو رشته s1 و s2 ومقدار صحیح و مثبت n را بعنوان آرگومان گرفته، حداکثر n کرکتر از رشتهٔ s2 را در انتهای رشتهٔ s1 کپی می‌نماید. اولین کرکتر رشته s2 روی کرکتر پایانی '\0' رشتهٔ s1 می‌نویسد ونهایتاً مقدار رشتهٔ s1 را برمیگرداند.

```
char s1[20]= "ali ";
char *s2="reza";
char s2[20]="reza";
cout << strncat(s1, s2,2) << endl;
```

ali re

تابع strlen(s)

رشتهٔ s را بعنوان آرگومان گرفته طول رشته را مشخص می‌نماید.

```
char *s1= "ali";
char s1[10]= "ali";
cout << strlen(s1);
```

3

```
char *s1= "ali ";
char s1[10]= "ali ";
cout << strlen(s1);
```

4

تابع strcpy(s1,s2)

دو رشتهٔ s1 و s2 را بعنوان آرگومان گرفته رشتهٔ s2 را در رشتهٔ s1 کپی می‌نماید و نهایتاً مقدار رشتهٔ s1 را بر می‌گرداند.

```
char s1[20]= "ali ";
char *s2="reza";
char s2[10]="reza";
cout << strcpy(s1, s2) << endl;
```

reza

```
char s1[20];
char *s2="ali";
char s2[10]="ali";
cout << strcpy(s1, s2) << endl;
```

ali

تابع strncpy(s1, s2,n)

دو رشتهٔ s2 , s1 و مقدار صحیح و مثبت n را بعنوان آرگومان گرفته، حداکثر n کرکتر را از رشتهٔ s2 در رشتهٔ s1 کپی نموده، نهایتاً مقدار رشتهٔ s1 را برمیگرداند.

```
char s1[20]= "ali ";
char *s2="reza";
char s2[10]="reza";
cout << strncpy(s1, s2,3) << endl;
```

rez

```
char s1[20];
char *s2="amir ali";
char s2[10]="amir ali";
cout << strncpy(s1, s2,4) << endl;
```

amir

نکته مهم

برای استفاده از توابع مربوط به رشته‌ها بایستی حتماً در ابتدا برنامه #include <string.h> را قرار دهیم.

برنامه ذیل پنج اسم را بصورت 5 رشته در نظر گرفته آن ها را بترتیب حروف الفباء مرتب نموده نمایش می‌دهد.

```
using namespace std;
#include<iostream>
#include<conio.h>
#include<string.h>
void sort(char name[][10])
  {char  t[10];
  for(int i=0; i<4; ++i)
    for(int j=i+1; j<5; ++j)
      if(strcmpi(name[i], name[j])> 0) {// interchange  the two strings
                                        strcpy(t,name[i]);
                                        strcpy(name[i] , name[j]);
                                        strcpy(name[j] , t) ;
                                        }
  }
void main()
{
char name[5][10] = {"sara", "afsaneh", "babak", "saman", "naser" };
sort(name);
// display sorted strings
for(int i=0; i<5; ++i)
        cout << name[i] << endl;
getch();
}
```

مثال :

```
using namespace std;
#include<iostream>
#include<conio.h>
#include<string.h>

int main( )
{
char s1[20]= "happy birthday";
char s2[20]= "happy holidays";
cout << strcmp(s1, s2) << endl;
cout << strncmp(s1, s2, 7) << endl ;
getch();
}
```

```
-1
-6
```

مثال :

```
using namespace std;
#include<iostream>
#include<conio.h>
#include<string.h>

int main( )
{
char s[10] = "sara";
cout << strlen(s);

getch();
}
```

تابع زیر معادل تابع کتابخانه strcmp می باشد.

```
int nikstrcmp(char s[] , char t[] )
{
int i=0;
while (s[i]==t[i] )
if ( s[i++]='\0' )
return 0;
return (s[i]-t[i]);
}
```

برنامه ای بنویسید که رشته ای را از ورودی خوانده، تمام حروف کوچک آن رشته را به حروف بزرگ تبدیل کرده و چاپ می کند.

