یا در مجموعهٔ

$$\{0, 1, 2, \ldots, M\}$$

می‌شود. برای مثال در حساب با باقیماندهٔ 12 که گاهی اوقات حساب "ساعتی" نیز نامیده می‌شود داریم:

$$6+9\equiv 3, \quad 7\times 5\equiv 11, \quad 1-5\equiv 8, \quad 2+10\equiv 0\equiv 12$$

(استفاده از 0 یا M بستگی به کاربرد آن در محاسبه دارد.)

تابعهای مقدار صحیح و قدرمطلق

فرض کنید x یک عدد اعشاری دلخواه باشد. **مقدار صحیح** x که به صورت **INT(x)** نوشته می‌شود عدد اعشاری x را با حذف (قطع) **قسمت کسری** آن به یک عدد صحیح تبدیل می‌کند. بنابراین

$$\mathrm{INT}(3.14)=3, \quad \mathrm{INT}(\sqrt{5})=2, \quad \mathrm{INT}(-8.5)=-8, \quad \mathrm{INT}(7)=7$$

ملاحظه می‌کنید بسته به اینکه x مثبت یا منفی باشد $\mathrm{INT}(x)=\lfloor x\rfloor$ یا $\mathrm{INT}(x)=\lceil x\rceil$.

قدر مطلق عدد اعشاری x که به صورت **ABS(x)** یا $|x|$ نوشته می‌شود بنابه تعریف برابر x یا $-x$ است. از این‌رو $\mathrm{ABS}(0)=0$ و به‌ازای $x\neq 0$، $\mathrm{ABS}(x)=x$ یا $\mathrm{ABS}(x)=-x$، بسته به این که x مثبت یا منفی باشد. بنابراین

$$|-15|=15, \quad |7|=7, \quad |-3.33|=3.33, \quad |4.44|=4.44, \quad |-0.075|=0.075$$

توجه دارید که $|-x|=|x|$ و به‌ازای $x\neq 0$، $|x|$ مثبت است.

نماد جمع؛ مجموعها

دراینجا نماد جمع Σ را معرفی می‌کنیم که علامت حرف یونانی سیگما است. دنبالهٔ a_1، a_2، a_3، .... را درنظر بگیرید. آنگاه مجموع

$$a_1 + a_2 + \ldots + a_n \quad \text{و} \quad a_m + a_{m-1} + \ldots + a_n$$

به ترتیب به صورت زیر نمایش داده می‌شود:

$$\sum_{j=1}^{n} a_j \quad \text{و} \quad \sum_{j=m}^{n} a_j$$

حرف **j** در بسطهای بالا، **اندیس ظاهری** یا **متغیر ظاهری** نامیده می‌شود. از حروف دیگری نظیر **i**، **k**، **s** و **t** نیز به عنوان متغیرهای ظاهری زیاد استفاده می‌شود.

مثال ۲ـ۲

$$\sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

$$\sum_{j=2}^{5} j^2 = 2^2 + 3^2 + 4^2 + 5^2 = 4 + 9 + 16 + 25 = 54$$

$$\sum_{j=1}^{n} j = 1 + 2 + \cdots + n$$

اغلب اوقات، آخرین جمع مثال ۲ در کاربردها ظاهر می‌شود. مقدار آن برابر **n(n+1)/ 2** است. به عبارت دیگر :

$$1+2+3+\cdots+n=\frac{n(n+1)}{2}$$

به این ترتیب برای مثال :

$$1+2+\cdots+50=\frac{50(51)}{2}=1275$$

تابع فاکتوریل

ضرب اعداد صحیح مثبت از **1** تا **n** و خود **n** را با **!n** نمایش داده، آن را **n فاکتوریل** می‌خوانند، به بیان دیگر :

$$n!=1\cdot2\cdot3\cdots(n-2)(n-1)n$$

بنابه قرارداد **0! = 1** تعریف می‌شود.

مثال ۳ـ ۲

(الف) $2!=1\cdot2=2;\quad 3!=1\cdot2\cdot3=6;\quad 4!=1\cdot2\cdot3\cdot4=24$

(ب) به ازای **n>1** ، داریم $n!=n.(n-1)!$. از این رو

$$5!=5\cdot4!=5\cdot24=120;\quad 6!=6\cdot5!=6\cdot120=720$$

جایگشتها

یک جایگشت، یک مجموعهٔ **n** عنصری از عناصر است که در آن، عناصر با یک نظم خاص مرتب شده باشند. برای مثال، جایگشتهای یک مجموعه با عنصرهای **a** ، **b** ، **c** به صورت زیر است :

$$abc,\quad acb,\quad bac,\quad bca,\quad cab,\quad cba$$

می‌توان ثابت کرد که : **یک مجموعهٔ n عنصری n! جایگشت دارد.** بنابراین یک مجموعهٔ ۴ عنصری ۲۴ = !۴ جایگشت و یک مجموعهٔ ۵ عنصری ۱۲۰ = !۵ جایگشت دارد و الی آخر.

توان‌رسانی و لگاریتم‌گیری

یادآوری می‌کنیم که تعریف‌های زیر برای توانهای صحیح **a** (که در آن **m** عدد صحیح مثبت است) همواره صادق است:

$$a^0=1,\qquad a^{-m}=\frac{1}{a^m}\qquad\qquad a^m=a\cdot a\cdots a\quad (\text{m بار})$$

عمل توان‌رسانی تا جایی که شامل تمام اعداد گویا می‌شود قابل تعمیم است. بنابه تعریف برای هر عدد گویای **m / n** داریم :

$$a^{m/n}=\sqrt[n]{a^m}=(\sqrt[n]{a})^m$$

مثلاً

$$2^4 = 16, \quad 2^{-4} = \frac{1}{2^4} = \frac{1}{16}, \quad 125^{2/3} = 5^2 = 25$$

درحقیقت، عمل توان رسانی تا جایی که شامل تمام اعداد حقیقی می‌شود نیز قابل تعمیم است. بنابه تعریف برای هر عدد حقیقی x،

$$a^x = \lim_{r \to x} a^r \qquad \text{که در آن r یک عدد گویا است}$$

برطبق آن، به‌ازای تمام اعداد حقیقی، تابع نمایی $f(x) = a^x$ تعریف می‌شود.

لگاریتم با توان به صورت زیر در ارتباط است. فرض کنید b یک عدد صحیح مثبت باشد. لگاریتم عدد مثبت و دلخواه x در مبنا یا پایهٔ b که به صورت زیر نوشته می‌شود.

$$\log_b x$$

برابر توانی است که b بایستی به آن توان برسد تا x بدست آید، به بیان دیگر

$$y = \log_b x \quad \text{و} \quad b^y = x$$

دو رابطهٔ معادل هستند. بنابراین

$\log_2 8 = 3$ چون $2^3 = 8$ ؛ $\log_{10} 100 = 2$ چون $10^2 = 100$

$\log_2 64 = 6$ چون $2^6 = 64$ ؛ $\log_{10} 0.001 = -3$ چون $10^{-3} = 0.001$

علاوه بر این برای هر مبنای دلخواه b :

$\log_b 1 = 0$ چون $b^0 = 1$

$\log_b b = 1$ چون $b^1 = b$

لگاریتم اعداد منفی و لگاریتم صفر تعریف نشده است.

می‌توان تابعهای نمایی و لگاریتمی

$$f(x) = b^x \quad \text{و} \quad g(x) = \log_b x$$

را به عنوان توابع معکوس یکدیگر، مورد توجه قرار داد. به‌موجب آن، نمودارهای این دو تابع، معکوس یکدیگر هستند. مسأله ۱۵- ۲ را ببینید.

اغلب مقدار لگاریتم اعداد به صورت مقادیر تقریبی بیان می‌شوند. برای مثال با استفاده از جدول لگاریتمی، ماشین حساب یا کامپیوتر

$$\log_{10} 300 = 2.4771 \quad \text{و} \quad \log_e 40 = 3.6889$$

را به عنوان جوابهای تقریبی به‌دست می‌آوریم. در اینجا e = 2.71828... لگاریتم‌هایی که در کارِ ما از اهمیت زیادی برخوردارند عبارتند از: لگاریتم در مبنای 10 که **لگاریتم معمولی** نام دارد، لگاریتم در مبنای e که **لگاریتم طبیعی** نام دارد و لگاریتم در مبنای 2 که **لگاریتم دودویی** نام دارد.

در بعضی از کتابها به جای $\log_e^x$ می‌نویسند **lnx** و به جای $\log_2^x$ می‌نویسند **lgx** یا **logx**.
این کتاب که دربارهٔ درس ساختمان داده‌هاست اساساً با لگاریتم‌های دودویی اعداد و عبارتها سر و کار دارد.

بنابراین **منظور ما از اصطلاح logx در این کتاب $\log_2^x$ است مگر آن که خلاف آن بیان شود.**
اغلب ما فقط به مقادیر کف و سقف لگاریتم دودویی نیازمندیم. با ملاحظهٔ توانهای 2، می‌توانیم به این مقادیر دست یابیم. برای مثال :

$\lfloor \log_2 100 \rfloor = 6$ چون $2^6 = 64$ $2^7 = 128$

$\lceil \log_2 1000 \rceil = 9$ چون $2^8 = 512$ و $2^9 = 1024$

و الی آخر.

۳ـ۲ نمایش الگوریتمی

به بیان شهودی، یک الگوریتم یک لیست متناهی و مرحله به مرحله از دستورات خوش تعریف است که برای حل یک مسأله خاص درنظر گرفته می‌شود. تعریف رسمی الگوریتم که از مفهوم **ماشین تیورینگ** بهره می‌گیرد بسیار پیچیده و در محدودهٔ مطالب این درس قرار ندارد. این بخش فُرمَتی را توضیح می‌دهد که برای نمایش الگوریتم‌ها در سراسر کتاب از آن استفاده می‌شود بهتر دیدیم این نمایش الگوریتمی را با ارائه چند مثال توضیح دهیم.

مثال ۴ـ۲

آرایهٔ **DATA** با مقادیر عددی در حافظه ذخیره شده است. می‌خواهیم **LOC** مکان و **MAX** مقدار بزرگترین عنصرِ آرایهٔ **DATA** را پیدا کنیم. دربارهٔ **DATA** هیچ اطلاع دیگری نداریم. یک راه برای حل این مسأله به قرار زیر است :

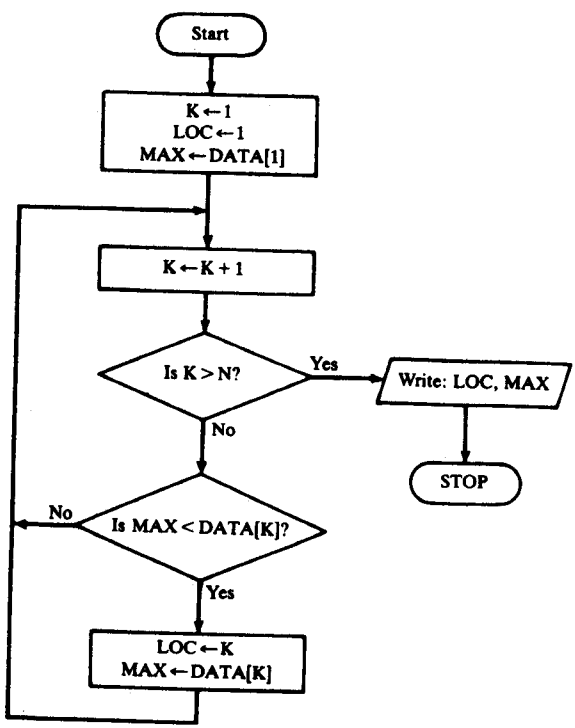
در آغاز، کار را با **LOC = 1** و **MAX = DATA[1]** شروع می‌کنیم. آنگاه **MAX** را با هر یک از عناصر بعدی **DATA** یعنی با **DATA[K]** مقایسه می‌کنیم. اگر **DATA[K]** بزرگتر از **MAX** باشد آنگاه **LOC** و **MAX** را تازه می‌کنیم طوری که **LOC = K** و **MAX = DATA[K]** شود. آخرین مقداری که در **LOC** و **MAX** قرار می‌گیرد مکان و مقدار بزرگترین عنصر **DATA** است.

نمایش رسمی این الگوریتم به صورت زیر است. فلوچارت آن در شکل ۲ـ۲ رسم شده است.

Algorithm 2.1: (Largest Element in Array) A nonempty array DATA with N numerical values is given. This algorithm finds the location LOC and the value MAX of the largest element of DATA. The variable K is used as a counter.

Step 1. [Initialize.] Set K := 1, LOC := 1 and MAX := DATA[1].
Step 2. [Increment counter.] Set K := K + 1.
Step 3. [Test counter.] If K > N, then:
Write: LOC, MAX, and Exit.
Step 4. [Compare and update.] If MAX < DATA[K], then:
Set LOC := K and MAX := DATA[K].
Step 5. [Repeat loop.] Go to Step 2.

فُرمت بالا برای نمایش رسمی یک الگوریتم از دو قسمت تشکیل شده است. در قسمت اول، متنی قرار دارد که اطلاعاتی راجع به هدف الگوریتم به دست می‌دهد و متغیرهایی را که در الگوریتم از آنها استفاده می‌شود معرفی می‌کند و لیستِ داده‌های ورودی را ارائه می‌دهد. قسمت دومِ این الگوریتم شامل لیست مرحله‌هایی است که بایستی به ترتیب اجرا شوند.



شکل ۲ـ ۲ فلوچارت الگوریتم ۱ـ ۲

در زیر چند قرارداد را که ما در ارائه الگوریتم‌ها از آن استفاده می‌کنیم به صورت خلاصه‌شده، می‌آوریم. برخی از دستورهای کنترلی در بخش بعد به‌کار می‌آیند.

