Writer : Omid Alizadeh Mohammad Hossein Ahmadi

فرترن در یک نگاه

براساس نسخه 90

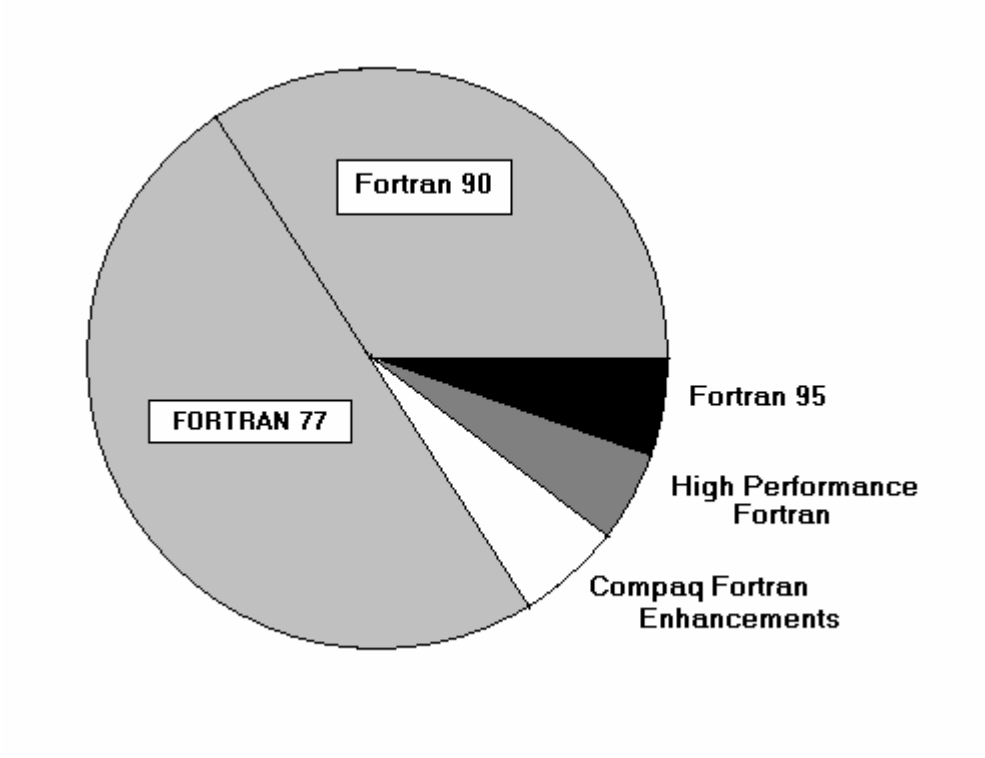
Fortran 90 Programming Examples
Including Fortran 90 CD

Omid Alizadeh
CCGroup
1/1/2009

بسم الله الرحمن الرحیم

فرترن

برنامه فرترن 95 شامل فرترن 90 و فرترن 77 می باشد . فرترن 90 نیز استانداردها و امکانات فرترن 77 را نیز در بر دارد . COMPAQ FORTRAN تمامی امکانات و ویژگی های فرترن 95 و 90 و 77 را شامل می شود و توسط MICROSOFT VISUAL به برنامه های دیگر شرکت MICROSOFT متصل می شود . نمودار زیر درصد بخش های مختلف COMPAQ VISUAL FORTRAN را مشخص می کند .



فصل اول

کلیات

برنامه نوشته شده بوسیله زبان فرترن شامل یک یا چند قسمت برنامه نویسی می باشد که از تعدادی خط دستور تشکیل شده است که شامل دستورات تعریف متغیر Variable Declaration و دستورات اجرایی Execution Statement می باشد . هر برنامه به کلمه END خاتمه می یابد . برنامه می تواند در قسمت های مختلفی نظیر بدنه اصلی ، زیربرنامه ها ، مدول ها ،بلوک اطلاعات نوشته شود هر برنامه دارای یک بدنه اصلی است و سایر قسمت ها می توانند در صورت نیاز به برنامه افزوده شوند . قسمت های مختلف برنامه می توانند به صورت جداگانه کامپایل شوند (کامپایل عمل بررسی دستورات و تبدیل آن به زبان پردازنده است) .

دستورات نوشته شده به دو دسته اصلی عبارات قابل اجرا و غیر قابل اجرا تقسیم می شوند . دستورات اجرایی عملی را جهت اجرا مشخص می کنند حال آنکه دستورات غیر اجرایی نحوه اجرای یک پردازش را تعیین می کنند . تصویر زیر موقعیت قسمت های مختلف برنامه را نمایش می دهد :

<table>
<tr><td rowspan="9">Comment Lines, INCLUDE Statements, and Directives</td><td colspan="3">OPTIONS Statements</td></tr>
<tr><td colspan="3">PROGRAM, FUNCTION, SUBROUTINE, MODULE, or BLOCK DATA Statements</td></tr>
<tr><td colspan="3">USE Statements</td></tr>
<tr><td rowspan="4">NAMELIST, FORMAT, and ENTRY Statements</td><td colspan="2">IMPLICIT NONE Statements</td></tr>
<tr><td>PARAMETER Statements</td><td>IMPLICIT Statements</td></tr>
<tr><td>PARAMETER and DATA Statements</td><td>Derived-Type Definitions, Interface Blocks, Type Declaration, Statement Function, and Specification Statements</td></tr>
<tr><td>DATA Statements</td><td>Executable Statements</td></tr>
<tr><td colspan="3">CONTAINS Statement</td></tr>
<tr><td colspan="3">Internal Subprograms or Module Subprograms</td></tr>
<tr><td colspan="4">END Statement</td></tr>
</table>

در برنامه های فرترن از کاراکتر های زیر می توان استفاده نمود :

1-ارقام 0 تا 9

2-حروف انگلیسی (بزرگ و کوچک)

3- خط فاصله _

4-حروفی که در جدول زیر قرار دارند :

Character	Name	Character	Name
blank	Blank (space)	:	Colon
=	Equal sign	!	Exclamation point
+	Plus sign	"	Quotation mark
-	Minus sign	%	Percent sign
*	Asterisk	&	Ampersand
/	Slash	;	Semicolon
(	Left parenthesis	<	Less than
)	Right parenthesis	>	Greater than
,	Comma	?	Question mark
.	Period (decimal point)	$	Dollar sign (currency symbol)
'	Apostrophe		

برنامه های فرترن به دو صورت قالب آزاد و ثابت نوشته می شوند .در قالب آزاد می توان متن را در هر قسمت دلخواه نوشت اما در قالب ثابت 5 ستون اول هر خط به برچسب (Label) اختصاص دارد و ستون ششم به علامت پیوستگی (+) اختصاص دارد و برنامه می بایست در بین ستون ها 7 تا 72 نوشته شود . علامت پیوستگی به این معناست که خط جاری در ادامه خط بالایی قرار دارد و زمانی که نتوان در یک خط کل دستور را نوشت در ستون ششم خط بعدی علامت پیوستگی را قرار می دهیم . در برنامه با قالب آزاد به جای علمت + از علامت & استفاده می شود به این ترتیب که در انتهای خط جهت پیوستگی دو خط قرار داده می شود .