- توضیح: در این برنامه از تابعی به نام upper() استفاده کنید که در این تابع باید عملیات تبدیل را انجام دهد
- نکته : تفاوت کد اسکی حروف کوچک و بزرگ۳۲ واحد است برای تبدیل ’a‘ به ’A‘ کافی است از ’a‘ به اندازه ۳۲ واحد کم کنید.

برنامه ای بنویسید که رشته ای را که به نقطه ختم می شود را از ورودی خوانده ، کاراکتر های موجود در رشته را به همراه تعداد دفعات تکرار آنها به خروجی ببرد

فصل هشتم

برنامه نویس شی گرا

فهرست مطالب فصل هشتم

۱. تعریف شی گرایی
۲. چند ریختی (polymorphism)
۳. خاصیت ارث بری
۴. پشته (stack)
۵. ایجاد شی
۶. ارث بری
۷. سازنده ها و نابود کننده ها
۸. توابع دوست
۹. کلاس های دوست
۱۰. توابع سازنده پارامتر دار
۱۱. توابع سازنده یک پارامتری
۱۲. عضوهای static
۱۳. کلاسهای تودرتو
۱۴. کلاس های محلی
۱۵. استفاده از object ها بعنوان پارامترهای تابع
۱۶. برگشت اشیاء
۱۷. انتساب اشیاء
۱۸. آرایه اشیاء
۱۹. اشاره گر به اشیاء
۲۰. اشاره گر this
۲۱. توابع مجازی و پلی مرفیسم

تعریف شی گرایی

برنامه نویسی شئ گرا یا oop یک روش جدید برنامه نویسی می‌باشد که در آن از ویژگی ساختیافته همراه با چند ویژگی‌های قوی جدید استفاده می‌شود. زبان برنامه نویسی C++ امکان استفاده از oop را به راحتی فراهم می‌نماید.

نوع داده انتزاعی

- برنامه ها برای حل مسائل دنیای واقعی نوشته می شوند ، مثل نگهداری اطلاعات مربوط به دانشجویان ، نگهداری اطلاعات مشتریان بانک ها.
- گر چه می توان بسیاری از مسئله های دنیای واقعی را با استفاده از انواع اولیه حل کرد ، اما با تعریف انواع جدیدی که بتوان اشیای دنیای واقعی ، مثل دانشجویان را مدلسازی کنند ، حل مسائل راحتر خواهد بود
- در زبان برنامه نویسی شی گرا امکان تعریف انواع جدید وجود دارد
- در زبان برنامه نویسی C++ برای تعریف نوع جدید ، از کلاس ها استفاده می شود

کلاس ها و اشیا

- کلاس ها نوع جدیدی است که برنامه نویس آن را برای حل مسئله های دنیای واقعی تعریف می کند ،
- کلاس ها حاوی داده ها و تعریف عملیات است
- داده های عضو کلاس را فیلد یا صفت می گویند.
- عملیات کلاس ها را تابع یا متد می نامند.

- شی یک نمونه از کلاس است که در تکنیک برنامه نویسی شی گرا یک نهاد زمان اجرا می باشد.

نکته مهم :

تمام زبانهای برنامه نویسی شی‌گرا دارای سه خصوصیت مشترک زیر می‌باشند :

الف: **encapsulation** (محصورسازی)

ب: **polymorphism** (چندریختی)

ج: **inheritance** (ارث بری)

محصورسازی (Encapsulation)

محصور سازی یا بسته بندی به معنای این است که تمام جنبه های یک نهاد در داخل یک کلاس جمع آوری و از سایر نهاد ها تفکیک می شود .بسته بندی ، مانع از این می شود که داده ها یک شی، توسط متد های شی دیگری دستیابی شود.

polymorphism (چند ریختی)

چند ریختی به معنای این است که نهادی مثل متغیر ، تابع ، شی، معانی یا کاربردهای مختلفی داشته باشند .به عنوان مثال توابع همنام نوعی چند ریختی محسوب می شوند.

inheritance (ارث بری)

ارث بری فرآیندی است که بوسیله آن یک شی (object) می‌تواند خاصیت‌های شی دیگری را دارا شود.