شمارهٔ شناسایی

به هر الگوریتم به صورت زیر یک شمارهٔ شناسایی نسبت داده شده است: که منظور از الگوریتم 3–4 الگوریتم سوم از فصل ۴ و منظور از الگوریتم P 5–3 الگوریتم مسأله ۳ـ۵ از فصل ۵ است. توجه دارید که حرف P مبین آن است که این الگوریتم در یک مسأله Problem عنوان شده است.

مرحله‌ها، کنترل، خروج از الگوریتم Exit

مراحل یک الگوریتم یکی پس از دیگری اجرا می‌شوند. این مراحل با مرحلهٔ 1 شروع می‌شود، مگر آن که خلاف آن گفته شود. دستور "برو به n، یا، Go To n" کنترل کار را به دستور مرحلهٔ n از الگوریتم منتقل می‌کند. مثلاً در الگوریتم 1–2 مرحلهٔ 5، کنترل اجرا را به مرحلهٔ 2 منتقل می‌کند. به بیان کلی‌تر، دستورهای GO To با بکارگیری تعدادی از دستورهای کنترلی که در بخش بعد مطرح می‌شوند عملاً و در ظاهر حذف می‌شوند. اگر در یک مرحله، از چند دستور استفاده شود نظیر

Set K := 1, LOC := 1 و MAX := DATA[1].

آنگاه این دستورها از چپ به راست اجرا می‌شوند.

الگوریتم زمانی به پایان می‌رسد که دستور خروج از الگوریتم

Exit

مشاهده شود به عبارت دیگر دستور Exit به معنی پایان الگوریتم است. این دستور مشابه دستور STOP در FORTRAN و فلوچارت‌ها است.

توضیحات Comments

هر مرحله ممکن است شامل یک توضیح در داخل کروشه باشد که هدف اصلی آن مرحله را بیان می‌کند. توضیح، معمولاً در آغاز یا پایان یک مرحله داده می‌شود.

اسامی متغیرها

در نام متغیرها نظیر MAX و DATA از حروف بزرگ استفاده می‌کنیم. از متغیرهای تک حرفی که در الگوریتم با حروف بزرگ نوشته شده‌اند مانند K و N، به عنوان شمارنده یا اندیس استفاده می‌کنیم حتی

اگر برای این متغیرها در تجزیه و تحلیل‌ها یا عبارات ریاضی از حروف کوچک نظیر **k** و **n** استفاده شود. برای یادآوری، به بحث مربوط به نمادهایی با حروف کج یا حروف کوچکِ بخش ۳- ۱ از فصل ۱، تحت عنوان آرایه‌ها مراجعه کنید.

دستورهای جایگزینی

برای دستورهای جایگزینی همانگونه که در زبان **PASCAL** رایج است از نماد کولن و تساوی **:=** استفاده می‌شود. برای مثال :

Max := DATA[1]

مقدار **DATA[1]** را در متغیر **MAX** جایگزین می‌کند. بعضی از کتابها برای عمل جایگزینی از نماد پیکان به چپ ← یا علامت تساوی = استفاده می‌کنند.

ورودی و خروجی

داده‌ها به کمک دستور **Read** به صورت زیر از ورودی دریافت شده، در متغیرها جایگزین می‌شوند :

Read : نام چند متغیر

به‌طور مشابه، پیغامهایی که در داخل علامت نقل‌قول یا کوتیشن قرار دارند به همراه داده‌های داخل متغیرها، به کمک دستور **Write** یا **Print** به صورت زیر در خروجی چاپ می‌شوند :

Write : نام چند متغیر یا / و یا پیغام

زیربرنامه

اصطلاح "زیربرنامه" برای قطعه الگوریتم مستقلی به‌کار می‌رود که بخشی از یک مسأله را حل می‌کند. استفاده از واژهٔ "زیربرنامه" یا "قطعه برنامه" به جای "الگوریتم" برای یک مسألهٔ داده شده، تنها یک موضوع سلیقه‌ای است. به بیان کلی‌تر، واژهٔ الگوریتم برای حل یک مسأله با نگرش کلی به‌کار می‌رود اما اصطلاح "زیربرنامه" برای توصیف و حل یک بخش از الگوریتم یا **زیرالگوریتم** مورد استفاده قرار می‌گیرد که در بخش ۶- ۲ به تفصیل مورد بحث و بررسی قرار خواهد گرفت.

۴- ۲ دستورهای کنترلی

الگوریتم‌ها و برنامه‌های کامپیوتریِ معادل آنها زمانی بهتر و ساده‌تر درک می‌شوند که در آنها اساساً از چند زیربرنامه و سه نوع **منطق** یا **جریان کنترلی** به شرح زیر استفاده شود :

(۱) **منطق توالی** یا جریان متوالی اجرای دستورات

(۲) **منطق انتخاب** یا دستورات شرطی

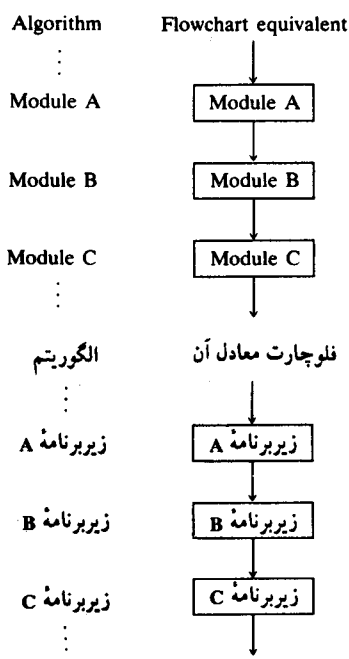
(۳) **منطق تکرار** یا جریان تکراری

سه منطق بالا در زیر، مورد بحث و بررسی قرار می‌گیرد و در هر حالت **فلوچارت** مربوط به آن رسم می‌شود.

منطق توالی (جریان متوالی اجرای دستورات)

منطق توالی قبلاً مورد بررسی قرار گرفت بجز در مواردی که بعضی از دستورات، اجرای پشت سرهم یا متوالی دستورات را برهم می‌زنند. تمام دستورات برنامه‌ها و زیربرنامه‌ها به صورت متوالی و پشت‌سرهم اجرا می‌شوند. توالی دستورات ممکن است با استفاده از شمارهٔ مراحل به صورت صریح بیان شود و یا براساس ترتیبی که در آن دستورات زیربرنامه‌ها یا برنامه‌ها نوشته می‌شوند به صورت ضمنی باشد. (شکل ۳ـ ۲ را ببینید).

در بیشتر پردازشها، حتی در مسایل پیچیده، عموماً از این الگوی مقدماتی جریان کنترلی، تبعیت می‌شود.



شکل ۳ـ ۲ . منطق توالی

منطق انتخاب (دستورات شرطی)

منطق انتخاب از تعدادی شرط استفاده می‌کند که منتهی به انتخاب یک زیربرنامه از میان چند زیربرنامه می‌شود. دستوراتی که ما برای پیاده‌سازی این منطق به‌کار می‌گیریم، دستورات شرطی یا دستورات IF است. برای روشنی بیشتر مطلب، غالباً در پایان این‌گونه دستورات عبارت

[End of If Structure. If پایان دستور .] یا [پایان دستور IF]

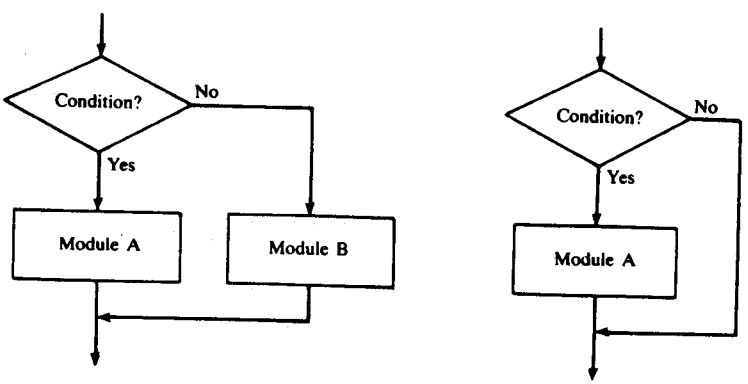
یا با عبارت معادل آن نوشته می‌شود.

دستورات شرطی به سه نوع مختلف تقسیم می‌شوند که هر یک از این دستورات به‌طور جداگانه مورد بحث قرار می‌گیرد.

(۱) حالت یک وضعیتی : این دستور به صورت زیر است :

IF شرط ، THEN :	If condition, then:
[زیربرنامهٔ A]	[Module A]
[پایان دستور IF]	[End of If structure.]

منطق این دستور در فلوچارت شکل ۴ـ ۲ (الف) نشان داده شده است. اگر شرط برقرار باشد، آنگاه زیربرنامهٔ A اجرا می‌شود که خود می‌تواند شامل یک یا چند دستور باشد، درغیر اینصورت زیربرنامهٔ A نادیده گرفته می‌شود و کنترل اجرای دستورات به مرحله بعدی الگوریتم داده می‌شود.



(ب) حالت دو وضعیتی (الف) حالت یک وضعیتی

شکل ۴ـ ۲

(۲) حالت دووضعیتی : این دستور به صورت زیر است :

IF شرط ، THEN :	If condition, then:

[زیربرنامهٔ A]	[Module A]
ELSE :	یا Else:
[زیربرنامهٔ B]	[Module B]
[پایان دستور IF]	[End of If structure.]

منطق این دستور در فلوچارت شکل ۴ـ ۲ (ب) نشان داده شده است. همانگونه که فلوچارت بیان می‌کند اگر شرط برقرار باشد آنگاه زیربرنامهٔ A اجرا می‌شود، درغیر اینصورت اجرا به زیربرنامهٔ B داده می‌شود.

(۳) حالت چندوضعیتی :

IF (۱) شرط, THEN :	If condition(1), then:
[زیربرنامهٔ A_1]	[Module A_1]
ELSE IF (۲) شرط, THEN :	Else if condition(2), then:
[زیربرنامهٔ A_2]	[Module A_2]
⋮	⋮
Else IF (M) شرط , THEN :	Else if condition(M), then:
[زیربرنامهٔ A_M]	[Module A_M]
ELSE :	Else:
[زیربرنامهٔ B]	[Module B]
[پایان دستور IF]	[End of If structure.]

منطق این دستور به‌گونه‌ای است که تنها اجازهٔ اجرای یک زیربرنامه را می‌دهد. به‌ویژه این‌که، یا زیربرنامهٔ پائینِ شرط اول در صورت برقراربودن اجرا می‌شود یا زیربرنامه‌ای که پائین آخرین دستور Else قرار دارد اجرا می‌شود. در عمل به‌ندرت اتفاق می‌افتد که بیش از سه شرطِ متداخل در زیربرنامه وجود داشته باشد.

مثال ۵ـ ۲

ریشه‌های معادلهٔ درجه دوم

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

که در آن $a \neq 0$ ، با استفاده از دستور زیر به دست می‌آید :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

مقدار $D = b^2 - 4ac$ مبین **Discriminant** معادلهٔ درجهٔ دوم نامیده می‌شود. اگر **D** منفی باشد، آنگاه معادله ریشهٔ حقیقی ندارد. اگر $D = 0$ باشد، آنگاه معادله تنها یک ریشهٔ مضاعف (دو ریشهٔ مساوی) $x = -b / 2a$ دارد. اگر **D** مثبت باشد، دستور بالا دو ریشهٔ حقیقی متمایز را به دست می‌دهد. الگوریتم زیرریشه‌های معادلهٔ زیر را پیدا می‌کند.

Algorithm 2.2: (Quadratic Equation) This algorithm inputs the coefficients A, B, C of a quadratic equation and outputs the real solutions, if any.

Step 1. Read: A, B, C.
Step 2. Set $D := B^2 - 4AC$.
Step 3. If $D > 0$, then:
  (*a*) Set $X1 := (-B + \sqrt{D})/2A$ and $X2 := (-B - \sqrt{D})/2A$.
  (*b*) Write: X1, X2.
Else if $D = 0$, then:
  (*a*) Set $X := -B/2A$.
  (*b*) Write: 'UNIQUE SOLUTION', X.
Else:
  Write: 'NO REAL SOLUTIONS'.
[End of If structure.]
Step 4. Exit.

توجه کنید : ملاحظه می‌کنید که در مرحلهٔ **3** از الگوریتم **2.2** سه شرطِ دوبه دو متقابل وجود دارد که اجرای هر یک از آنها بستگی به آن دارد که آیا **D** مثبت است یا صفر یا منفی. در چنین وضعیتی، می‌توان به‌طور متناوب حالتهای مختلف را به صورت زیر ارائه داد :

Step 3. (1) If $D > 0$, then:
  .....
  (2) If $D = 0$, then:
  .....
  (3) If $D < 0$, then:
  .....

بیان دستور **If** مرحلهٔ **3** از الگوریتم **2.2** ، مشابهٔ استفاده از دستور **CASE** در زبان **PASCAL** است.

منطق تکرار (جریان تکراری)

نوع سوم از منطق یا جریان کنترلی به یکی از دو نوع دستور حلقهٔ تکرار زیر مربوط می‌شود. هر یک از این **حلقه‌ها** با یک **دستور Repeat** شروع می‌شود، بدنبال آن یک زیربرنامه قرار می‌گیرد که **بدنهٔ حلقه** نامیده می‌شود. برای روشنی بیشتر مطلب، در پایان این گونه دستورات عبارت

[End of loop.]