تفاوت دیگری که بین این دو قالب وجود دارد این است که در قالب آزاد جهت قرار دادن توضیحات از ! استفاده می شود و در قالب ثابت از C استفاده می شود . خط توضیح خطی است که اجرا نمی شود و تنها جهت درک بهتر برنامه نوشته می شود .

عبارات ریاضی که در برنامه نویسی استفاده می شوند در جدول زیر آمده اند :

کارایی	عملگر
توان	**

ضرب	*
تقسیم	/
جمع و یا علامت مثبت	+
تفریق و یا علامت منفی	-

فرض کنید می خواهیم مقدار عبارت 2×6+2 را محاسبه کنیم . اگر در ابتدا ضرب را انجام دهیم مقدار کل عبارت فوق 14 خواهد شد و چنانچه عمل جمع در ابتدا انجام شود مقدار آن 16 خواهد بود . حال آنکه در ریاضیات هیچ یک از دو عمل جمع و ضرب نسبت به یکدیگر تقدم ندارند و برای حل این دوگانگی می توان از طراح سوال در مورد تقدم عملگرها اطلاعاتی را کسب نمود . در زبانهای برنامه نویسی نیز به محاسبه عبارتی نظیر عبارت فوق نیازمندیم اما برای حل مشکل ، یک تقدم استاندارد برای عملگرها در نظر گرفته شده است . در زبان برنامه نویسی فرترن اولویت عملگرها بصورت زیر است :

همواره اولویت با آنچه داخل پرانتز است می باشد .	*()*
توان	**
ضرب و تقسیم	/ **یا** *
جمع و منها	– **یا** +
چسباندن دو رشته	//
عبارت مقایسه ای	< **یا** > <= **یا** =< >= **یا** = = **یا** =/
نقیض گزاره شرطی	**.NOT.**
ترکیب دو گزاره	**.AND.**
یای منطقی	**.OR.**
معادل است با	**.EQV.**
معادل نیست با	**.NEQV.**

بدین ترتیب همواره ابتدا عملگری که در جدول بالاتر از دیگر عملگرها قرار دارد اعمال می شود .در مثال های زیر اولویت عملگرها با شماره در زیر عبارت مشخص شده است :

```
4 + 3 * 2 - 6/2    = 7
  ^   ^   ^  ^
  2   1   4  3

(4 + 3) * 2 - 6/2    = 11
   ^     ^   ^  ^
   1     2   4  3

(4 + 3 * 2 - 6)/2    = 2
   ^   ^   ^   ^
   2   1   3   4

((4 + 3) * 2 - 6)/2    = 4
    ^     ^   ^   ^
    1     2   3   4
```

و برای تبدیل عبارات ریاضی به فرترن نیز باید این اولویت ها را رعایت نمود به عنوان مثال برای محاسبه $\frac{5}{3+4}$ در فرترن باید نوشت 5/(3+4) .

در مورد عبارات منطقی نیز باید به ترتیب اولویت عبارات را بر اساس جدول زیر ساده کرد تا به یک نتیجه صحیح یا غلط رسید . (ستون NOT بر روی گزاره دوم اعمال می شود و بقیه ستون ها بر روی هر دو ستون اعمال می شوند)

گزاره اول	گزاره دوم	.NOT.	.AND.	.OR.	.EQV.	.NEQV.
T	T	F	T	T	T	F
T	F	T	F	T	F	T
F	T	F	F	T	F	T
F	F	T	F	F	T	F

عملگر // که در جدول اولویت ها آمده است موجب می شود که دو عبارت رشته ای به یکدیگر بچسبند و یک عبارت رشته ای بزرگتر را ایجاد می کنند به عنوان مثال خروجی سه عبارت زیر همان ' ABCDEF' است .

```
('ABC'//'DE')//'F'
'ABC'//('DE'//'F')
'ABC'//'DE'//'F'
```

در جدول زیر عملگر های دیگری که در زبان برنامه نویسی فرترن وجود دارند نوشته شده اند :

کاربرد	عملگر
کوچکتر است از	.LT. or <

کوچکتر یا مساوی است با .LE. or <=

مساوی است با .EQ. or ==

مساوی نیست با .NE. or /=

بزرگتر است از .GT. or >

بزرگتر یا مساوی است با .GE. or >=

این عملگرها به عملگرهـای مقایـسه ای معروفنـد و باعـث تولیـد .TRUE. و یـا .FALSE. مـی شوند و می توانند در بین داده های رشته ای و عددی و منطقی اعمال شوند .

مطلب دیگری که می توان در این فصل به آن اشاره کرد نتیجه عملگرهاست . به عبارت دیگرمی توان پیش از انجام عملیات توسط رایانه مشخص کرد که نتیجه عمل از چه نوعی خواهد بود . به عنوان مثـال عمل 2+2 عبارت 4 را نتیجه می دهد یا 4.

در توضیح می توان گفت که چنانچه عملگر بین دو نوع یکسان انجام شود خروجـی نیـز از همـان نـوع است یعنی در مثال بالا چون هر دو عدد از نوع صحیح هستند خروجی نیز از نوع صحیح خواهـد بـود . تنها نکته ای که در این میان است آنکه حاصل 1/2 عدد 0 مـی باشـد زیـرا هـر دو عـدد 1 و 2 از نـوع صحیح می باشند و خروجی که باید مقـدار 0.5 باشـد از نـوع حقیقـی اسـت و پـس از حـذف قـسمت اعشاری به صفر تبدیل می شود .

اگر عملگر بین دو نوع متفاوت باشد نتیجه نوع وسیع تر است . نوع عددی صحیحی زیر مجموعه اعداد حقیقی و اعداد حقیقی زیر مجموعه اعداد مختلط است . اگر مثال بنـد قبلـی را بـه شـکل 1./2 مطـرح کنیم جواب 0.5 خواهد بود .

فصل دوم

متغیر ها

در تمام زبانهای برنامه نویسی به ذخیره موقت اطلاعات نیاز داریم تا در زمان نیاز از آنها بهره مند شویم . این امکان توسط ذخیره اطلاعات در یک شی برنامه نویسی به نام متغیر فراهم شده است .متغیر محلی از حافظه است که دارای اسم و مشخصات خاص خود است و برای دسترسی به این متغیرها لازم است که یک سری ویژگی برای آنها تعریف شود . که اولین آن همان نام متغیر است . در انتخاب نام متغیر نکاتی وجود دارد که باید به آنها توجه کرد .