تعریف کلاس و اشیادر++c

Class نام کلاس

{

داده ها توابع اختصاصی

Public :

داده ها و عمومی

Private :

داده ها و توابع اختصاصی

Protected:

داده ها و توابع محافظت شده

};اشایی از کلاس

مثال

```
class employee
{
        char name[20];
    public:
        void putname(char *s);
        void getname(char * s);
    private:
        double age;
}em1,em2;

void employee ::putname()
{
    .....
}
void employee :: getname()
{
    .....
}
```

دستیابی به اعضای یک کلاس

برای دستیابی به اعضای یک کلاس از (نقطه) به صورت زیر استفاده می شود.

نام عضو کلاس. نام شی کلاس

em1.name امکان پذیر نیست

Em1.putname()

Em1.age() امکان پذیر نیست

برنامه ای که طول و عرض مستطیلی را از ورودی خوانده ، مساحت و محیط آن را به

```
#include <iostream.h>
#include <conio.h>
class rect
{
      int len, width;

   public:
      void input();
      void calc();
      int prime, area;
};
void rect ::calc()
{
   area = len * width;
   prime= 2 * (len + width);
}
void rect ::input()
{
   cout<<"Enter Length , Width:";
   cin>>len >>width;
}
int main()
{
   rect  r;
   r.input();
   r.calc();
   cout<<"Area  ="<< r.area<<",Perime = "<<r.prime;
   getch();
   return 0;
}
```

برنامه ای با استفاده از کلاس ها بنویسید که عددی را از ورودی گرفته و فاکتوریل آن را محاسبه ونمایش می‌دهد.

پشته (stack)

پشته ساختاری است که دارای خاصیت **last in first out** می‌باشد. پشته فضای پیوسته در حافظه اشغال می‌نماید. عملیاتی کـــه روی پشته انجام می‌شوند عبارتند از :

الف: **push** ، که باعث می‌شود یک عنصر وارد پشته شده.
ب: **pop** ، که باعث می‌شود یک عنصر از پشته خارج گردد.

ایجاد شی (object)

بمنظور ایجاد یک شی بایستی از کلمهٔ رزروشده **class** استفاده نمود. **class** از نظر ظاهر شبیه ساختار یا **struct** می‌باشد. پشته را بعنوان یک **object** می‌توان در نظر گرفت که **data** آن شامل یک آرایه و یک **tos** ، و عملیاتی که روی این **object** انجام می‌شود عبارتست از **push** ، **initialize** ، **pop** کردن پشته.

در اسلاید بعد مثالی از نحوه ایجاد شی آورده شده است.

مثال :

```
#define SIZE 100
// this creates the class stack.
class stack {
        private :
        int stck[SIZE];
        int tos;
public:
            void init( );
            void push(int i);
            int pop( );
                        };
```

بدین معنی است که stck و tos بوسیله توابعی که عضو object نباشند غیر قابل دسترسی هستند. و این یکی از روش‌های محصور سازی اقلام داده‌هاست.

بدین معنی است که بوسیله سایر قطعات برنامه قابل دسترسی می‌باشد.

نکته :

فقط توابع عضو می‌توانند به متغیرهای عضو از نوع **private** دسترسی داشته باشند. بایستی توجه داشت که اگر نوع عضوی مشخص نگردد آن عضو به صورت اتوماتیک **private** می باشد.