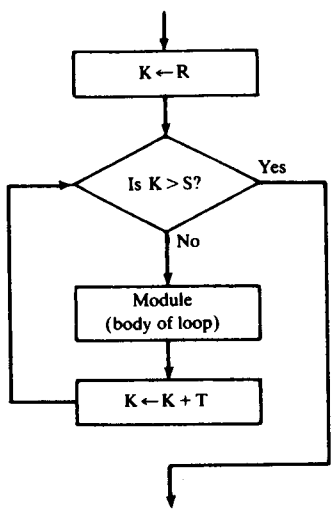
یا عبارت معادل آن نوشته می‌شود.

هر یک از این دو نوع دستور حلقهٔ تکرار بطور جداگانه مورد بحث قرار می‌گیرد.

حلقهٔ تکرار Repeat – For از یک متغیر **شاخص Index** نظیر **K** برای کنترل مراحل تکرار حلقه استفاده می‌کند. این حلقه معمولاً به صورت زیر است :

```
Repeat for K = R to S by T:
    [Module]
[End of loop.]
```

منطق این دستور در فلوچارت شکل ۵ـ ۲ (الف) نشان داده شده است. در اینجا **R مقدار اولیه** و **S مقدار پایانی** یا **مقدار آزمایشی حلقه** و **T نمو حلقه** نام دارد.



شکل ۵ـ ۲ (الف)

دستور Repeat – For

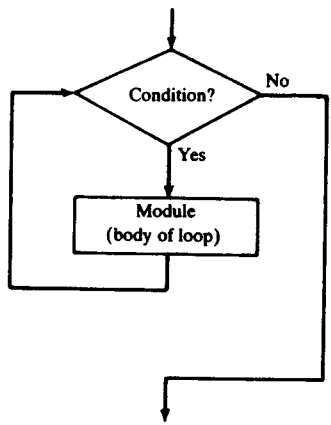
ملاحظه می‌کنید که بدنهٔ این حلقه ابتدا به‌ازای **K = R**، بعد به‌ازای **K = R+T** و بـدنبال آن بـه‌ازای **K = R + 2T** و غیـره اجـرا می‌شـود. حلقه وقتی پایان می‌یابـد که **K>S** شود. این فلوچارت فرض می‌کند که نمو **T** مثبت است، اگر **T** منفی باشد و درنتیجه مقدار **K** دائماً کاهش می‌یابد آنگاه حلقه وقتی پایان می‌یابد که **K < S** باشد.

حلقهٔ Repeat – While از یک شرط برای کنترل مراحل تکرار حلقه استفاده می‌کند. این حلقه معمولاً

به صورت زیر است :

Repeat while condition:	یا	شرط Repeat While :
[Module]		[زیربرنامه]
[End of loop.]		[پایان حلقه.]

منطق این دستور در فلوچارت شکل ۵ـ ۲ (ب) نشان داده شده است.



شکل ۵ـ ۲ (ب)

دستور Repeat ـ While

ملاحظه می‌کنید که اجرای این حلقه تا زمانی ادامه می‌یابد که شرط برقرار باشد. تأکید می‌کنیم که بایستی قبل از این دستور، دستوری وجود داشته باشد تا به متغیرِ کنترل‌کنندهٔ شرطِ حلقه، مقدار اولیه بدهد و برای اینکه اجرای حلقه بتواند درنهایت متوقف شود، بایستی در بدنهٔ حلقه دستوری وجود داشته باشد تا شرط را تغییر دهد.

مثال ۶ـ ۲

الگوریتم 2.1 بجای دستور GO TO ، با استفاده از یک حلقهٔ Repeat – While به صورت زیر نوشته می‌شود.

Algorithm 2.3: (Largest Element in Array) Given a nonempty array DATA with N numerical values, this algorithm finds the location LOC and the value MAX of the largest element of DATA.

```
1. [Initialize.] Set K := 1, LOC := 1 and MAX := DATA[1].
2. Repeat Steps 3 and 4 while K ≤ N:
3.      If MAX < DATA[K], then:
            Set LOC := K and MAX := DATA[K].
        [End of If structure.]
4.      Set K := K + 1.
    [End of Step 2 loop.]
5. Write: LOC, MAX.
6. Exit.
```

الگوریتم 2.3 برخی از ویژگیهای دیگر آن را بیان می‌کند. معمولاً کلمهٔ "مرحله" در الگوریتم حذف می‌شود و سعی می‌شود به جای دستورات **Go To** از دستورات **Repeat** استفاده شود. دستور **Repeat** می‌تواند به صورت صریح مراحلی را نشان دهد که بدنهٔ حلقه را تشکیل می‌دهند. دستور "پایان حلقه **End of Loop**" می‌تواند به صورت صریح مرحله‌ای را نمایش دهد که از آن حلقه شروع می‌شود. زیربرنامه‌های درون دستورات منطقی معمولاً برای سادگی بیشتر در خواندن آنها، به صورت **پله‌ای** یا **تورفته** نوشته می‌شوند. این مطلب تأکیدی دیگر بر نوشتن دستورات معمولی حلقه‌ها به صورت **پله‌ای** در زبانهای برنامه‌نویسی ساختیافته است.

هر نماد یا قرارداد جدید دیگری که در این کتاب آمده باشد یا بروشنی واضح است یا هنگام استفاده از آن، توضیح داده می‌شود.

۵ـ۲ پیچیدگی الگوریتم‌ها

تجزیه و تحلیل الگوریتم‌ها، وظیفهٔ اصلی علم کامپیوتر است. برای مقایسهٔ الگوریتم‌ها، باید معیارهایی برای اندازه‌گیری کارآیی الگوریتم در اختیار داشته باشیم. این موضوع مهم، و قابل توجه در این بخش از کتاب شرح داده می‌شود.

فرض کنید **M** یک الگوریتم و **n** تعداد داده‌های ورودی آن باشد. دو معیار اصلی برای الگوریتم **M**، زمان و مقدار حافظه‌ای است که الگوریتم **M** از آنها استفاده می‌کند. زمان اجرا با شمارش و محاسبهٔ تعداد عملیات کلیدیِ داخل الگوریتم اندازه گرفته می‌شود به عنوان مثال در الگوریتم‌های مرتب‌کردن و جستجوی اطلاعات، تعداد مقایسه‌ها عمل کلیدیِ الگوریتم است. به این علت که عملیات کلیدی چنان تعریف می‌شوند تا زمان مربوط به عملیات دیگر خیلی کوچکتر یا حداکثر متناسب با زمان مربوط به عملیات کلیدی باشد. حافظه، با شمارش و محاسبهٔ ماگزیمم خانهٔ حافظهٔ موردنیاز الگوریتم اندازه‌گیری می‌شود.

پیچیدگی الگوریتم M، تابع $f(n)$ است که زمان اجرا و/ یا حافظهٔ موردنیاز الگوریتم را برحسب n تعداد دادهٔ‌ورودی به دست می‌دهد. اغلب، حافظهٔ موردنیاز یک الگوریتم تنها مضربی از n یا تعداد داده‌های ورودی است. برطبق آن، منظور از اصطلاح "پیچیدگی" زمان اجرای الگوریتم است مگر آن که خلاف آن بیان شود یا بکار آید.

مثال زیر نشان می‌دهد تابع $f(n)$ که زمان اجرای یک الگوریتم را به دست می‌دهد نه تنها به n تعداد داده‌های ورودی بستگی دارد بلکه به نوعِ دادها نیز وابسته است.

مثال ۷ـ ۲

فرض کنید یک داستان کوتاه به نام TEXT دراختیار داریم و بخواهیم در TEXT اولین کلمهٔ سه‌حرفی به نام W را با جستجو پیدا کنیم. اگر W کلمهٔ سه‌حرفی the باشد، آنگاه به‌نظر می‌رسد که W در ابتدای داستان TEXT وجود داشته باشد از این‌رو $f(n)$ یک عدد کوچک خواهد بود. ازطرف دیگر. اگر W کلمهٔ سه حرفی zoo باشد آنگاه ممکن است این کلمه اصلاً در TEXT وجود نداشته باشد، درنتیجه $f(n)$ بسیار بزرگ خواهد بود.

بحث بالا منتهی به این سؤال می‌شود که تابع پیچیدگی $f(n)$ را در حالتهای خاص پیدا کنیم. دو حالتی که معمولاً در نظریهٔ پیچیدگی الگوریتم‌ها موردتوجه قرار می‌گیرد عبارتند از :

(۱) **بدترین حالت** : که ماگزیمم مقدار $f(n)$ برای هر ورودی ممکن است.

(۲) **حالت میانگین** : که مقدار انتظاری $f(n)$ یا امید ریاضی $f(n)$ است.

گاهی اوقات، حداقل مقدار ممکن $f(n)$ نیز مدّنظر قرار می‌گیرد که **بهترین حالت** نامیده می‌شود. تجزیه و تحلیل حالت میانگین فرض می‌کند داده‌های ورودی از یک توزیع احتمال مشخص پیروی می‌کند. در یک چنین فرضی ممکن است تمام جایگشتهای ممکن مجموعه داده‌های ورودی با احتمال مساوی باشد. حالت میانگین از مفهوم زیر نیز در نظریهٔ احتمالات استفاده می‌کند. فرض کنید اعداد n_1، n_2، ... ، n_k به ترتیب با احتمال P_1 ، P_2 ، ... ، P_k ظاهر شوند. آنگاه **امید ریاضی** یا **مقدار میانگین** E از رابطهٔ زیر به دست می‌آید :

$$E = n_1p_1 + n_2p_2 + \cdots + n_kp_k$$

این ایده‌ها در مثال زیر نشان داده شده است.[1]

مثال ۸ـ ۲ : جستجوی خطی

فرض کنید آرایهٔ خطی DATA دارای n عنصر است و دادهٔ مشخص ITEM نیز داده شده است.

می‌خواهیم در صورت وجود **ITEM** در آرایهٔ **DATA** ، **LOC** مکان آن را پیدا کنیم و درصورت عدم وجود، پیغامی نظیر **LOC = 0** بدهیم، تا نشان داده‌شود **ITEM** در **DATA** وجود ندارد. الگوریتم جستجوی خطی، این مسأله را از طریق مقایسهٔ **ITEM** با تک‌تک عناصر آرایهٔ **DATA** حل می‌کند. به‌عبارت دیگر، **ITEM** با **DATA[1]** ، آنگاه با **DATA[2]** و الی آخر مقایسه می‌شود تا **LOC** پیدا شود طوری که **ITEM = DATA[LOC]** باشد. نمایش رسمی این الگوریتم به قرار زیر است :

Algorithm 2.4: (Linear Search) A linear array DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of ITEM in the array DATA or sets LOC = 0.

1. [Initialize] Set K := 1 and LOC := 0.
2. Repeat Steps 3 and 4 while LOC = 0 and K ≤ N.
3. If ITEM = DATA[K], then: Set LOC := K.
4. Set K := K + 1. [Increments counter.]
   [End of Step 2 loop.]
5. [Successful?]
   If LOC = 0, then:
   Write: ITEM is not in the array DATA.
   Else:
   Write: LOC is the location of ITEM.
   [End of If structure.]
6. Exit.

پیچیدگی الگوریتم جستجو با **C** ، تعداد مقایسه‌های انجام‌شده بین **ITEM** و **DATA[K]** به دست می‌آید. **C(*n*)** را در بدترین حالت و حالت میانگین تعیین می‌کنیم.

بدترین حالت

واضح است که بدترین حالت وقتی اتفاق می‌افتد که **ITEM** آخرین عنصر آرایهٔ **DATA** باشد یا اصلاً در آرایه وجود نداشته باشد. در هر دو حالت داریم :

$$C(n) = n$$

بنابراین **C(*n*) = n** پیچیدگی بدترین حالت الگوریتمِ جستجوی خطی است.

حالت میانگین

در اینجا فرض می‌کنیم که **ITEM** در آرایهٔ **DATA** و با احتمال مساوی در هر مکانی از آرایه وجود دارد. به موجب آن، تعداد مقایسه‌ها را می‌توان هر یک از اعداد **1** ، **2** ، **3** ، ... ، **n** دانست و احتمال وقوع هر عدد برابر **P = 1 / n** است. آنگاه

$$C(n) = 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + \cdots + n \cdot \frac{1}{n}$$
$$= (1 + 2 + \cdots + n) \cdot \frac{1}{n}$$
$$= \frac{n(n+1)}{2} \cdot \frac{1}{n} = \frac{n+1}{2}$$

این نتیجه با احساس شهودی ما نیز تطابق دارد که میانگین تعداد مقایسه‌های موردنیاز برای تعیین مکان ITEM تقریباً برابر نصف تعداد عناصر لیست خطی DATA است.

توجه کنید: پیچیدگی حالت میانگینِ یک الگوریتم، معمولاً بسیار پیچیده‌تر از تجزیه و تحلیل پیچیدگی بدترین حالت است. علاوه بر این، توزیع احتمالی که برای حالت میانگین درنظر گرفته می‌شود عملاً در وضعیتهای حقیقی ممکن نیست بکار گرفته شود. به موجب آن، منظور از پیچیدگی یک الگوریتم، تابعی است که زمان اجرای الگوریتم را در بدترین حالت برحسب تعداد داده‌های ورودی به دست می‌دهد، مگر آنکه خلاف آن بیان شود یا بکار آید. این فرض آنقدرها که فکر می‌کنیم قوی نیست، چون پیچیدگی حالت میانگین در مورد بسیاری از الگوریتم‌ها متناسب با بدترین حالت است.