1 – نام متغیر نباید با عدد شروع شود .
2 – نام متغیر نباید نام توابع آماده فرترن باشد .
3 – در انتخاب نام تنها باید از کاراکترهای مجاز استفاده شود .

کاراکترهای مجاز حروف لاتین از a تا z ، A تا Z ، 0 تا 9 و _ (که underscore خوانده می شود) می باشند . البته باید توجه کرد که متغیرهایی با نام 7up (بدلیل نقض مورد 1) و Sin (به دلیل نقض مورد 3) مجاز نمی باشند اگرچه از کاراکترهای مجاز در نام آنها استفاده شده است .
حال اسامی با نام Sinx و Flor و A-B را بررسی می کنیم . نام Sinx نه تنها با عدد شروع نمی شود بلکه از کاراکترهای مجاز تشکیل شده است . حال برای بررسی شرط دوم به این نکته توجه کنید که در تابع Sin(x) قسمت اول یعنی Sin نام تابع است و متغیر Sinx نام هیچ تابعی نمی باشد پس می تواند یک نام مجاز برای متغیر به حساب بیاید .
Flor با عدد شروع نمی شود و در نام آن از کاراکترهای مجاز استفاده شده است همچنین نام متغیر نیز نمیباشد (تابع جزء صحیح Floor می باشد)بنابراین میتوان از آن برای نامگذاری یک متغیر استفاده کرد . اما A-B از کاراکتر غیر مجاز – (dash) تشکیل شده است پس نمی تواند یک متغیر باشد .
در انتخاب نام متغیر سعی کنید از اسامی تک حرفی استفاده نکنید زیرا تعداد این متغیرها به دلیل محدود بودن تعداد حروف زبان انگلیسی محدود می باشد . به عبارت دیگر در این صورت تنها می توانید متغیر های a تا z را تعریف کنید و از آنجا که فرترن به بزرگی و کوچکی حروف حساس نیست تنها می توانید از 26 متغیر استفاده کنید . همچنین سعی کنید نام متغیرها را متناسب با اطلاعات داخل آن انتخاب کنید . به عنوان مثال اگر متغیری را برای ذخیره نام دانشجو تعریف می کنید بهتر است از متغیرهایی با نام StudentName و Name استفاده کنید . در تعریف نام متغیر حروف کوچک و بزرگ تفاوتی ندارند بنابراین متغیرهای NAME و name یکی می باشند . می توان متغیر StudentName را برای خوانایی بیشتر به صورت Student_Name ویا StudentName تعریف کرد.

پس از انتخاب نام متغیر لازم است تعیین کنیم که چه نوع داده ای باید در متغیر ذخیره شود . تمام انواع داده هایی که می توانند در متغیرها ذخیره شوند در زیر آمده است :

- انواع داده
  - اعداد
    - صحیح — مانند اعداد 2 ، 3 و ... — **Integer**
    - اعشاری — مانند اعداد 2.1 یا 3.0 — **Real**
    - مختلط — مانند عدد (23 ، 12) — **Complex**
  - متن — **Character**
  - نوع منطقی — **Logical**

با توجه به تقسیم بندی فوق عدد صحیح Integer فاقد هرگونه نشانه ای است و تنها از ارقام 0 تا 9 تشکیل می شود . حال آنکه عدد حقیقی Real دارای ممیز (.) می باشد اعدادی نظیر 0.2 و 2. و .2 اعشاری می باشند . عدد مختلط Complex نیز در داخل پرانتز نگاشته می شود که عدد اول قسمت حقیقی و عدد دوم قسمت موهومی عدد است . این دو قسمت خود از نوع داده اعشاری می باشند .

برای امتحان این مطلب برنامه زیر را می نویسیم :

```
Complex CN
CN=(1,2)
Print *,CN
End
```

در این برنامه متغیر CN از نوع مختلط تعریف شده است و هر دو قسمت حقیقی و موهومی آن به صورت صحیح نوشته شده اند اما خروجی عدد (1.00000,2.00000) می باشد که این نشان می دهد بطور پیش فرض قسمتهای حقیقی و موهومی عدد مختلط Real می باشند به عبارت دیگر تمام انواع داده های عددی در این دو قسمت به نوع داده عددی صحیح تبدیل می شوند .

نوع داده ای Real قابلیت ذخیره تمامی ارقام اعشار را ندارد وبسته به سیستم عامل آنرا تا دقت مشخصی گرد می کند . برای رفع این اشکال از نوع متغیری Double Precision استفاده می کنیم . این نوع عددی که نوع خاصی از اعشاری می باشد به کاربر این امکان را می دهد تا دقت اعداد خود

را بالاتر ببرد و تعداد رقم اعشار بیشتری ذخیره گردد . در مثال زیر مقداردهی اولیه دو متغیر A و B یکسان است اما مقداری که در آنها ذخیره می گردد متفاوت است

```
REAL A
DOUBLE PRECISION B
A=4*ATAN(1.)
B=4*ATAN(1.)
PRINT *,A,B
END
```

بعد از اجرا کردن برنامه فوق مقدار زیر چاپ خواهد شد :

3.141593 3.14159274101257

دقت انواع داده ای رابطه مستقیمی با تعداد بایت های اختصاص یافته جهت ذخیره دارد به عبارت دیگر با تغییر تعداد بایت های تخصیص یافته می توان دقت اعداد را تغییر داد .برای این منظور تعداد بایت ها را با نوشتن نوع داده ضربدر تعداد بایت ها مشخص می کنیم . در زبان های برنامه نویسی BASIC نوع داده ای بایت وجود دارد که این نوع داده را می توان با نوشتن عبارت زیر ایجاد کرد :

```
INTEGER*1 A
```

در این مثال متغیر A تنها در یک بایت ذخیره می گردد و از آنجا که هر بایت از 8 بیت تشکیل شده است لذا تنها مقادیر -128 تا 127 را در خود نگه می دارد و بقیه اعداد را با بردن به این محدوده در خود ذخیره می کند . داده های صحیح می توانند در 1 یا 2 یا 4 یا 8 بایت ذخیره گردند و داده های اعشاری در 4 یا 8 بایت ذخیره می گردند که REAL*8 همان نوع داده ای DOUBLE PRESICION است و به طور پیش فرض تعداد بایت های نوع داده ای اعشاری و صحیح 4 بایت می باشد . می توان تعداد بایت ها را در داخل پرانتز نیز قرار داد به این ترتیب نوع بایت به این صورت نوشته می شود :

```
INTEGER(1) A
```

بر روی این نوع داده ها (Integer , Real , Complex) محاسبات ریاضی را می توان انجام داد .