Private

نحوه تعریف تابع عضو یک کلاس

```
void stack : : push(int i)
{
      if(tos = = SIZE ) {
         cout << ″stack is full.″;
          return;
                  }
         stck[tos]= i ;
         tos ++ ;
}
```

عملگر: : مشخص می‌نماید که تابع متعلق به کدام **object** می‌باشد. عملگر : : عملگر **scope resolution** نامیده می‌شود.

برنامه کامل stack :

```
#include   <iostream.h>
#define   SIZE  100
// this creates the class stack.
class stack {
        int stck[SIZE];
        int tos;
public:
void init(int i);
      int pop( );
      void push(int i);
                  };
void stack : : init( )
{
            tos = 0 ;
}
void stack : : push(int i)
{
         if(tos = = size) {
        cout << ″stack is full.″ ;
         return ;
                }
stck[tos] = i ;
tos ++ ;
}
```

```
int stack : : pop( )
{
      if(tos = = 0) {
       cout << ″stack empty.″ ;
       return 0 ;    }
       tos - - ;
       return stck[tos];
 }

int main( )
{
stack st1, st2; // create two objects
st1. init( );
st2.init( );
st1.push(1);
st2.push(2);
st1.push(3);
st2.push(4);
cout << st1.pop( ) << endl;
cout  << st1.pop( ) << endl;
cout << st2. pop( ) << endl;
cout << st2. pop( ) << endl;
return 0; }
```

سازنده‌ها و نابودکننده‌ها (constructors and destructors)

Initialization یا مقدار اولیه دادن بصورت اتوماتیک از طریق تابعی انجام می‌شود بنام تابع **constructor** یا تابع سازنده. تابع سازنده تابع مخصوصی است که عضوی از کلاس بوده و همنام با کلاس می‌باشد.

سازنده‌ها و نابودکننده‌ها (constructors and destructors)

تابع نابود کننده یا **destructor** ، عکس عمل تابع سازنده را انجام می‌دهد. وقتی که شی‌ای از بین می‌رود بصورت اتوماتیک تابع نابود کننده آن فراخوانی می‌گردد.

سازنده‌ها (constructors)

```
class myclass
{
        int a,b;
    public:
        myclass();
        void show();
};
myclass ::myclass()
{
    a=0 ;b=0;
}
void myclass::show()
{
    cout<<"a= " <<a;
    cout<<"b= "<<b;

}

myclass ob1;
ob1.show();
```

سازنده‌ها با پارامتر (constructors)

```
class myclass
{
      int a,b;
   public:
      myclass(int i, int j);
      void show();
};
myclass ::myclass(int i, int j)
{
   a=i ;b=j;
}
void myclass::show()
{
   cout<<"a= " <<a;
   cout<<"b= "<<b;

}

myclass ob1(10,15);
myclass ob2=myclass(20,30);

ob1.show();
ob2.show();
```

مثال از سازنده با یک پارامتر

```
#include <iostream.h>
#include <conio.h>
class store
{
        int a;
    public:
        store(int j){a=j;}
        int getA(){return a;}

};
int main()
{
    store ob= 99;
    cout<<"j = "<<ob.getA();
    getch() ;
    return 0;
}
```

توابع دوست کلاس

همانطورکه گفته شد تابعی که عضو کلاس نباشد ، نمی تواند به اعضای اختصاصی آن کلاس دستیابی داشته باشند

برای اعلان تابع دوست ، باید الگوی آن را در داخل کلاس قرار دهید و کلمه کلیدی Friend را قبل از آن ذکر کنید.