آهنگ (نرخ) رشد، نماد O ی بزرگ (Big O)

فرض کنید **M** یک الگوریتم و **n** تعداد داده‌های ورودی آن باشد. واضح است که پیچیدگی $f(n)$ الگوریتم **M** با زیاد شدن تعداد داده‌های ورودی **n** افزایش می‌یابد. معمولاً ما مایل به‌تعیین آن، آهنگ افزایش $f(n)$ هستیم. این کار معمولاً از مقایسهٔ $f(n)$ با چند تابع استاندارد نظیر

$$\log_2 n, \quad n, \quad n \log_2 n, \quad n^2, \quad n^3, \quad 2^n$$

حاصل می‌شود. آهنگهای رشد این توابع استاندارد، در شکل ۶ـ ۲ نشان داده شده است که مقادیر تقریبی آنها را، به ازای چند مقدار معین **n** به دست می‌دهد.

n \ $g(n)$	$\log n$	n	$n \log n$	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}
1000	10	10^3	10^4	10^6	10^9	10^{300}

شکل ۶ـ ۲ . آهنگ رشد چندتابع استاندارد

ملاحظه می‌شود که تابعها، به ترتیب آهنگ رشدشان ارائه شده‌اند. تابع لگاریتمی $\log_2 n$ رشد نسبتاً کندی دارد، تابع نمایی 2^n رشد نسبتاً تندی دارد و تابع چندجمله‌ای n^c با توجه به مقدار توان c رشد می‌کند. یک راه برای مقایسهٔ تابع $f(n)$ با این تابعهای استاندارد، استفاده از نماد تابعی O است که به صورت زیر تعریف می‌شود:

فرض کنید $f(n)$ و $g(n)$ دو تابعی باشند که بر روی اعداد صحیح مثبت تعریف شده، و دارای این خاصیت هستند که $f(n)$ به‌ازای تمام مقادیر n توسط مضربی از $g(n)$ محدود شده است. به عبارت دیگر، فرض کنید که عدد صحیح مثبت n_0 و عدد مثبت M وجود دارد به‌گونه‌ای که به‌ازای تمام $n>n_0$ داریم:

$$|f(n)| \le M|g(n)|$$

در آن صورت می‌توان نوشت:

$$f(n) = O(g(n))$$

که خوانده می‌شود $f(n)$ از مرتبهٔ $g(n)$ است. برای چندجمله‌ای $p(n)$ که از درجهٔ m است در مسأله ۱۰ـ۲ نشان می‌دهیم $P(n) = O(n^m)$، به عنوان مثال

$$8n^3 - 576n^2 + 832n - 248 = O(n^3)$$

همچنین می‌توان نوشت:

$$f(n) = h(n) + O(g(n)) \quad \text{وقتی که} \quad f(n) - h(n) = O(g(n))$$

(این نماد "O ی بزرگ یا Big O" نام دارد چون $f(n) = O(g(n))$ معنی کاملاً متفاوتی دارد.)

برای این که نشان دهیم این نمادگذاری چقدر مناسب است پیچیدگی چند الگوریتمِ معروف جستجو و مرتب‌کردن را ارائه می‌دهیم:

(الف) جستجوی خطی $O(n)$

(ب) جستجوی دودویی $O(\log n)$

(ج) مرتب کردن حبابی $O(n^2)$

(د) مرتب‌کردن با ادغام $O(n \log n)$

این نتیجه‌ها در فصل ۹ در مبحث مرتب‌کردن و جستجوی اطلاعات به تفصیل مورد بررسی قرار می‌گیرد.

۶ـ۲ زیرالگوریتم‌ها

یک زیرالگوریتم یک قطعه برنامهٔ الگوریتمیِ کامل و به‌طور مستقل تعریف شده، است که به‌وسیلهٔ الگوریتم اصلی یا الگوریتم دیگری مورد استفاده قرار می‌گیرد (احضار یا صدا زده می‌شود). یک

زیرالگوریتم، مقادیر خود را که **آرگومان** نامیده می‌شوند از الگوریتم اصلی یا **فراخوانده** دریافت می‌کند، محاسبات لازم را انجام می‌دهد و آنگاه نتیجه را به **الگوریتم فراخوانده** برمی‌گرداند. زیر الگوریتم به این علت به صورت مستقل تعریف می‌شود تا الگوریتم‌های مختلف بتوانند آن را احضار کنند یا در همان الگوریتم در زمانهای مختلف بتوانند آن را صدا بزنند. رابطهٔ بین یک الگوریتم و یک زیرالگوریتم مشابهٔ رابطهٔ بین برنامه اصلی و یک زیربرنامه در زبانهای برنامه‌نویسی است.

اختلاف اصلی بین فُرمَت یک زیرالگوریتم و یک الگوریتم در آن است که زیرالگوریتم معمولاً دارای عنوانی به صورت زیر است :

$$NAME(PAR_1, PAR_2, \ldots, PAR_K)$$

که در اینجا منظور از **NAME** نام زیرالگوریتم است و هنگامی مورد استفاده قرار می‌گیرد که زیرالگوریتم **احضار** شده باشد و PAR_1 ، PAR_2 ، ... ، PAR_k پارامترهایی هستند که برای انتقال یا جابجایی داده‌ها بین زیرالگوریتم و الگوریتم فراخواننده بکار می‌روند.

تفاوت دیگر، آن است که در زیرالگوریتم به جای دستور **Exit** دستور **Return** داریم. تأکید می‌کنیم کنترلِ اجرا، زمانی به برنامهٔ فراخواننده داده می‌شود که اجرای زیرالگوریتم کامل شده باشد.

زیرالگوریتم‌ها به دودستهٔ اصلی تقسیم می‌شوند : زیرالگوریتم‌های **تابع FUNCTION** و **زیرالگوریتم‌های PROCEDURE** . به کمک چند مثال شباهتها و تفاوتهای این دو نوع زیرالگوریتم را مورد بررسی قرار می‌دهیم. یک تفاوت اساسی این دو زیرالگوریتم آن است که زیرالگوریتمِ تابع تنها یک مقدار را به الگوریتم فراخواننده برمی‌گرداند، درحالی که زیرالگوریتم **PROCEDURE** می‌تواند بیش از یک مقدار برگرداند.

مثال ۹ـ ۲

زیرالگوریتم تابع **MEAN** زیر، میانگین **AVE** سه عدد **A** ، **B** و **C** را پیدا می‌کند.

Function 2.5: MEAN(A, B, C)

1. Set AVE := (A + B + C)/3.
2. Return(AVE).

توجه دارید که **MEAN** نام زیرالگوریتم است و **A** ، **B** و **C** پارامترهای آن هستند. دستور **Return** مقدار متغیر **AVE** را که در داخل پرانتز قرار دارد به برنامهٔ فراخواننده برمی‌گرداند.

به همان صورتی که یک زیربرنامهٔ تابع به وسیله یک برنامهٔ فراخواننده احضار می‌شود، زیرالگوریتم MEAN نیز توسط یک الگوریتم احضار می‌شود. برای مثال، فرض کنید یک الگوریتم شامل دستور

$$TEST := MEAN(T_1 , T_2 , T_3)$$

است کـه درآن T_1، T_2 و T_3 نـمرهای آزمـون دانشـجویان هسـتند. مـقدار آرگـومان T_1، T_2 و T_3 در زیرالگوریتم به پارامترهای A، B و C داده‌می‌شود. زیرالگوریتم MEAN اجرامی‌شود و بدنبال آن مقدار AVE به برنامه برگردانده می‌شود و جایگزین $MEAN (T_1 , T_2 , T_3)$ در دستور بالا می‌شود. درنتیجهٔ آن، میانگین T_1 و T_2 و T_3 در TEST جایگزین می‌شود.

مثال ۱۰ـ۲

زیربرنامهٔ Procedure در زیر با نام SWITCH مقدار متغیرهای AAA و BBB را جابجا می‌کند.

Procedure 2.6: SWITCH(AAA, BBB)

1. Set TEMP := AAA, AAA := BBB and BBB := TEMP.
2. Return.

این زیربرنامه توسط دستور CALL صدا زده می‌شود. برای مثال با دستور CALL زیر

Call SWITCH(BEG, AUX)

مقادیر متغیرهای BEG و AUX جابجا می‌شوند. به ویژه این که، هنگام احضار زیـربرنامهٔ SWITCH ، آرگومان BEG و AUX به‌ترتیب به پارامترهای AAA و BBB منتقل می‌شود، زیربرنامهٔ Procedure اجرا می‌شود، که درنتیجهٔ آن مقادیر AAA و BBB جابجا می‌شوند و بدنبال آن مقادیر جدید AAA و BBB به ترتیب به BEG و AUX داده می‌شود.

توجه کنید : هر زیرالگوریتم تابع را به‌سادگی می‌توان به یک زیرالگوریتم Procedure معادل آن تبدیل کرد، به این صورت که، کافی است به الگوریتم فراخواننده یک پارامتر دیگر اضافه کنیم تا از آن برای برگرداندن مقدار محاسبه‌شده استفاده شود. مثلاً می‌توان تابع Function 2.5 را بـه صـورت زیـربرنامهٔ Procedure

$$MEAN(A , B , C , AVE)$$

نوشت که در پارامترِ AVE آن میانگین A ، B و C جایگزین می‌شود. آنگاه دستور

$$Call\ MEAN(T_1 , T_2 , T_3 , TEST)$$

دارای هـمان اثـر جـایگزینی مـیانگین T_1 ، T_2 و T_3 در TEST است. بـه بـیان کـلی‌تر، بـه جـای زیرالگوریتم‌های تابع می‌توانیم از زیرالگوریتم‌های Procedure استفاده کنیم.

۷ـ ۲ متغیرها و انواع داده‌ها

هر متغیر در یک الگوریتم یا برنامه، دارای یک نوعِ داده‌ای است که، آن نوع داده، کُدی را مشخص می‌کند که از آن برای ذخیره مقدار متغیر استفاده می‌شود. چهار نوع دادهٔ استاندارد به قرار زیر است :

(۱) نوع دادهٔ کاراکتری : در این حالت، داده‌ها با استفاده از نوعی کد کاراکتری مانند **EBCDIC** یا کد **ASCII** ذخیره می‌شوند. کد ۸ بیتی **EBCDIC** تعدادی از کاراکترها، در شکل ۷ـ ۲ ارائه شده است.

Char.	Zone	Numeric	Hex	Char.	Zone	Numeric	Hex	Char.	Zone	Numeric	Hex
A	1100	0001	C1	S	1110	0010	E2	*blank*	0100	0000	40
B		0010	C2	T		0011	E3	.		1011	4B
C		0011	C3	U		0100	E4	<		1100	4C
D		0100	C4	V		0101	E5	(		1101	4D
E		0101	C5	W		0110	E6	+	0100	1110	4E
F		0110	C6	X		0111	E7	&	0101	0000	50
G		0111	C7	Y		1000	E8	$		1011	5B
H		1000	C8	Z	1110	1001	E9	*		1100	5C
I	1100	1001	C9					)		1101	5D
J	1101	0001	D1	0	1111	0000	F0	;	0101	1110	5E
K		0010	D2	1		0001	F1	−	0110	0000	60
L		0011	D3	2		0010	F2	/		0001	61
M		0100	D4	3		0011	F3	,		1011	6B
N		0101	D5	4		0100	F4	%		1100	6C
O		0110	D6	5		0101	F5	>		1110	6E
P		0111	D7	6		0110	F6	?	0110	1111	6F
Q		1000	D8	7		0111	F7	:	0111	1010	7A
R	1101	1001	D9	8		1000	F8	#		1011	7B
				9	1111	1001	F9	@		1100	7C
								=	0111	1110	7E

شکل ۷ـ ۲ . بخشی از کُد **EBCDIC**

معمولاً یک کاراکتر در یک بایت حافظه ذخیره می‌شود.

(۲) نوع دادهٔ اعشاری (یا نقطه شناور): در این حالت داده‌های عددی با استفاده از صورت نماییِ داده‌ها ذخیره می‌شوند.

(۳) نوع دادهٔ صحیح (یا نقطه ثابت) : در این حالت اعداد صحیح مثبت با استفاده از نمایش دودویی ذخیره می‌شوند و اعداد صحیح منفی با یک تبدیل دودویی مانند **مکمل ۲** ذخیره می‌شوند.

(۴) نوع دادهٔ منطقی : در این حالت از کدگذاری، متغیر می‌تواند تنها مقدار **True** و **False** داشته باشد. از این رو متغیر تنها بااستفاده از یک بیت کدگذاری می‌شود. **1** برای **True** و **0** برای **False** . گاهی اوقات از بایتهای **1111 1111** و **0000 0000** به ترتیب برای **True** و **False** استفاده می‌شود.

در الگوریتم‌های این کتاب، نوع متغیرها مانند برنامه‌های کامپیوتری به صورت صریح بیان نمی‌شود اما در درون متن، به صورت ضمنی بیان می‌شود.

مثال ۱۱ـ ۲

فرض کنید X یک خانهٔ حافظهٔ ۳۲ بیتی با دنباله بیتهای زیر باشد. :

0110 1100 1100 0111 1101 0110 0110 1100

بجز در حالتی که نوع متغیر X بیان شده است به هیچ طریق نمی‌توان هیچگونه اطلاع دقیقی از محتوای این خانهٔ حافظه به دست آورد.

(الف) فرض کنید متغیر X دارای نوع کاراکتری است و برای آن از کُدگذاری **EBCDIC** هم استفاده شده است. در آن صورت چهار کاراکتر **%GO%** در متغیر X قابل ذخیره است.

(ب) فرض کنید X دارای نوع دیگری نظیر صحیح یا اعشاری است. در آن‌صورت می‌توان یک عدد صحیح یا اعشاری را در X ذخیره کرد.