در زبان برنامه نویسی فرترن این امکان فراهم شده است تا بتوان اعداد را در مبناهای مختلف نوشت . مبنای دو را بوسیله قرار دادن حرف B قبل از عددی که داخل علامت نقل قول قرار دارد می توان مشخص کرد . مثال های زیر اعداد در مبنای 2 را نشان می دهند این اعداد به باینری مشهورند .

```
B"1"
B’10001’
B’111001’
```

عدد B'0112' بدلیل استفاده از رقم 2 عدد باینری نمی باشد . اعداد در مبنای 8 را توسط حرف O و اعداد در مبنای 16 را با حرف Z نمایش می دهیم .

جدول زیر مقدار اعداد در مبناهای مختلف را نمایش می دهد :

مبناي 16	مبناي 2	مبناي 8	ميناي 10
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
A	1010	12	10
B	1011	13	11
C	1100	14	12
D	1101	15	13
E	1110	16	14
F	1111	17	15

نوع رشته ای که همان متن یا text می باشد دارای نشانه های " یا ' است . اگر متن حاوی یکی از علایم گفته شده باشد برای مشخص کردن نوع رشته ای از علامت دوم استفاده می شود به عنوان مثال چنانچه بخواهیم عبارت "Hello": Ali را در یک متغیر رشته ای قرار دهیم باید از علامت ' استفاده کنیم زیرا علامت " در خود متن استفاده شده است . پس نتیجه عمل '"Hello": Ali' خواهد بود .

بر روی این نوع داده کارهایی از قبیل حذف قسمتی از متن ، برش قسمتی از متن ، کاراکتریزه کردن و ... را می توان انجام داد . اما اعمال زیاضی از قبیل سینوس و کسینوس ، ضرب و ... را نمی توان انجام داد .

نوع منطقی Logical نتیجه صحیح یا غلط بودن یک سری پردازش را در خود نگه می دارد که تنها شامل دو مقدار .True. یا .False. می باشد . این نوع متغیر معادل گزاره های منطقی ریاضی می باشد . به عنوان مثال به شخص گفته می شود که جزوه فرترن در دست اوست ، او پس از نگاه کردن به جزوه تشخیص می دهد که آیا این گزاره درست است یا نه و نتیجه این پردازش ها در قالب یک کلمه بله یا نه نمود پیدا میکند که معادل .True. یا .False. در فرترن می باشد . زبان برنامه نویسی فرترن هم چنین کاری را درقبال عبارت های منطقی انجام میدهد .

نوع متغیری Type متشکل از تمام موارد بالاست که در ادامه مطلب به آن اشاره خواهد شد .

پس از تعیین تمام موارد بالا نوبت به نحوه تعریف متغیر ها می رسد .برای این منظور از دستور خلاصه شده زیر استفاده می کنیم :

نوع متغیر 1 نام متغیر ، 2 نام متغیر ، ...

به عنوان مثال برای معرفی متغیری به نام text1 از نوع رشته ای از دستور زیر استفاده می شود :

```
Character text1
```

همانطور که در دستور نوشته شده است می توان در یک خط تعریف ، بیش از یک متغیر را تعریف نمود . در خط دستور زیر سه نوع متغیر از نوع صحیح تعریف شده اند :

```
INTEGER A,B,C
```

از آنجا که فرترن جهت انجام محاسبات ریاضی طراحی شده است لذا از انواع داده های یاد شده ، بیشتر از اعداد استفاده می شود و دربین اعداد از اعداد اعشاری و صحیح بیشتر از اعداد مختلط استفاده می شود به این دلیل متغیرهایی که بدون تعریف استفاده شوند در صورتی که با حروف I تا N شروع شوند بطور پیش فرض بصورت صحیح تعریف می شوند و چنانچه با غیر از این حروف آغاز شوند بصورت حقیقی تعریف می شوند .این پیش فرض ها را می توان توسط دستور IMPLICIT NONE که در اول برنامه نوشته می شود حذف کرد و حتی می توان پیش فرض های جدیدی را توسط دستور IMPLICIT به برنامه افزود . تعریف کلی این دستور به صورت زیر است :

نوع متغیر Implicit ،1 سرنام ، 2 سرنام و ...

این دستور به این معنی است که متغیرهایی که با این حرف شروع می شوند از نوع نوشته شده اند . بنابراین عبارت زیر موجب می شود که متغیر هایی که با حروف C و T شروع می شوند در صورت عدم تعریف از نوع رشته ای در نظر گرفته شوند :

```
IMPLICIT CHARACTER T,C
```

پیش فرض کلی فرترن نیز به این صورت قابل نوشتن است :

```
Implicit integer ( I-N)
Implicit Real (A-H) , ( O-Z)
```

استفاده از خط فاصله به این معنی است که حروف I تا N را به متغیر های صحیح و حروف A تا H و O تا Z را به متغیر های اعشاری نسبت دهد .

در ریاضیات عباراتی مانند x_1 ، x_2 ، ... ، x_n یعنی n متغیر x داریم که برای سهولت کار از اندیس استفاده کردیم از آنجا که فرترن بیشتر جنبه های ریاضیات را دارد می توان متغیر با اندیس تعریف کرد برای تعریف n متغیر با اندیس از دستور زیر استفاده می کنیم :

... ، (تعداد اندیس) نام متغیر 2 ، (تعداد اندیس) نام متغیر 1 نوع متغیر

وجود نام متغیر در ابتدای خط موجب می شود عددی که در داخل پرانتز نوشته می شود به عنوان تعداد اندیس باشد . در مثال زیر 100 متغیر اندیس دار x که از نوع صحیح می باشد تعریف شده است :

```
Integer x(100)
```

چنانچه نوع متغیر ذکر نشود منظور اندیس مشخصی از آن متغیر است . به عنوان مثال در نمونه زیر، در پنجمین x عدد 23 نوشته می شود :

```
X(5)=23
```

در ماتریس ها و در فیزیک از اندیس دو بعدی استفاده می شود . در فرترن نیز می توان از اندیس چند بعدی استفاده کرد برای این منظور از دستور زیر استفاده می کنیم

... ، (... , تعداد بعد 2 , تعداد بعد 1) نام متغیر نوع متغیر

دستور زیر 100 متغیر با اندیس دو بعدی را تعریف می کند (یک ماتریس 4×25)

```
Real A(25,4)
```

در مثال بالا بعد اول دارای 25 اندیس و بعد دوم دارای 4 اندیس می باشد ، چنانچه تعداد اندیس ها مشخص نباشد از : استفاده می کنیم حال دستور بالا را این چنین تغییر می دهیم :

```
Real A(:,4)
```

این بدان معنی است که متغیر A دارای دو بعد که تعداد اندیس اول نامشخص و تعداد اندیس دوم 4 است که از شماره 1 شروع و به شماره 4 ختم می شود . چنانچه بخواهیم کران پایین و بالای متغیرها را تغییر دهیم از دستور زیر استفاده می کنیم :

... ، (... , کران بالای بعد 1 : کران پایین بعد 1) نام متغیر نوع متغیر

مثال بالا را دوباره تغییر می دهیم :

```
Real A(:,-1:2)
```

اینک بعد دوم این دستور دارای 4 اندیس است اما شروع اندیس ها از عدد 1- است .