برنامه ای که با استفاده از تابع دوست دوعدد را با هم جمع و تفریق می کند

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

class AClass {
private:
   int a,b;
public:
   void test() {a=10;b=20;}
// Declare the two functions friends.
friend int add(AClass input);
friend int minus(AClass input);
};

// Function one
int add(AClass input) {return int(input.a + input.b);}

// Function two
int minus(AClass input) {return int(input.b - input.a);}

void main(void){
AClass output;
// Initialize output
output.test();
cout << add(output) << endl;
cout << minus(output) << endl;
getch();
}
```

مثال :تابع دوست دو کلاس

```
#include <iostream.h>
#include <conio.h>
class B;
class A
{
    private:
      int a;
    public:
      A(){a = 100;}
      friend void add(A,B);
};

class B{
    private:
      int b;
    public:
      B(){b = 100;}
      friend void add(A,B);
};

void add(A Aobj, B Bobj)
{
    cout << (Aobj.a + Bobj.b);
}

void main()
{
    A A1;
    B B1;
    add(A1,B1);
    getch();
}
```

مثال :کلاس های دوست

کلاس های را می توان دوست کلاس های دیگر معرفی کرد .دراین حالت ، کلاس دوست و تمام توابع عضو آن به اعضای اختصاصی کلاس دیگر دسترسی دارند

برنامه ای که با استفاده از کلاس دوست ، بین دو عدد کوچکترین را پیدا می کند.

```
#include <iostream.h>
#include <conio.h>
class twovalues
{
      int a,b;
   public:
      twovalues(int i, int j){a=i;b=j;}
      friend class min;
};
class min
{
   public:
      int findmin(twovalues x);
};
int  min::findmin(twovalues x)
{
   return x.a<x.b?x.a:x.b;
}
int main()
{
   min m;
   twovalues ob(10,20);
   cout<<"min of 10 , 20is:"<<m.findmin(ob);
   getch();
   return 0;
}
```

اعضای کلاس با ویژگی static

در بعضی از مواقع ، فقط یک کپی از یک متغیر باید بین تمام اشیای یک کلاس مشترک باشد . کلمه کلیدی static برای این منظور به کار می رود.

وقتی کلمه کلیدی static را با اعضای داده ای کلاس به کار می بریم ، به کامپایلر می گوییم که فقط یک کپی از آن متغیر وجود خواهد داشت و تمام اشیای آن کلاس ، آن متغیر را به اشتراک می گذارند.

مثال :اعضای کلاس با ویژگی static

برنامه ای که کاربرد و اثر عضو داده ای staticرا نشان می دهد.

```
#include <iostream.h>
#include <conio.h>
class shared
{
      static int a;
      int b;
   public:
      void set(int i,int j){a=i;b=j;}
      void show();
};
int shared::a;
void shared::show()
{
cout<<"this is static a:"<<a<<endl;
cout<<"this is non-static b:"<<b<<endl;
}
int main()
{
   shared x,y;
   x.set(1,1);
   x.show();

   y.set(2,2);
   y.show();

   x.show();

   getch();
   return 0;
}
```

انتساب اشیاء (object assignment)

در صورتیکه دو تا **object** از یک نوع باشند
می‌توان یک **object** را بدیگری انتساب نمود.

در اسلاید بعد مثالی آورده شده است.

مثال :

```
#include   <iostream.h>
#include  <conio.h>
class myclass{
        int i;
        public:
        void set_i(int n) {i=n;}
        int get_i( ) {return i;}
          };
int main( )
{
myclass ob1, ob2;
obl.set_i(126);
ob2= ob1; // assign data from ob1 to ob2
clrscr( );
cout << ob2.get_i( );
retrun 0 ;
            }
```

مثال :آرایه ای از اشیا