متغیرهای محلی و سراسری

سازماندهی یک برنامهٔ کامپیوتری به صورت یک برنامهٔ اصلی و چند زیربرنامه منتهی به مفهوم **متغیرهای محلی و سراسری** شده است. معمولاً یک قطعه برنامه شامل لیستی از چند متغیرِ مربوط به خود موسوم به **متغیرهای محلی** است که فقط توسط این قطعه برنامه قابل دسترسی است. علاوه بر این، قطعه‌هایی که به صورت زیربرنامه هستند، می‌توانند شامل چند پارامتر، باشند این پارامترها، متغیرهایی هستند که داده‌ها را بین یک زیربرنامه و برنامهٔ فراخواننده انتقال می‌دهند.

مثال ۱۲ـ ۲

زیربرنامهٔ **Procedure** مثال ۱۰ـ ۲ یعنی **SWITCH(AAA , BBB)** را درنظر بگیرید. متغیرهای **AAA** و **BBB** پارامتر هستند. از این پارامترها برای انتقال داده‌ها بین زیربرنامهٔ **Procedure** و الگوریتم فراخواننده استفاده می‌شود. ازطرف دیگر متغیر **TEMP** در زیربرنامهٔ **Procedure** یک متغیر محلی است. محل "زندگی" یا "محیط زیست" این متغیر تنها در زیربرنامهٔ **Procedure** است یعنی مقدار آن تنها با اجرای این زیربرنامه قابل دسترسی و تغییر است. حقیقت این است که از نام **TEMP** می‌توان در هر قطعه برنامهٔ دیگر تحت عنوان نامِ یک متغیر استفاده کرد و استفاده از این نام در قطعه برنامهٔ دیگر هیچ تأثیری در اجرای زیربرنامهٔ **SWITCH** ندارد.

طراحان زبانهای برنامه‌نویسی بر این باورند که در یک برنامهٔ کامپیوتری بهتر است از چند متغیرِ مشخص استفاده شود، تا یک یا حتی تمام قطعه برنامه‌ها بتوانند به آن متغیرها دسترسی داشته باشند. متغیرهایی که تمام قطعه برنامه‌ها به آنها دسترسی دارند متغیرهای **سراسری** نام دارند و متغیرهایی که قطعه برنامه‌های معین به آنها دسترسی دارند متغیرهای **غیرمحلی** نام دارند. هر زبان برنامه‌نویسی برای معرفی چنین متغیرهایی، ساختار دستوری مختص به خود دارد. برای مثال، در زبان **FORTRAN** از دستور **COMMON** برای معرفی متغیرهای سراسری استفاده می‌شود و زبان **PASCAL** از قاعدهٔ دامنهٔ تغییرات متغیرها **SCOPE** برای معرفی متغیرهای سراسری و غیرمحلی استفاده می‌کند.

بنابراین، برای انتقال اطلاعات بین قطعه برنامه‌ها دو روش اساسی زیر وجود دارد :

(۱) **روش مستقیم** ، به کمک پارامترهایی که خوش تعریف‌اند.

(۲) **روش غیرمستقیم** ، به کمک متغیرهای غیرمحلی و سراسری.

تغییر غیرمستقیم مقدار متغیرِ یک قطعه برنامه توسط قطعه برنامهٔ دیگر **اثر جانبی** یا **عوارض جانبی Side Effect** نام دارد. دانشجویان می‌باید به هنگام استفاده از متغیر غیرمحلی و سراسری نهایت دقت را داشته باشند چون خطاهای ایجاد شده توسط اثر جانبی به سختی قابل کشف است.

مسأله‌های حل‌شده

نمادگذاری ریاضی و تابعهای کامپیوتری

مسأله ۱ـ ۲ : مطلوب است تعیین

(الف) $\lfloor 7.5 \rfloor, \lfloor -7.5 \rfloor, \lfloor -18 \rfloor, \lfloor \sqrt{30} \rfloor, \lfloor \sqrt[3]{30} \rfloor, \lfloor \pi \rfloor$

(ب) $\lceil 7.5 \rceil, \lceil -7.5 \rceil, \lceil -18 \rceil, \lceil \sqrt{30} \rceil, \lceil \sqrt[3]{30}, \lceil \pi \rceil$

حل : (الف) بنابه تعریف، $\lfloor x \rfloor$ بزرگترین عدد صحیحی را نمایش می‌دهد که بزرگتر از x نباشد و **کف x** نامیده می‌شود. درنتیجه :

$$\lfloor 7.5 \rfloor = 7 \qquad \lfloor -7.5 \rfloor = -8 \qquad \lfloor -18 \rfloor = -18$$
$$\lfloor \sqrt{30} \rfloor = 5 \qquad \lfloor \sqrt[3]{30} \rfloor = 3 \qquad \lfloor \pi \rfloor = 3$$

(ب) بنابه تعریف، $\lceil x \rceil$ کوچکترین عدد صحیحی را نمایش می‌دهد که کوچکتر از x نباشد و **سقف x** نامیده می‌شود. درنتیجه :

$$\lceil 7.5 \rceil = 8 \qquad \lceil -7.5 \rceil = -7 \qquad \lceil -18 \rceil = -18$$
$$\lceil \sqrt{30} \rceil = 6 \qquad \lceil \sqrt[3]{30} \rceil = 4 \qquad \lceil \pi \rceil = 4$$

مسأله ۲ـ ۲ :

(الف) مطلوب است تعیین $26 \pmod 7,\ 34 \pmod 8,\ 2345 \pmod 6,\ 495 \pmod{11}$

(ب) مطلوب است تعیین $-26 \pmod 7,\ -2345 \pmod 6,\ -371 \pmod 8,\ -39 \pmod 3$

(ج) با استفاده از حساب با باقیماندهٔ 15 ، حاصل $10-9,2-11,4+13,7+9$ را به دست آورید.

حل : (الف) چون k مثبت است تنها با تقسیم k بر M باقیماندهٔ r بدست می‌آید. در آن‌صورت $r \equiv K \pmod M$. بنابراین

$$5 = 26 \pmod 7 \qquad 2 = 34 \pmod 8 \qquad 5 = 2345 \pmod 6 \qquad 0 = 495 \pmod{11}$$

(ب) هرگاه k منفی باشد، با تقسیم $|k|$ بر M باقیمانده r' به دست می‌آید. در آن صورت $k \equiv -r' \pmod M$. هرگاه $r' \neq 0$ از این‌رو است که $k \pmod M = M - r'$. بنابراین

$$-26 \pmod 7 = 7 - 5 = 2 \qquad -371 \pmod 8 = 8 - 3 = 5$$
$$-2345 \pmod 6 = 6 - 5 = 1 \qquad -39 \pmod 3 = 0$$

(ج) از $a \pm M \equiv a \pmod M$ استفاده کنید.

$$9 + 13 = 22 \equiv 22 - 15 = 7 \qquad 7 + 11 = 18 \equiv 18 - 15 = 3$$
$$4 - 9 = -5 \equiv -5 + 15 = 10 \qquad 2 - 10 = -8 \equiv -8 + 15 = 7$$

مسأله ۳ـ ۲ : تمام جایگشتهای سه عدد 1 ، 2 ، 3 و 4 را بنویسید.

حل : ابتدا توجه داشته باشید که تعداد کل این جایگشتها $4! = 24$ است.

1234	1243	1324	1342	1423	1432
2134	2143	2314	2341	2413	2431
3124	3142	3214	3241	3412	3421
4123	4132	4213	4231	4312	4321

ملاحظه می‌کنید که شش جایگشت سطر اول با یک، شش جایگشت سطر دوم با ۲ شروع می‌شوند همینطور تا آخر.

مسأله ۴ـ ۲ : (الف) مطلوب است تعیین $2^{-5},\ 8^{2/3},\ 25^{-3/2}$;

(ب) $\log_2 32,\ \log_{10} 1000,\ \log_2 (1/16)$; (ج) $\lfloor \log_2 1000 \rfloor,\ \lfloor \log_2 0.01 \rfloor$

حل : (الف) $2^{-5} = 1/2^5 = 1/32;\ 8^{2/3} = (\sqrt[3]{8})^2 = 2^2 = 4;\ 25^{-3/2} = 1/25^{3/2} = 1/5^3 = 1/125$

(ب) $\log_2 32 = 5$ چون، $2^5 = 32$ ، $\log_{10} 1000 = 3$ چون ، $10^3 = 1000$

(ج) $\log_2(1/16) = -4$ چون، $2^{-4} = 1/2^4 = 1/16$

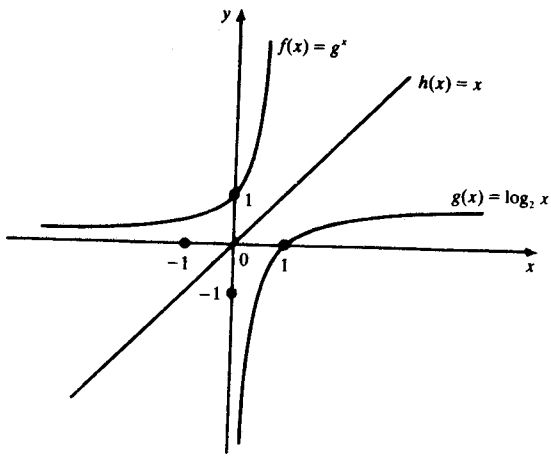
$\lfloor \log_2 1000 \rfloor = 9$ چون $2^9 = 512$, ، اما $2^{10} = 1024$

$\lfloor \log_2 0.01 \rfloor = -7$ چون $2^{-7} = 1/128 < 0.01 < 2^{-6} = 1/64$

مسأله ۵ـ ۲ : نمودار تابع نمایی $f(x) = 2^x$ ، تابع لگاریتمی $f(x) = \log_2 x$ و تابع خطی $h(x) = x$ را بر

روی یک محور مختصات رسم کنید. (الف) خاصیت هندسی نمودار $f(x)$ و $g(x)$ را بیان کنید. (ب) به‌ازای عدد مثبت دلخواه c، چه رابطه‌ای بین $f(c)$، $g(c)$ و $h(c)$ برقرار است؟

حل : در شکل ۸ـ ۲ نمودار این سه تابع رسم شده است.



شکل ۸ـ ۲

(الف) چون $f(x) = 2^x$ و $g(x) = \log_2 x$ معکوس یکدیگر هستند. از این‌رو نمودار این دو تابع نسبت به خط $y = x$ متقارن است.

(ب) به‌ازای هر عدد مثبت c، داریم :

$$g(c) < h(c) < f(c)$$

واقعیت این است که با افزایش مقدار c، فاصلهٔ قائم بین تابعهای

$$f(c) - h(c), \quad \text{و} \quad h(c) - g(c)$$

از نظر مقدار افزایش می‌یابد، علاوه بر این تابع لگاریتمی $g(x)$ درمقایسه با تابع خطی $h(x)$ رشد کندی دارد و رشد تابع نمایی $f(x)$ در مقایسه با تابع خطی $h(x)$ بسیار تند است.

الگوریتم‌ها، پیچیدگی آنها

مسأله ۶ـ ۲ : الگوریتم 2.3 را درنظر بگیرید که مکان LOC و مقدار MAX بزرگترین عنصر آرایهٔ DATA با n عنصر را پیدا می‌کند. $C(n)$ پیچیدگی تابع را درنظر بگیرید که تعداد دفعاتی را که LOC و MAX در

مرحلهٔ 3 تازه می‌شوند اندازه می‌گیرد. تعداد مقایسه‌ها مستقل از ترتیب قرارگیری عناصر در DATA است.

(الف) بدترین حالت را تشریح کنید و **C(n)** آن را پیدا کنید.

(ب) بهترین حالت را تشریح کنید و **C(n)** آن را پیدا کنید.

(ج) به‌ازای $n \geq 3$ ، **C(n)** را برای حالت میانگین پیدا کنید، فرض کنید ترتیب قرارگیری تمام عناصر در DATA با احتمال مساوی است.

حل : **(الف)** بدترین حالت وقتی اتفاق می‌افتد که عناصر DATA به ترتیب صعودی باشند که در آن، مقایسهٔ MAX با DATA[K] باعث می‌شود LOC و MAX تازه شوند، در این حالت $C(n) = n-1$.

(ب) بهترین حالت وقتی اتفاق می‌افتد که بزرگترین عنصر آرایه، اولین عنصر آن باشد و از این‌رو هنگام مقایسهٔ MAX با DATA[K] ، LOC و MAX هرگز تازه نمی‌شوند. بنابراین در این حالت $C(n) = 0$.

(ج) فرض کنید 1 ، 2 و 3 به ترتیب بزرگترین عنصر، دومین عنصر بزرگ آرایه و کوچکترین عنصر آرایه DATA باشند. عناصر در آرایهٔ DATA می‌توانند به شش طریق ممکن ظاهر شوند که متناظر با جایگشتهای سه‌عدد 1 ، 2، 3 یعنی $3! = 6$ است. برای هر جایگشت P ، فرض کنید n_p نمایش تعداد دفعات تازه‌شدن LOC و MAX باشد، که الگوریتم با ورودی P اجرا می‌شود. شش جایگشت P و مقادیر n_p ی متناظر آن به صورت زیر است :

جایگشت P :	321	312	231	213	132	123
مقادیر n_p :	2	1	1	1	0	0

با فرض این که تمام جایگشت‌ها با احتمال مساوی باشند :

$$C(3) = \frac{0+0+1+1+1+2}{6} = \frac{5}{6}$$

محاسبهٔ مقدار میانگین $C(n)$ برای n دلخواه، خارج از حدود مطالب درس ساختمان داده‌ها است. یکی از اهداف این مسأله، آن است که نشان دهیم هنگام تعیین پیچیدگی حالت میانگین یک الگوریتم، با چه مشکلاتی دست به گریبان هستیم.