برای مقدار دهی ابعادی که تعداد نامشخص دارند (متغیرهایی که در تعریف آنها از : استفاده شده است) از تابع Allocate استفاده می کنیم . در مثال زیر ابتدا یک متغیر سه بعدی تعریف و سپس تعداد اندیس ها را مشخص کرده ایم :

```
Real Allocatable :: A(:,:,3)
Allocate (A(11:12,4,3))
```

در این مثال بعد از استفاده از دستور Allocate بعد اول دارای 2 اندیس با شروع از 11 و بعد دوم دارای 4 اندیس با شروع از 1 (به دلیل عدم تعریف) تعریف شده است . استفاده مجدد از دستور ALLOCATE برای یک متغیر امکان پذیر نیست و برنامه با خطای در حین اجرا مواجه می شود .

در برنامه زیر دوبار از دستور ALLOCATEبرای یک متغیر استفاده شده است که برنامه پیغام خطا می دهد :

```
INTEGER*4, ALLOCATABLE:: A(:)
ALLOCATE(A(1))
ALLOCATE(A(2))
END
```

همچنین استفاده از دستور ALLOCATE برای متغیری که تمامی ابعاد مشخص می باشند نیز امکان پذیر نمی باشد و برنامه اجرا نخواهد شد .

برای تشخیص اینکه آیا متغیری تمامی ابعاد آن مشخص شده است از تابع ALLOCATED استفاده می کنیم . در صورتی که متغیر از پیش مشخص شده باشد مقدار آن.TRUE. و در غیر اینصورت مقدار آن .FALSE. خواهد بود .

در برنامه نویسی گاهی اوقات لازم است ویژگی های مختلف را به متغیرها نسبت داد بنابراین باید ابتدا این ویژگی ها را شناخت و سپس از آنها استفاده کرد . اکنون به بررسی این ویژگی ها می پردازیم :

Allocatable–

اگر در تعریف متغیر ها تعداد اندیس ها را مشخص نکردیم باید از ویژگی Allocatable برای امکان استفاده از Allocate استفاده کنیم . به عبارت دیگر در سری Visual Fortran 6.5 و غیره تابع Allocate تنها قادر است به متغیرهایی که دارای ویژگی Allocate هستند اندیس اختصاص دهد .

Dimension –

از این ویژگی برای دادن بعد به متغیرها استفاده می شود به عنوان مثال در دستور تعریف متغیر زیر به ترتیب به سه متغیر A و B و C ابعاد (2,2) داده می شود . :

```
Real ,Dimension (2,2) ::A,B,C
```

این دستور معادل دستور زیر است :

```
Real A(2,2) , B(2,2) , C(2,2)
```

Parameter -

از این ویژگی برای ثابت نگه داشتن یک مقدار در متغیر استفاده می شود به صورتی که تا آخر برنامه مقدار آن ثابت خواهد ماند . به عنوان مثال در برنامه زیر برای جلوگیری از نوشتن عبارت 3.141592 آن را در متغیر p ذخیره و برای ثابت نگه داشتن آن در کل برنامه از این ویژگی استفاده می کنیم :

```
Double Precision , Parameter :: P= 3.141592
Read *,R
Print *,"Area = " , p*R**2
Print *, "S = " , 2*p*R
End
```

حال آنکه با ویژگی ها آشنا شدیم می بایست آنها رادر دستور تعریف متغیر بکار بریم . برای تعریف متغیرها همراه با ویژگی ها از دو نوع entity – oriented و attribute – oriented استفاده می کنیم

در دستور entity – oriented داریم :

نوع متغیر 1 ، ویژگی 2 ، ... :: نام متغیر 1 ، نام متغیر 2 ، ...

Real Allocatable , Dimension (:) :: A,B

در دستور attribute – oriented داریم :

نوع متغیر 1 ، نام متغیر 2 ، ...

ویژگی 1 ، نام متغیر 1 ، نام متغیر 2 ، ...

ویژگی 2 ، نام متغیر 1 ، نام متغیر 2 ، ...

...

```
Real A,B
Dimension (:) A,B
```

```
Allocatable A,B
```

تنها تفاوت موجود بین هر دو نوع تعریف این است که چنانچه بخواهیم تمامی ویژگی ها را به تمامی متغیرها اعمال کنیم از حالت اول و اگر بخواهیم ویژگی های متفاوت را به متغیرهای متفاوت اعمال کنیم از حالت دوم استفاده می کنیم .

علاوه بر ویژگی های گفته شده ویژگی های دیگری که با نحوه متفاوتی اعمال می شوند نیز وجود دارند . یکی از این ویژگی ها دستورات Kind هستند .

در مورد نوع صحیح به عنوان مثال به صورت زیر تعریف می شوند :

```
Integer (Selected_INT_Kind(3)) A
```

یعنی عدد A از نوع صحیح و در بازه -10^3 تا 10^3 می باشد . در مورد عدد اعشاری باید دو مقدار را وارد کرد که عدد اول حداقل رقم اعشار و عدد دوم بازه عدد است . به عنوان مثال :

```
Real (Selected_Real_Kind(3,4))B
```

یعنی عددB از نوع صحیح با حداقل 3 رقم اعشار و در بازه 10^{-4} تا 10^4 می باشد .

می توان برای خلاصه تر شدن Selected_INT_Kind و Selected_Real_Kind را حذف کرد و به صورت زیر نوشت :

```
Integer (3) A
Real (3,4) B
```

در مورد متغیر رشته ای می توان طول رشته را تعیین کرد برای این منظور از یکی از سه دستور زیر استفاده می کنیم :

Character (Len=n) 1 نام متغیر ، ...

Character (n) 1 نام متغیر ، ...

Character * n 1 نام متغیر ، ...

که n طول رشته است . در صورتی که از این تعریف استفاده نشود طول به طور پیش فرض 1 در نظر گرفته می شود . چنانچه در متغیری با n حرف تعداد m حرف نوشته شود اگر m>n باشد n حرف اول در متغیر ذخیره می شود . و در غیر اینصورت قسمت خالی رشته با Space پر می شود .

```
Character *4 C1
```

```
C1="Hello"
```

در این مثال چهار حرف اول Hello یعنی Hell ذخیره می شود .

```
Character *6 C1
C1="Hello"
```

ودر این مثال مقدار " Hello" در متغیر ذخیره می شود . توجه کنید که یک فاصله خالی (Space) بعد از کلمه وجود دارد .

فصل سوم

توابع آماده فرترن

در برنامه نویسی برای انجام تعدادی از کارهای معمول ، از توابع آماده استفاده می شود . توابع در حالت کلی از یک نام و یک یا چند آرگومان (ورودی) تشکیل می شوند .

نام تابع (ورودی 1 ، ورودی 2 ، ...)