```
#include <iostream.h>
#include <conio.h>
class display {
   int number;
public:
   display(int n) {number=n;}
   int show() { cout <<number << endl; }
};
int main() {
   display myarray[4] = {1,2,3,4,};
   for (int i = 0; i < 4 ;i++ )
      myarray[i].show();
   getch();
   return 0;
}
```

اشاره گر به کلاس Pointers to Classes

- اشاره گرها همانطور که می توانند به متغیرها اشاره کنند، می توانند به اشیاء یک کلاس نیز اشاره کنند و چون شی نمونه ای از کلاس است پس به کلاس اشاره می کنند .
- در حالت ساده برای دسترسی به اعضای کلاس توسط شی تعریفی، از نقطه (.) استفاده کردیم. اما در اشاره گر به شی باید از ترکیب ۲ کاراکتر خط و بزرگتر (>-) استفاده کنیم. با هم مثالی کامل از اشاره گر به شی یا همان کلاس را بررسی می کنیم :

مثال :اشاره گر به کلاس

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

class CRectangle
{
    int width, height;
  public:
    void set_values (int, int);
    int area (void) {return (width * height);}
};

void CRectangle::set_values (int a, int b)
{
  width = a;
  height = b;
}

int main()
{
  CRectangle a, *b, *c;
  CRectangle * d = new CRectangle[2];
  b= new CRectangle;
  c= &a;
  a.set_values (1,2);
  b->set_values (3,4);
  d->set_values (5,6);
  d[1].set_values (7,8);
  cout << "a area: " << a.area() << endl;
  cout << "*b area: " << b->area() << endl;
  cout << "*c area: " << c->area() << endl;
  cout << "d[0] area: " << d[0].area() << endl;
  cout << "d[1] area: " << d[1].area() << endl;
  delete[] d;
  delete b;
  return 0;
}
```

برنامه ای که نام ، شماره دانشحویی و معدل تعداد ۱۵ دانشجو را از ورودی خوانده در آرایه ای از اشیا قرار داده و سپس دانشجویی که بیشترین معدل را دارد نمایش دهد.

ارث بری

ارث بری فرآیندی است که بوسیله آن یک شی (**object**) می‌تواند خاصیت‌های شی دیگری را دارا شود.

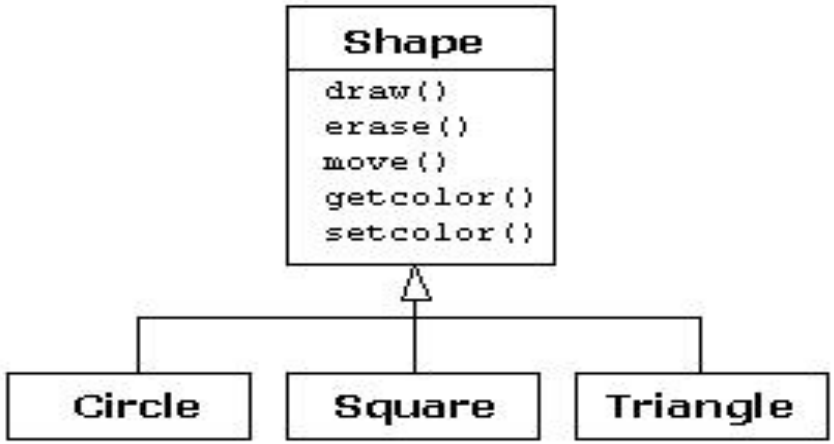
کلاس های پایه و مشتق شده

- چند کلاس ممکن است خصوصیات و رفتارهای مشترکی داشته باشند اما هریک شامل خواص و توابع دیگری هم باشد. وراثت اجازه می دهد یک کلاس عمومی تعریف شود که اشیا درخصوصیات آن مشترک هستند و این کلاس می تواند توسط سایر کلاس ها ارث برده شود و خواص جدیدی به آن اضافه شود بدون اینکه تاثیری روی کلاس عمومی داشته باشد.
- وراثت شباهت بین دو کلاس را با استفاده از مفاهیم کلاس پایه baseو کلاس مشتق شده derivedبیان می کند. کلاسی که از آن ارث بری می شود کلاس پایه یا مبنا و کلاس وارث که خصوصیات کلاس پایه را به ارث می برد را کلاس مشتق شده می نامند. کلاس پایه شامل کلیه خواص و رفتارهائی است که بین کلاس های مشتق شده مشترک است.