مسألهٔ ۷ـ ۲ : فرض کنید M واحد زمانی، لازم است تا قطعه برنامهٔ A اجرا شود که در آن M یک مقدار ثابت است. هرگاه n تعداد داده‌های ورودی و h عدد مثبتی بزرگتر از یک باشد، $C(n)$ پیچیدگی هر یک از دو الگوریتم زیر را به دست آورید.

(الف) الگوریتم P 2–7 A :

Algorithm P2.7A:
1. Repeat for I = 1 to N:
2. Repeat for J = 1 to N:
3. Repeat for K = 1 to N:
4. Module A.
[End of Step 3 loop.]
[End of Step 2 loop.]
[End of Step 1 loop.]
5. Exit.

(ب) الگوریتم P 2–7 B :

Algorithm P2.7B:
1. Set J := 1.
2. Repeat Steps 3 and 4 while J ≤ N:
3. Module A.
4. Set J := B × J.
[End of Step 2 loop.]
5. Exit.

ملاحظه می‌کنید که در این الگوریتم به جای n از N و به جای b از B استفاده شده است.

حل : (الف) در اینجا

$$C(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} M$$

تعداد دفعاتی که M در این مجموع ظاهر می‌شود برابر تعداد سه‌تایی‌های (i,j,k) است که در آن i، j، k اعداد صحیح از 1 تا خود n هستند. تعداد n^3 سه‌تایی از این نوع وجود دارد. از این‌رو

$$C(n) = Mn^3 = O(n^3)$$

(ب) ملاحظه می‌کنید که مقادیر شاخص حلقه J توانهایی از b هستند :

$$1, b, b^2, b^3, b^4, .$$

بنابراین قطعه برنامهٔ A دقیقاً T بار تکرار می‌شود که در آن T اولین توانی از b است که

در آن $b^T > n$

درنتیجه : $T = \lfloor \log_b n \rfloor + 1$

بنابراین : $C(n) = MT = O(\log_b n)$

مسأله ۸ـ ۲ : (الف) یک زیربرنامهٔ Procedure به نام FIND(DATA, N, LOC1, LOC2) بنویسید که LOC1 مکان بزرگترین عنصر و LOC2 مکان دومین عنصر بزرگ آرایهٔ DATA را برای n>1 عنصر پیدا کند.

حل : (الف) عناصر DATA را یک به یک مورد بررسی قرار دهید. درخلال اجرای این زیربرنامهٔ procedure ، FIRST و SECOND به ترتیب مقادیر بزرگترین عنصر و دومین عنصر بزرگ آرایه راکه قبلاً مورد بررسی قرارگرفتند نمایش می‌دهند. هر عنصر جدید DATA[K] به صورت زیر آزمایش می‌شود که اگر

$$SECOND \le FIRST < DATA[K]$$

آنگاه FIRST عنصر SECOND جدید و DATA[K] عنصر FIRST جدید می‌شود. ازطرف دیگر اگر

$$SECOND < DATA[K] \le FIRST$$

آنگاه DATA[K] عنصر SECOND جدید می‌شود. درآغاز قرار دهید FIRST:=DATA[1] و SECOND:= DATA [2] و بررسی کنید که آیا این عنصرها با ترتیبِ درست قرار گرفته‌اند یا خیر. نمایش رسمی این زیربرنامه به صورت زیر است :

```
Procedure P2.8: FIND(DATA, N, LOC1, LOC2)
    1. Set FIRST := DATA[1], SECOND := DATA[2], LOC1 := 1, LOC2 := 2.
    2. [Are FIRST and SECOND initially correct?]
       If FIRST < SECOND, then:
          (a) Interchange FIRST and SECOND,
          (b) Set LOC1 := 2 and LOC2 := 1.
       [End of If structure.]
    3. Repeat for K = 3 to N:
          If FIRST < DATA[K], then:
             (a) Set SECOND := FIRST and FIRST := DATA[K].
             (b) Set LOC2 := LOC1 and LOC1 := K.
          Else if SECOND < DATA[K], then:
             Set SECOND := DATA[K] and LOC2 := K.
          [End of If structure.]
       [End of loop.]
    4. Return.
```

(ب) استفاده از پارامتر اضافی FIRST و SECOND به‌نظر زاید می‌آید چون LOC1 و LOC2 خود به خود به برنامهٔ فراخواننده می‌گویند که DATA[LOC1] و DATA[LOC2] به ترتیب مقادیر بزرگترین عنصر و دومین عنصر بزرگ آرایهٔ DATA هستند.

مسأله ۹ـ ۲ : عدد صحیح $n>1$ یک **عدد اول** نامیده می‌شود اگر مقسوم علیه‌های مثبت آن تنها 1 و n باشند درغیر اینصورت به n یک **عدد مرکب** می‌گویند، برای مثال، در زیر لیست اعداد اول کوچکتر از 20 ارائه شده است :

2, 3, 5, 7, 11, 13, 17, 19

اگر $n>1$ اول نباشد، یعنی اگر n یک عدد مرکب باشد آنگاه n باید مقسوم علیهی مانند $K \neq 1$ داشته باشد بطوری که $k \le \sqrt{n}$ یا به بیان دیگر $k^2 \le n$.

فرض کنید خواسته باشیم تمام اعداد اول کوچکتر از یک عدد معلوم m مانند 30 را به دست آوریم.

این کار با روشی موسوم به "روش غربال اِراتُستِن" عملی است که از مراحل زیر تشکیل شده است.

لیست تمام اعداد از یک تا 30 را بنویسید :

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30

عدد 1 و تمام مضربهای عدد 2 به غیر از خود 2 را به صورت زیر حذف کنید :

~~1~~, 2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, 9, ~~10~~, 11, ~~12~~, 13, ~~14~~, 15
~~16~~, 17, ~~18~~, 19, ~~20~~, 21, ~~22~~, 23, ~~24~~, 25, ~~26~~, 27, ~~28~~, 29, ~~30~~

حال چون عدد 3 اولین عدد بعد از 2 است که حذف نشده است در این مرحله تمام مضربهای عدد 3 غیر از خود 3 را به صورت زیر از لیست حذف کنید :

~~1~~, 2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~, 11, ~~12~~, 13, ~~14~~, ~~15~~
~~16~~, 17, ~~18~~, 19, ~~20~~, ~~21~~, ~~22~~, 23, ~~24~~, 25, ~~26~~, ~~27~~, ~~28~~, 29, ~~30~~

حال چون عدد 5 اولین عدد بعد از 3 است که حذف نشده است در این مرحله تمام مضربهای عدد 5 غیر از خود 5 را به صورت زیر از لیست حذف کنید :

~~1~~, 2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~, 11, ~~12~~, 13, ~~14~~, ~~15~~
~~16~~, 17, ~~18~~, 19, ~~20~~, ~~21~~, ~~22~~, 23, ~~24~~, ~~25~~, ~~26~~, ~~27~~, ~~28~~, 29, ~~30~~

حال چون عدد 7 اولین عدد بعد از 5 است که حذف نشده است و چون $30 < 7^2$. این الگوریتم به پایان رسیده است و اعداد باقیمانده در لیست، اعداد اول کوچکتر از 30 هستند :

2, 3, 5, 7, 11, 13, 17, 19, 23, 29

باتوجه به توضیحات فوق، روش غربال اِراتُستِن برای پیداکردن کلیهٔ اعداد اول کوچکتر از عدد معلوم n را به یک الگوریتم تبدیل کنید.

حل : ابتدا آرایهٔ A را به‌گونه‌ای تعریف می‌کنیم که

A[1] = 1, A[2] = 2, A[3] = 3, A[4] = 4, . . . , A[N − 1] = N − 1, A[N] = N

عدد صحیح L را با دستور جایگزینی A[L] = 1 از لیست حذف می‌کنیم زیربرنامهٔ Procedure ، CROSSOUT زیر، آزمایش می‌کند که آیا A[K] = 1 یا خیر، درصورت منفی بودن جواب، قرار می‌دهد :

A[2K] = 1, A[3K] = 1, A[4K] = 1, . . .

یعنی مضربهای K از لیست حذف شده‌اند :

Procedure P2.9A: CROSSOUT(A, N, K)

1. If A[K] = 1, then: Return.
2. Repeat for L = 2K to N by K:
   Set A[L] := 1.
   [End of loop.]
3. Return.

روش غربال را اکنون می‌توان به صورت سادهٔ زیر نوشت :

Algorithm P2.9B: This algorithm prints the prime numbers less than N.

1. [Initialize array A.] Repeat for K = 1 to N:
   Set A[K] := K.
2. [Eliminate multiples of K.] Repeat for K = 2 to $\sqrt{N}$.
   Call CROSSOUT(A, N, K).
3. [Print the primes.] Repeat for K = 2 to N:
   If A[K] ≠ 1, then: Write: A[K].
4. Exit.

مسأله ۱۰ـ ۲ : فرض کنید $P_{(n)} = a_0 + a_1 n + a_2 n^2 + \ldots + a_m n^m$ یعنی فرض کنید درجهٔ P(n) برابر m باشد. ثابت کنید که $P(n) = O(n^m)$.

حل : اگر $b_0 = |a_0|, b_1 = |a_1|, \ldots, b_m = |a_m|.$ آنگاه به ازای $n \geq 1$ ،

$$P(n) \leq b_0 + b_1 n + b_2 n^2 + \cdots + b_m n^m = \left(\frac{b_0}{n^m} + \frac{b_1}{n^{m-1}} + \cdots + b_m\right) n^m$$
$$\leq (b_0 + b_1 + \cdots + b_m) n^m = M n^m$$

که در آن $M = |a_0| + |a_1| + \cdots + |a_m|.$ از این رو $P(n) = O(n^m)$.

برای مثال $5x^3 + 3x = O(x^3)$ و $x^5 - 4\,000\,000x^2 = O(x^5)$.

متغیرها، انواع داده‌ای

مسأله ۱۱ـ ۲ : به اختصار تفاوت بین متغیرهای محلی، پارامترها و متغیرهای سراسری را بیان کنید.

حل : متغیرهای محلی متغیرهایی هستند که تنها در درون یک برنامهٔ خاص یا یک زیربرنامه، قابل دسترس هستند. پارامترها، متغیرهایی هستند که برای انتقال داده‌ها بین یک زیربرنامه و برنامهٔ فراخواننده بکار می‌رود. متغیرهای سراسری، متغیرهایی هستند که تمام قطعه برنامه‌های یک برنامهٔ کامپیوتری می‌توانند به آنها دسترسی داشته باشند. هر زبان برنامه‌نویسی که استفاده از متغیرهای سراسری را مجاز دانسته باشد دارای دستوراتی برای معرفی این نوع متغیرها است.

مسأله ۱۲ـ ۲ : فرض کنید NUM معرف تعداد رکوردهای یک فایل باشد. مزایای تعریف NUM را به صورت یک متغیر سراسری شرح دهید. معایب استفاده از متغیرهای سراسری را در حالت کلی شرح

دهید.

حل : در بسیاری از زیربرنامه‌های **Procedure** پردازش رکوردهای یک فایل با استفاده از نوعی حلقه صورت می‌گیرد. از آنجا که تمام این زیربرنامه‌ها، تنها از یک نام برای رکورد، یعنی، **NUM** استفاده می‌کنند که این یکی از عیبهای معرفی **NUM** به صورت یک متغیر سراسری است، به بیان کلی‌تر، متغیرهای سراسری و غیرمحلی ممکن است در اثر عوارض جانبی **Side Effect** منجر به خطاهایی **errors** شوند که احتمالاً کشف آنها مشکل است.

مسأله ۱۳ـ ۲ : فرض کنید **AAA** یک خانهٔ حافظه ۳۲ بیتی با دنباله بیتهای زیر باشد :

0100 1101 1100 0001 1110 1001 0101 1101

مطلوب است تعیین دادهٔ ذخیره‌شده در **AAA**.

حل : بجز در حالتی که نوع داده‌ای **AAA** را می‌دانیم، هیچ راهی برای تعیین دادهٔ ذخیره شده در **AAA** وجود ندارد. اگر **AAA** یک متغیر کاراکتری باشد و از کدگذاری **EBCDIC** نیز برای ذخیرهٔ داده‌ها استفاده شود آنگاه **(AZ)** در **AAA** ذخیره می‌شود. اگر **AAA** یک متغیر صحیح باشد، آنگاه یک عدد صحیح با نمایش دودویی بالا در **AAA** ذخیره می‌شود.

مسأله ۱۴ـ ۲ : از نظر ریاضی، اعداد صحیح را می‌توان مانند اعداد حقیقی (اعشاری) مورد بررسی قرار داد. دلایلی برای داشتن دو نوع دادهٔ مختلف صحیح و اعشاری ارائه دهید.

حل : حساب اعداد صحیح که با استفاده از نوعی نمایش دودویی در کامپیوتر ذخیره می‌شوند خیلی ساده‌تر از حساب اعداد اعشاری است که به صورت نمایی ذخیره می‌شوند. علاوه بر این خطای ناشی از گردکردن که در اعداد اعشاری اتفاق می‌افتد در حساب اعداد صحیح وجود ندارد.