آنچه که باید در مورد توابع بدانیم این است که تابع چه کاری را انجام می دهد و برای انجام این کار از چه نوع ورودی استفاده می کند و خروجی تابع چیست . جهت بررسی این موارد جدول زیر طراحی شده است :

توضیحات	ورودی	نام تابع
قدر مطلق	اعشاری Real	ABS (x)
قدر مطلق	مختلط Complex	CABS (x)
قدر مطلق	دقت مضاعف	DABS (x)
قدر مطلق	صحیح Integer	IABS (x)
آرک کسینوس	اعشاری Real	ACOS (x)
آرک کسینوس	دقت مضاعف	DACOS (x)
حذف قسمت اعشاری	اعشاری Real	AINT (x)
حذف قسمت اعشاری	دقت مضاعف	DINT (x)
آرک سینوس	اعشاری Real	ASIN (x)
آرک سینوس	دقت مضاعف	DSIN (x)
آرک تانژانت	اعشاری Real	ATAN (x)
آرک تانژانت	دقت مضاعف	DTAN (x)
آرک تانژانت	اعشاری Real	ATAN2 (x)
آرک تانژانت	دقت مضاعف	DTAN2 (x)
حرف مطابق با جدول Ascii	صحیح Integer	CHAR (x)
کسینوس	اعشاری Real	COS (x)
کسینوس	مختلط Complex	CCOS (x)
کسینوس	دقت مضاعف	DCOS (x)
مزدوج عدد مختلط	مختلط Complex	CONJ (x)
کسینوس هیپربولیک	اعشاری Real	COSH (x)
کسینوس هیپربولیک	دقت مضاعف	DCOSH (x)

تفاضل در صورت مثبت بودن	اعشاری Real	DIM (x,y)
تفاضل در صورت مثبت بودن	صحیح Integer	IDIM (x,y)
تولید عدد با دقت بیشتر	اعشاری Real	DPROD (x,y)
e بتوان عدد	اعشاری Real	EXP (x)
e بتوان عدد	مختلط Complex	CEXP (x)
e بتوان عدد	دقت مضاعف	DEXP (x)
کد اسکی مربوط به رشته	رشته ای	ICHAR (x)
جستجو در میان رشته	رشته ای	INDEX (String,Substring)
تبدیل عدد به صحیح	اعشاری Real	INT (x)
تبدیل عدد به صحیح	اعشاری Real	IFIX (x)
تبدیل عدد به صحیح	دقت مضاعف	IDINT (x)
طول رشته	رشته ای	LEN(String)
لگاریتم در پایه طبیعی	اعشاری Real	LOG (x)
لگاریتم در پایه طبیعی	اعشاری Real	ALOG (x)
لگاریتم در پایه طبیعی	مختلط Complex	CLOG (x)
لگاریتم در پایه طبیعی	دقت مضاعف	DLOG (x)
لگاریتم در پایه ده	اعشاری Real	LOG10 (x)
لگاریتم در پایه ده	اعشاری Real	ALOG10 (x)
لگاریتم در پایه ده	دقت مضاعف	DLOG10 (x)
ماکزیمم اعداد	اعشاری Real	MAX (x,y,…)
ماکزیمم اعداد	صحیح Integer	MAX0 (x,y,…)
ماکزیمم اعداد	اعشاری Real	AMAX1 (x,y,…)
ماکزیمم اعداد	دقت مضاعف	DMAX1 (x,y,…)
ماکزیمم اعداد	اعشاری Real	MAX1 (x,y,…)
ماکزیمم اعداد	صحیح Integer	AMAX0 (x,y,…)
مینیموم اعداد	اعشاری Real	MIN (x,y,…)
مینیموم اعداد	صحیح Integer	MIN0 (x,y,…)
مینیموم اعداد	اعشاری Real	AMIN1 (x,y,…)
مینیموم اعداد	دقت مضاعف	DMIN1 (x,y,…)

مینیموم اعداد	اعشاری Real	MIN1 (x,y,…)
مینیموم اعداد	صحیح Integer	AMIN0 (x,y,…)
باقیمانده	صحیح Integer	MOD (x,y)
باقیمانده	اعشاری Real	AMOD (x,y)
باقیمانده	دقت مضاعف	DMOD (x,y)
تبدیل عدد به اعشاری	صحیح Integer	REAL (x)
تبدیل عدد به اعشاری	صحیح Integer	FLOAT (x)
تبدیل عدد به اعشاری	دقت مضاعف	SNGL (x)
مفدار x به همراه علامت y	اعشاری Real	SIGN (x,y)
مفدار x به همراه علامت y	دقت مضاعف	DSIGN (x,y)
مفدار x به همراه علامت y	صحیح Integer	ISIGN (x,y)
سینوس	اعشاری Real	SIN (x)
سینوس	مختلط Complex	CSIN (x)
سینوس	دقت مضاعف	DSIN (x)
سینوس هیپربولیک	اعشاری Real	SINH (x)
سینوس هیپربولیک	دقت مضاعف	DSINH (x)
جذر	اعشاری Real	SQRT (x)
جذر	مختلط Complex	CSQRT (x)
جذر	دقت مضاعف	DSQRT (x)
تانژانت	اعشاری Real	TAN (x)
تانژانت	دقت مضاعف	DTAN (x)
تانژانت هیپربولیک	اعشاری Real	TANH (x)
تانژانت هیپربولیک	دقت مضاعف	DTANH (x)
قرار دادن فاصله های اول در آخر رشته	رشته ای	ADJUSTL (String)
قرار دادن فاصله های آخر در اول رشته	رشته ای	ADJUSTR (String)
حذف فواصل خالی آخر رشته	رشته ای	TRIM (String)
طول متن بدون فاصله خالی در آخر	رشته ای	LEN_TRIM (String)

فصل چهارم

کنترل اجرای برنامه

برای نوشتن برنامه های کاربردی علاوه بر استفاده از متغیرها و توابع می بایست از دستورات متفاوتی که در این بخش توضیح داده می شود استفاده کنیم .

- دستور و ساختار شرط

چنانچه در برنامه نویسی بخواهیم در صورت برقراری شرطی اتفاقی بیفتد (نیفتد) از ساختار یا دستور شرط استفاده می کنیم . دستور شرط به صورت زیر تعریف می شود :

IF (عبارت شرطی) یک دستور

فرترن ابتدا عبارت شرطی را محاسبه و به یکی از دو حالت صحیح یا غلط می رسد چنانچه عبارت شرطی درست باشد ، دستور داده شده اجرا می شود . باید توجه داشت که تنها یک دستور می توان نوشت .

چنانچه بخواهیم بیش از یک دستور را در یک شرط اعمال کنیم و یا اینکه شروط مختلفی را با دستورات مختلف اعمال کنیم می توانیم به شکل زیر از ساختار شرط استفاده کنیم :

IF (عبارت شرطی 1) Then

بلوک دستورات

Else IF (عبارت شرطی 2)Then

بلوک دستورات

.