مثال:

- مثال. کلاس پایه shapeرا درنظر بگیرید که دارای خاصیت های اندازه، رنگ و موقعیت است. هر شکل می تواند رسم شود، پاک شود، حرکت کند و رنگ شود. هر کدام از اشکال دارای خواص و رفتارهای اضافه تری هستند. برای یک شکل معین بعضی رفتارها ممکن است متفاوت باشد مثلا محاسبه مساحت.
- نکته. یک کلاس متشق شده به نوبه خود می تواند کلاس پایه برای سایر کلاس ها باشد.

نکته. اگر کلاس پایه تغییر کند کلاس مشتق شده نیز تحت تاثیر این تغییرات قرار می گیرد.



تعریف کلاس مشتق شده

- فرم کلی تعریف یک کلاس مشتق شده به صورت زیر است:

```
class derived : access base
{
  //members of new class;
}
```

derivedنام کلاس جدید است که از کلاس پایه baseمشتق شده است. قسمت accessاختیاری است ولی می تواند public، privateیا protectedباشد و برای تعیین مجوز دسترسی اعضای کلاس پایه در کلاس جدید بکار می رود. اگر مجوز دسترسی ذکر نشود به این معنی است که کلیه اعضای عمومی کلاس پایه در کلاس مشتق شده به صورت خصوصی خواهند بود.

مثال. کلاس جدید Derivedاز کلاس Baseمشتق شده است. در برنامه اصلی تابع changeاز کلاس Derivedفراخوانی شده که خود دو تابع setو readاز کلاس Baseرا صدا می زند.

```
#include <iostream.h>
#include <conio.h>
class Base
{
   int i;
public:
   int read() { return i; }
   void set(int ii) { i = ii; }
   Base() { i=0; }
};
class Derived : public Base
{
   int j;
public:
   Derived() { j=0; }
   void change(int x) { set(x); cout << read(); }
};
int main() {
   Derived d;
   d.change(10);
   getch();
   return 0;
}
```

مثال:

```
#include <iostream.h>>
#include <conio.h>
// base class
class Shape
{
   public:
      void setWidth(int w){width = w;}
      void setHeight(int h){height = h;}
   protected:
      int width;
      int height;
};
// Derived class
class Rectangle: public Shape
{
   public:
      int getArea(){return (width * height);}
};
int main()
{
   Rectangle Rect;
   Rect.setWidth(5);
   Rect.setHeight(7);

   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() << endl;
   getch();
   return 0;
}
```

پروژه

کلاسی به نام tictac تعریف کنید که با آن بتوانید دوزبازی انجام دهید این کلاس حاوی یک آرایه دوبعدی ۳* ۳ از نوع صحیح است .سازنده کلاس باید عناصر آرایه را برابر صفر قرار دهد. دو بازیکن ، بازی می کنند . بازیکن اول عدد ۱ را در قرار می دهد و بازیکن دوم عدد ۲ را در مربعی قرار می دهد . حرکت ها باید در خانه های خالی انجام شود .پس از هر حرکت کنترل می کنید که آیا کسی برنده شده است یا خیر . برنامه باید نوبت را رعایت کند مشخص کند که کدام بازیکن باید اول شروع کند.

keywords and alternative tokens.

asm	enum	protected	typedef
auto	explicit	public	typeid
bool	extern	register	typename
break	false	reinterpret_cast	union
case	float	return	unsigned
catch	for	short	using
char	friend	signed	virtual
class	goto	sizeof	void
const	if	static	volatile
const_cast	inline	static_cast	wchar_t
continue	int	struct	while
default	long	switch	xor
delete	mutable	template	xor_eq
do	namespace	this	or_eq
double	new	throw	not
dynamic_cast	operator	true	bitand
else	private	try	and_eq
And -- or	bitor	not_eq	compl

پایان