مسأله‌های تکمیلی

نمادهای ریاضی و تابعهای کامپیوتری

مسأله ۱۵ـ ۲ : مطلوب است تعیین

(الف) $\lfloor 3.4 \rfloor, \lfloor -3.4 \rfloor, \lfloor -7 \rfloor, \lfloor \sqrt{75} \rfloor, \lfloor \sqrt[3]{75} \rfloor, \lfloor e \rfloor$

(ب) $\lceil 3.4 \rceil, \lceil -3.4 \rceil, \lceil -7 \rceil, \lceil \sqrt{75} \rceil, \lceil \sqrt[3]{75} \rceil, \lceil e \rceil$

مسأله ۱۶ـ ۲ : (الف) مطلوب است محاسبهٔ $48 \pmod 5, 48 \pmod 7, 1397 \pmod{11}, 2468 \pmod 9$

(ب) مطلوب است محاسبهٔ $-48 \pmod 5, -152 \pmod 7, -358 \pmod{11}, -1326 \pmod{13}$

(ج) با استفاده از حساب با باقیماندهٔ 13 ، مطلوب است محاسبهٔ

$$9+10, \quad 8+12, \quad 3+4, \quad 3-4, \quad 2-7, \quad 5-8$$

مسأله ۱۷ـ ۲ : مطلوب است تعیین

(الف) $|3+8|, |3-8|, |-3+8|, |-3-8|$

(ب) $7!, 8!, 14!/12!, 15!/16!$

مسأله ۱۸ـ ۲ : مطلوب است محاسبهٔ

(الف) $3^{-4}, 4^{7/2}, 27^{-2/3}$

(ب) $\log_2 64, \log_{10} 0.001, \log_2 (1/8)$

(ج) $\lfloor \lg 1\,000\,000 \rfloor, \lfloor \lg 0.001 \rfloor$

الگوریتم‌ها، پیچیدگی آنها

مسأله ۱۹ـ ۲ : تابع پیچیدگی $C(n)$ را درنظر بگیرید که تعداد دفعاتی را که **LOC** در مرحلهٔ **3** ی الگوریتم **2.3** تازه می‌شود اندازه می‌گیرد. به‌ازای $n = 4$ ، حالت میانگین $C(n)$ را پیداکنید. فرض می‌شود ترتیب تمام‌چهار عنصر داده‌شده با احتمال برابر است. مسأله ۶ـ ۲ را نیز ببینید.

مسأله ۲۰ـ ۲ : زیربرنامهٔ **Procedure P2.8** را درنظر بگیرید که **LOC1** مکان بزرگترین عنصر و **LOC2** مکان دومین عنصر بزرگ آرایهٔ **DATA** را که $n>1$ ، پیدا می‌کند. فرض کنید $C(n)$ نمایش تعداد مقایسه‌ها درخلال اجرای این **Procedure** باشد.

(الف) $C(n)$ را در بهترین حالت پیداکنید.

(ب) $C(n)$ را در بدترین حالت پیداکنید.

(ج) به‌ازای $n = 4$ ، $C(n)$ را در حالت میانگین پیداکنید. فرض می‌شود ترتیب تمام عناصر داده‌شده در **DATA** با احتمال برابر است.

مسأله ۲۱ـ ۲ : مسأله ۲۰ـ ۲ را مجدداً حل کنید با این تفاوت که اکنون فرض کنید که $C(n)$ نمایش تعداد دفعاتی باشد که باید مقدارهای **FIRST** و **SECOND** (یا **LOC1** و **LOC2**) تازه شوند.

مسأله ۲۲ـ ۲ : فرض کنید که زمان اجرای قطعه برنامهٔ **A** مقدار ثابت **M** است. مرتبهٔ بزرگی تابع پیچیدگی $C(n)$ را به دست آورید که زمان اجرای الگوریتم‌های زیر را که در آن **n** تعداد داده‌های ورودی است پیدا می‌کند. **n** در الگوریتم‌ها با **N** نشان داده شده است.

(الف)

```
Procedure P2.22A:  1.  Repeat for I = 1 to N:
                   2.      Repeat for J = 1 to I:
                   3.          Repeat for K = 1 to J:
                   4.              Module A.
                           [End of Step 3 loop.]
                       [End of Step 2 loop.]
                   [End of Step 1 loop.]
                   5.  Exit.
```

(ب)

```
Procedure P2.22B:  1. Set J := N.
                   2. Repeat Steps 3 and 4 while J > 1.
                   3.     Module A.
                   4.     Set J := J/2.
                      [End of Step 2 loop.]
                   5. Return.
```

برای مسأله‌های زیر، برنامه بنویسید

مسأله ۲۳ـ ۲ : یک زیربرنامهٔ تابع **DIV(J,K)** بنویسید که در آن **J** و **K** اعداد صحیح مثبت هستند طوری که اگر **J** یک عامل **K** باشد **DIV(J,K) = 1**، درغیر اینصورت **DIV(J,K) = 0** . (برای مثال **DIV(3,15) = 1** اما **DIV(3,16) = 0** .

مسأله ۲۴ـ ۲ : با استفاده از **DIV(J,K)** برنامه‌ای بنویسید که یک عدد صحیح مثبت **N>10** را بخواند و تعیین کند که آیا **N** اول است یا خیر.

راهنمایی : **N** اول است اگر **(i) DIV(2,N) = 0** یعنی **N** فرد باشد و **(ii)** به‌ازای تمام اعداد صحیح فرد **K** که $1<K^2 \leq N$ ، **DIV(K,N) = 0** .

مسأله ۲۵ـ ۲ : زیربرنامهٔ **Procedure 2.8** را به یک برنامهٔ کامپیوتری تبدیل کنید یعنی برنامه‌ای بنویسید که **LOC1** مکان بزرگترین عنصر و **LOC2** مکان دومین عنصر بزرگ آرایهٔ **DATA** را که دارای **N>1** عنصر است پیدا کند. برنامه را با استفاده از **70** ، **30** ، **25** ، **80** ، **60** ، **50** ، **30** ، **75** ، **25** و **60** آزمایش کنید.

مسأله ۲۶ـ ۲ : روش غربال اِراتُستِن برای تعیین اعداد اول را که در مسأله ۹ـ ۲ شرح داده شده است به یک برنامهٔ کامپیوتری تبدیل کنید تا اعداد اول کوچکتر از **N** را پیدا کند. برنامه را با استفاده از **(الف)** **N = 1000** و **(ب)** **N = 10000** آزمایش کنید.

مسأله ۲۷ـ ۲ : فرض کنید **C** نمایش تعداد دفعاتی باشد که **LOC** در الگوریتم ۳ـ ۲ برای تعیین بزرگترین عنصر آرایهٔ **N** عنصریِ **A** ، تازه می‌شود.

(الف) یک زیربرنامهٔ **COUNT(A,N,C)** برای تعیین **C** بنویسید.

(ب) یک زیربرنامهٔ **Procedure P2.27** بنویسید که **(i) N** عدد تصادفی بین **0** و **1** در آرایهٔ **A** بخواند و **(ii)** با استفاده از **COUNT(A,N,C)** مقدار **C** را پیدا کند.

(ج) برنامه‌ای بنویسید که زیربرنامهٔ **Procedure P2.27** را **1000** بار اجرا کند و میانگین هزار **C** را پیدا کند.

(i) به‌ازای **N = 3** برنامه را آزمایش کنید و نتیجه را با مقدار به‌دست آمده در مسأله ۶ـ ۲ مقایسه کنید.

(ii) به‌ازای **N = 4** برنامه را اجرا کنید و نتیجه را با مقدار به‌دست آمده در مسأله ۲۹ـ ۲ مقایسه کنید.

فصل ۳

پردازش رشته‌ها

۱ـ۳ مقدمه

ازنظر تاریخی، از کامپیوتر نخست به منظور پردازش داده‌های عددی استفاده می‌شد. امروزه، اغلب کامپیوتر برای پردازش داده‌های غیرعددی موسوم به **داده‌های کاراکتری** مورد استفاده قرار می‌گیرد. در این فصل چگونگی ذخیرهٔ داده‌های کاراکتری و پردازش آنها بررسی می‌شود.

امروزه یکی از کاربردهای اصلی کامپیوتر در عرصهٔ پردازش کلمات یا رشته‌ها است. معمولاً چنین پردازشهایی شامل نوعی از تطبیق الگو است نظیر تحقیق در مورد این که آیا یک کلمهٔ خاص مانند S در متن داده شدهٔ T وجود دارد یا خیر. ما این مسألهٔ تطبیق الگو را به‌طور مشروح بررسی می‌کنیم و علاوه بر این دو الگوریتم مختلف برای تطبیق الگوها ارائه خواهیم داد.

در دروس کامپیوتری معمولاً برای دنباله‌ای از کاراکترها به جای اصطلاح "کلمه" از اصطلاح "رشته" استفاده می‌شود چون "کلمه" دارای معنی دیگری در علم کامپیوتر است. به همین دلیل، در بسیاری از متنها و کتابهای کامپیوتری از عبارت "پردازش رشته‌ها"، "عملیات بر روی رشته‌ها" یا "ویرایش متن" به جای عبارت "پردازش کلمه" استفاده می‌شود.

مطالب این فصل اساساً جهت یادآوری بیان شده و مستقل از بقیهٔ مطالب کتاب است. بنابراین دانشجو یا استاد می‌تواند در مطالعه اول این فصل را حذف کند و یا مطالعه آن را تا زمان دیگر به تعویق بیندازد.

۲ـ۳ اصطلاحات پایه‌ای

هر زبان برنامه‌نویسی دارای **یک مجموعه از کاراکترها** است که از آنها برای برقراری ارتباط با کامپیوتر استفاده می‌کند. این مجموعه معمولاً شامل کاراکترهای زیر است :

کاراکترهای الفبایی : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

کاراکترهای رقمی : 0 1 2 3 4 5 6 7 8 9

کاراکترهای مخصوص : + − / * ( ) , . $ = ' □

مجموعه کاراکترهای مخصوص که شامل فضای خالی نیز است غالباً با □ نمایش داده می‌شود که اندکی از یک زبان برنامه‌نویسی تا زبان دیگر در تغییر است.

رشته یک دنبالهٔ متناهی **S** از صفر، یک یا چند کاراکتر است. تعداد کاراکترهای یک رشته، **طول رشته** نام دارد. رشته‌ای که هیچ کاراکتری نداشته باشد، **رشتهٔ تهی** یا **رشتهٔ پوچ** نام دارد. رشته‌ها را با محصورکردن کاراکترهای آن بین یک علامت نقل‌قول یا دو علامت نقل‌قول نمایش می‌دهند. علامت نقل‌قول یا کوتیشن **Quotation** به عنوان علائم ابتدا و انتهای رشته یا محدودکنندهٔ رشته نیز بکار می‌رود. از این رو

'THE END' 'TO BE OR NOT TO BE' ' ', '□□'

رشته‌هایی به ترتیب با طول **7** ، **18** ، **0** و **2** هستند. تأکید می‌کنیم که فضای خالی، یک کاراکتر است و درنتیجه در محاسبهٔ طول رشته‌ها شرکت می‌کند. گاهی اوقات علامت کوتیشن را می‌توان حذف کرد و این هنگامی است که از خود متن چنین استنباط شود که عبارت داده شده یک رشته است.

فرض کنید S_1 و S_2 دو رشته باشند. رشته‌ای که از ترکیب کاراکترهای S_1 و بدنبال آن کاراکترهای رشته S_2 حاصل می‌شود **پیوند** یا **اتصال** S_1 و S_2 نامیده می‌شود. ما در این کتاب اتصال دو رشتهٔ S_1 و S_2 را به صورت $S_1 // S_2$ نمایش می‌دهیم. برای مثال :

'THE' // 'END' = 'THEEND' اما 'THE' // '□' // 'END = 'THE END'

واضح است که طول $S_1 // S_2$ برابر مجموع طول‌های دو رشتهٔ S_1 و S_2 است. رشتهٔ **Y زیررشتهٔ**، رشتهٔ **S** نامیده می‌شود هرگاه رشته‌هایی مانند **X** و **Z** وجود داشته باشد طوری که : $S = X // Y // Z$.

اگر **X** یک رشتهٔ تهی باشد، آنگاه **Y** یک **زیر رشتهٔ اولیه** یا **ابتدایی** **S** نامیده می‌شود و اگر **Z** یک رشتهٔ تهی باشد آنگاه **Y** یک **زیررشتهٔ انتهایی** **S** نامیده می‌شود. برای مثال :

'BE OR NOT' یک زیر رشتهٔ 'TO BE OR NOT TO BE' است.

'THE' یک زیر رشتهٔ ابتدایی 'THE END' است.

واضح است که اگر **Y** یک زیر رشتهٔ **S** باشد آنگاه طول **Y** نمی‌تواند بیشتر از طول **S** باشد.

توجه کنید : کاراکترها در کامپیوتر به کمک یک کُد 6 بیتی، یک کد 7 بیتی، یا یک کد 8 بیتی ذخیره می‌شوند. واحدی که برابر تعداد بیتهای موردنیاز برای نمایش یک کاراکتر است یک بایت نامیده می‌شود. کامپیوتری که بتواند به یک بایت از حافظه دسترسی پیدا کند، یک ماشین با قابلیت آدرس‌دهیِ بایتی نامیده می‌شود.

۳ـ۳ ذخیرهٔ رشته‌ها

در حالت کلی، رشته‌ها را می‌توان با سه ساختار مختلف ذخیره کرد. (۱) ***ساختارهایی که طول ثابت دارند.*** (۲) ***ساختارهایی با طول متغیر که ما گزیینم طول آن ثابت است و*** (۳) ***ساختارهای پیوندی.*** هر نوع ساختار به‌طور جداگانه مورد بررسی قرار می‌گیرد، مزایا و معایب آنها بیان می‌شود.

رکورد ، حافظه با طول ثابت

در حافظه با طول ثابت هر خطِ چاپ، یک رکورد محسوب می‌شود که در آن تمام رکوردها دارای طول برابر هستند، یعنی در هر رکورد تعداد کاراکترها برابر می‌باشند. از آنجا که داده‌ها اغلب به صورت تصویرهای ۸۰ ستونی به عنوان ورودی به ترمینالها داده می‌شوند یا از کارتهای ۸۰ ستونی استفاده می‌کنند، فرض می‌کنیم که رکوردها طول ۸۰ دارند، در غیراینصورت طول رکورد به صورت صریح یا ضمنی بیان می‌گردد.