.

.

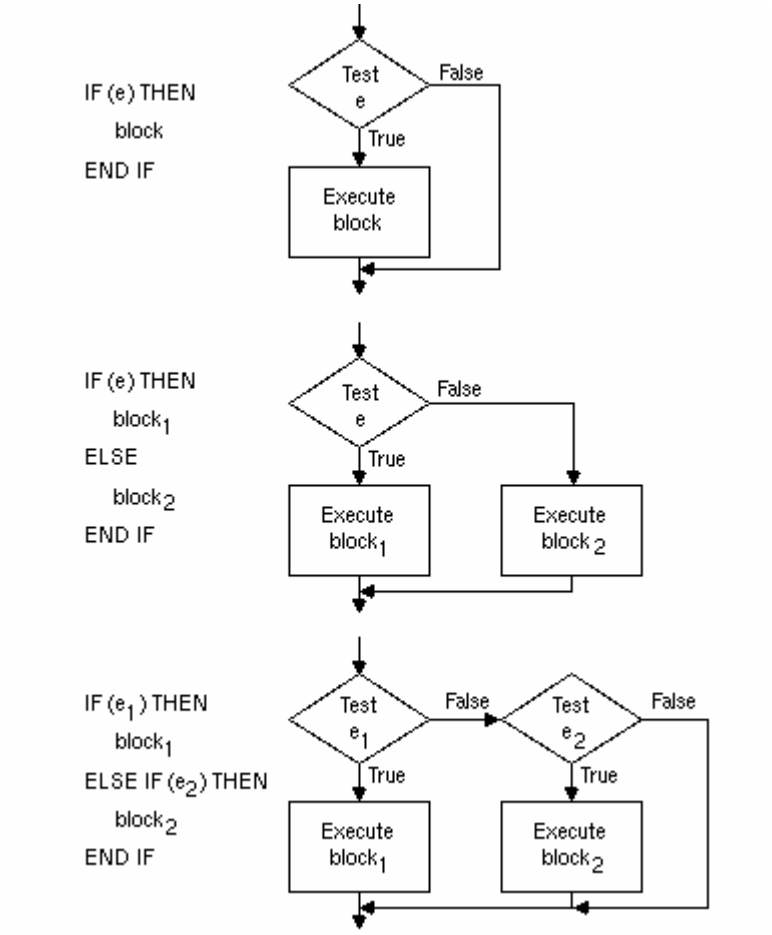
Else

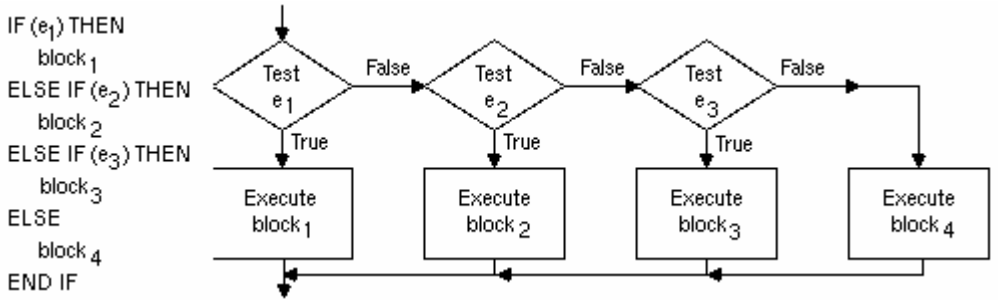
بلوک دستورات

End IF

در ساختار شرط قسمتهای Else IF و Else کاملا اختیاری می باشند اما چنانچه ELSE IF نوشته شود می بایست از کلمه Then استفاده شود . استفاده از End IF برای پایان ساختار الزامی است . در توضیح ساختار شرط می توان گفت که پس از بررسی شرط اول ، چنانچه درست باشد بلوک دستورات مربوط به آن را اجرا می کند و در غیر اینصورت شرط بعدی را چک میکند . چنانچه هیچ یک از دستورات بالا اجرا نشود و قسمت Else در ساختار شرط آمده باشد بلوک دستوری مربوط به Else اجرا خواهد شد

در واقع Else دارای عبارت شرطی معادل با ترکیب نقیض شروط بالاست .نماگرد دستور شرط در تصاویر زیر آمده است .





در برنامه زیر چنانچه عدد وارد شده یک یا دو رقمی باشد عبارت مناسب چاپ می شود :

```
Read *,I
If ( I>=0 .AND. I<10 ) Then
Print *,"Your Number has one digit"
Else If ( I<100 .AND. I>9 ) Then
Print *,"Your Number has two digits"
Else
Print *,"Your Number has more than two digits"
End If
PAUSE
End
```

برنامه زیر سینوس یک زاویه که بر حسب رادیان است را خوانده و تانژانت آنرا نمایش می دهد .

```
Read *,A
If ( A<=1 .And. A>=-1) Then
Ang=Asin(A)
Print *,Ang*180/3.141592,Tan(Ang)
End If
End
```

- ساختار انتخاب

این ساختار حالت خاصی از ساختار شرط است . حالت کلی این ساختار به شرح زیر است

Select Case (عبارت مورد نظر)

Case (حالت اول عبارت)

بلوک دستورات

Case (حالت دوم عبارت)

بلوک دستورات

.

.

.

Case Default

بلوک دستورات

End Select

این ساختار معادل دستورات شرط زیر است :

If (عبارت شرطی == حالت اول) Then

بلوک دستورات

Else If (عبارت شرطی == حالت دوم) Then

بلوک دستورات

.

.