مثال ۱ـ۳

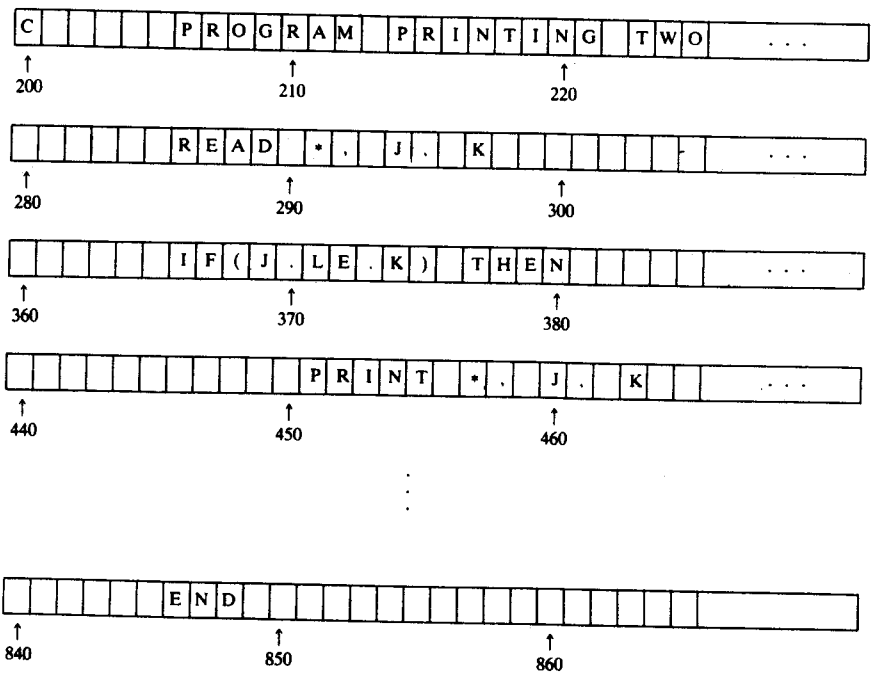
فرض کنید دادهٔ ورودی کامپیوتر، برنامهٔ **FORTRAN** شکل ۱ـ۳ است.

```
C    PROGRAM PRINTING TWO INTEGERS IN INCREASING ORDER
     READ *, J, K
     IF(J.LE.K) THEN
        PRINT *, J, K
     ELSE
        PRINT *, K, J
     ENDIF
     STOP
     END
```

شکل ۱ـ۳. دادهٔ ورودی

با استفاده از یک رکورد برای محیط حافظه با طول ثابت، دادهٔ ورودی در حافظه به‌صورتی که در

شکل ۲ـ ۳ ارائه شده است نمایش داده می‌شود که در آن فرض شده است **200** ، آدرس اولین کاراکتر برنامه است.



شکل ۲ـ ۳. رکوردها به صورت متوالی در کامپیوتر ذخیره می‌شوند.

مزایای اصلی روش بالا در ذخیرهٔ رشته‌ها عبارت است از :

(۱) دسترسی آسان به اطلاعات هر رکورد معین

(۲) تازه‌سازی آسان اطلاعات هر رکورد معین (مشروط بر این که طول اطلاعات جدید بیشتر از طول رکورد نباشد)

معایب عمدهٔ روش بالا عبارت است از :

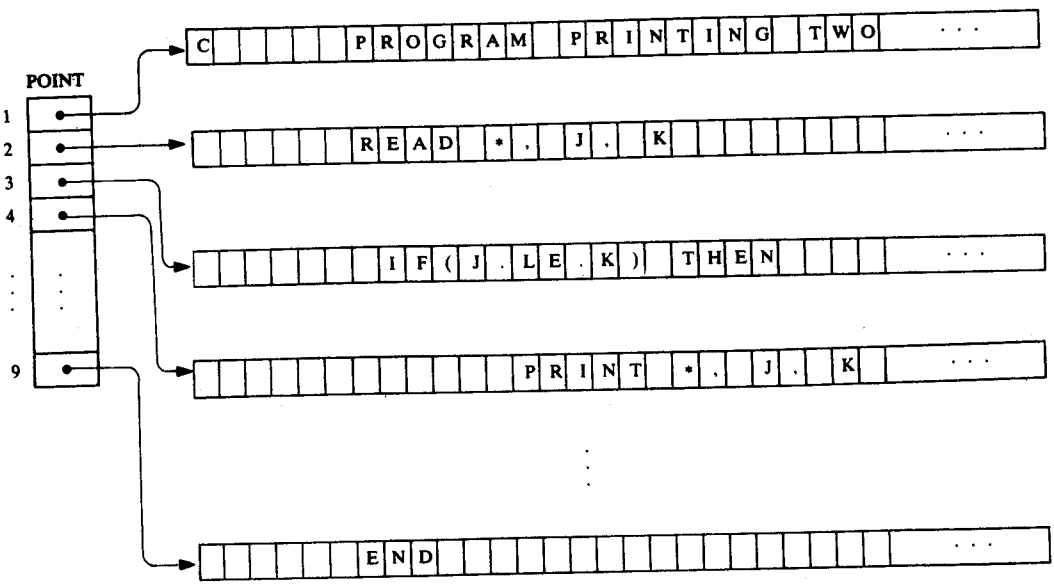
(۱) هرگاه اکثر حافظه از فضاهای خالی تشکیل شده باشد خواندن تمام رکورد باعث به‌هدر رفتن زمان می‌شود.

(۲) بعضی از رکوردها به حافظه بیشتر از حافظه در دسترس، نیاز دارند.

(۳) هرگاه در متن اصلی احتیاج به تصحیح یک یا چند کاراکتر باشد، آنگاه برای تغییر یک کلمهٔ نادرست

لازم است تمام رکوردها تغییر کند.

توجه کنید : فرض کنید بخواهیم در مثال ۱ـ۳ یک رکورد جدید اضافه کنیم. این کار مستلزم آن است که تمام رکوردهای بعدی به خانه‌های حافظه جدید منتقل شوند. با وجود این، این عیب را می‌توان به صورتی که در شکل ۳ـ۳ بیان شده است، برطرف کرد.



شکل ۳ـ۳ . ذخیره‌سازی رکوردها با استفاده از اشاره‌گرها

یعنی می‌توان از یک آرایه خطی **POINT** که آدرس هر رکورد متوالی را به دست می‌دهد استفاده کرد طوری که لازم نباشد رکوردها در خانه‌های متوالی حافظه ذخیره شوند. بدین ترتیب اضافه کردن یک رکورد جدید تنها مستلزم تازه‌سازی آرایهٔ **POINT** است.

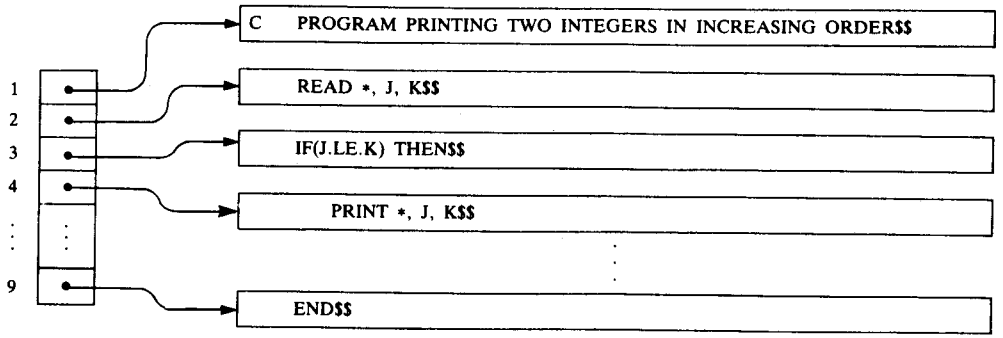
حافظه با طول متغیر، که ماگزیمم طول آن ثابت است

اگر چه مانند بالا رشته‌ها را می‌توان با طول ثابت در خانه‌های حافظه ذخیره کرد اما بیان طول واقعی هر رشته دارای معایبی است. مثلاً هنگامی که رشته تنها بخش آغازین خانهٔ حافظه را اشغال می‌کند در آن صورت نباید تمام رکورد خوانده شود. علاوه بر این بعضی از عملیات روی رشته‌ها (که در بخش ۴ـ۳ بررسی می‌شود) بستگی به داشتن رشته‌هایی با طول متغیر دارند.

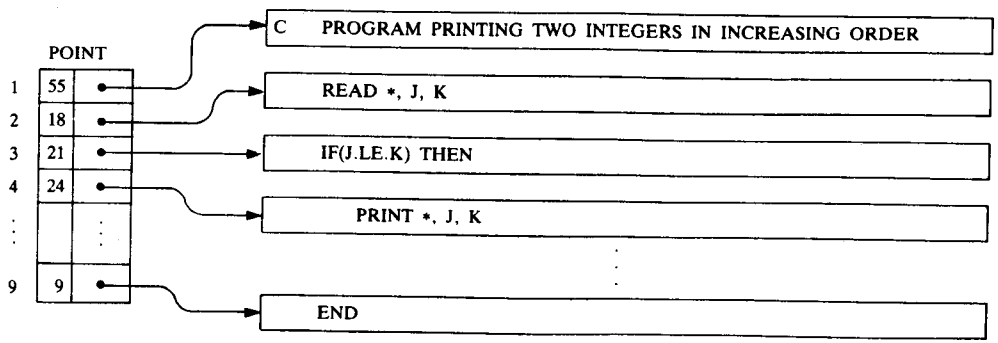
ذخیره رشته‌هایی با طول متغیر در حافظه راکه طول ثابت دارند می‌توان به دو روش کلی زیر انجام داد :

(۱) می‌توان برای پایان رشته از یک علامت، نظیر دو علامت دلار $ $ استفاده کرد.

(۲) می‌توان طول رشته را مثلاً به عنوان یک فیلد اضافی در آرایهٔ اشاره‌گر بیان کرد. با استفاده از داده‌های شکل ۱ـ ۳، روش اول در نمودار ۴ـ ۳ (الف) و روش دوم در نمودار ۴ـ ۳ (ب) به تصویر کشیده شده است.



(الف) رکوردها به همراه علامت پایان رشته

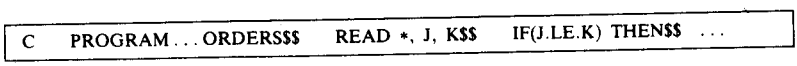


(ب) رکوردهایی که طولشان به صورت پیوندی هستند.

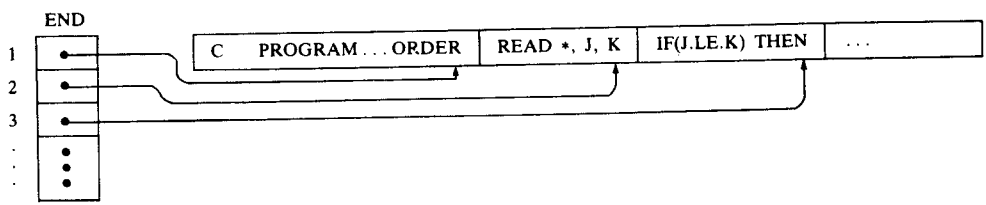
شکل ۴ـ ۳

توجه کنید : بعضی از دانشجویان ممکن است دچار این وسوسهٔ اغواآمیز شوند که بخواهند رشته‌ها را

یکی بعد از دیگری با استفاده از یک علامت خاص نظیر دو علامت دلار $$ شکل ۵ـ ۳ (الف) یا با استفاده از یک آرایهٔ اشاره‌گر که مکان رشته‌ها را مانند شکل ۵ـ ۳ (ب) به دست می‌دهد ذخیره کنند. واضح است که این‌گونه روشهای ذخیرهٔ رشته‌ها باعث صرفه‌جویی در مقدار حافظهٔ مصرفی می‌شود، و گاهی اوقات هنگامی که رکوردها دائمی هستند و به ندرت دستخوش تغییرات کوچک می‌شوند در حافظه ثانویه مورد استفاده قرار می‌گیرد. با وجود این، هنگامی که رشته‌ها و طول آنها غالباً در حال تغییر است، این روشِ ذخیره‌سازی معمولاً غیرکارا و ناموثر است.



(الف)



(ب)

شکل ۵ـ ۳ . ذخیره رکوردها یکی بعد از دیگری

ذخیرهٔ رشته‌ها به صورت پیوندی

امروزه از کامپیوتر به میزان زیادی در پردازش کلمات یا رشته‌ها استفاده می‌شود یعنی متن به عنوان ورودی به کامپیوتر داده می‌شود آنگاه پردازش شده و به صورت چاپ شده ٠ر خروجی ارائه می‌شود. بنابراین کامپیوتر باید توانایی تصحیح و ویرایش متن چاپ شده را نیز داشته باشد که منظور از ویرایش، حذف، تغییر و اضافه‌کردن کلمات، یا رشته‌ها عبارات، جملات و حتی پاراگراف‌های یک متن می‌باشد. با وجود این، حافظه‌های با طول ثابت که در بالا مورد بررسی قرارگرفت برای عملیات فوق‌الذکر مناسب نیستند. بنابراین، در اکثر کاربردهای پردازش کلمه، رشته‌ها با استفاده از لیستهای پیوندی ذخیره می‌شوند. لیستهای پیوندی و چگونگی حذف و اضافه‌کردن داده‌ها در آن، به‌طور مشروح در فصل ۵ بررسی