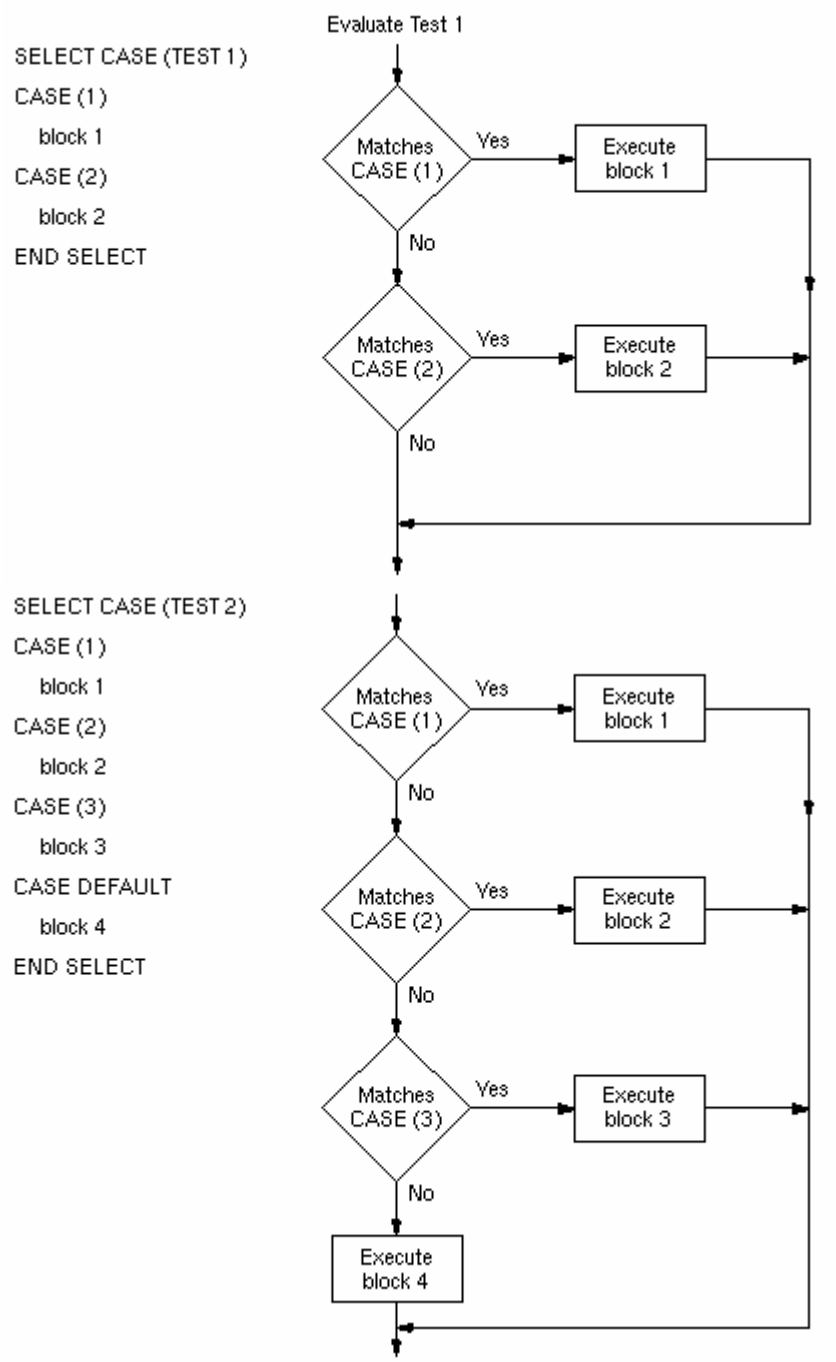
.

Else

بلوک دستورات

End IF

همانگونه که از مقایسه دو ساختار بالا بدست می آید بلوک دستورات زمانی اجرا می شود که عبارت مورد نظر یکی از حالات ذکر شده باشد و Case default معادل Else در ساختار شرط است یعنی اگر هیچ کدام از بلوک های دستور اجرا نشوند بلوک دستوری این قسمت اجرا خواهد شد . در تصاویر زیر نماگرد دستور ترسیم شده است .



باید توجه داشته باشید که حالات ذکر شده باید از نوع عبارت مورد نظر باشند و نباید اشتراک داشته باشند . عبارت مورد نظر ساختار انتخاب تنها یکی از سه حالت Integer ، Character ، Logical می باشد .

در دو برنامه زیر چنانچه عدد یک وارد شود برنامه عدد 100 و در غیر اینصورت عدد 0 را چاپ خواهد کرد

برنامه اول : استفاده از نوع صحیح

```
Integer A
Read *,A
Select Case (A)
Case (1)
Print *,"100"
Case default
Print *,"0"
End select
End
```

برنامه دوم : استفاده از نوع منطقی

```
Logical L1
Integer A
Read *,A
L1=(A==1)
Select Case (L1)
Case (.True.)
Print *,"100"
Case (.False.)
Print *,"0"
End select
End
```

برای ایجاد گستره انتخاب از : استفاده می شود به این صورت که n:m یعنی کلیه اعداد صحیح از عدد n تا عدد m (با احتساب n و m) و :n یعنی تمامی اعداد صحیح بزرگتر مساوی عدد n و m: یعنی تمامی اعداد کوچکتر یا مساوی عدد m

از : می توان برای کاراکترها نیز استفاده کرد . در برنامه زیر یک حرف از کاربر گرفته می شود و نوع آن مشخص می شود .

```
Character *1 C1
Read *,C1
Select Case (C1)
          Case ("a":"z","A":"Z")
     Print *,"Alphabetic"
```

```
            Case ( "0":"9")
            Print *,"Number"
 Case Default
       Print *,"Other"
End Select
End
```

از برنامه بالا می توان فهمید که در مقابل Case مـی تـوان بـیش از یـک حالـت را قـرار داد . همچنـین حالت عددی از نوع کاراکتر می باشد و هیچ اشتراکی بین حالات نیست .

- دستور پرش ساده

اگر بخواهیم ادامه اجرای برنامه را از سطر خاصی شروع کنیم می توانیم از دستور پرش استفاده کنیم :

Goto Label

که Label یک عدد می باشد که در ابتدای خط مقصد نوشته می شود . استفاده از دستور پرش بـدون اختصاص Label موجب نمایش پیغام خطا خواهد شد . در ادامه مثالی از این مطلب آورده خواهد شد

- دستور پرش محاسباتی

Goto (Label1 , Label2 , ...) عدد

این دستور ابتدا عدد مورد نظررا محاسبه کرده و چنانچه مقدار آن یک باشـد بـه عـددی کـه بـه عنـوان label 1 نوشته شده است پرش می کند و ... اگر عدد مورد نظر منفی و یا دارای اعشار باشد و یا هـیچ Label به آن اختصاص داده نشده باشد ، دستور کار خاصی انجام نمی دهد .

برنامه زیر ریشه های معادله درجه دو را تنهـا بـا اسـتفاده از دسـتور پـرش و دسـتور پـرش محاسـباتی ، محاسبه می کند :

```
Read *,A,B,C
Delta=b**2-4*A*C
Goto ( 10 , 20 ) Floor(Delta/(ABS(Delta)+1))+2
10 Print *,"there is no root "
Goto 30
20 Print *,"x1=",(-B+Sqrt(Delta))/2/A
Print *,"x2=",(-B-Sqrt(Delta))/2/A
30 End
```

- دستور ادامه

این دستور به صورت Continue نوشته می شود و گاهی اوقات به همراه یک Label در ابتدای خط می آید . اجرای این دستور باعث ادامه پردازش به خط بعدی می شود و کـاربرد خاصـی در ایـن زمینـه ندارد . از این دستور می توان برای خاتمه حلقه نیز استفاده کرد .

- دستور توقف

از این دستور برای توقف عملیات اجرای برنامه و خاتمه برنامه استفاده می شود . این دستور به صورت زیر تعریف می شود :

Stop [Stop-expresion]

می توانیم دستور Stop را تنها به کار برد و در صورتی که بخواهیم عبارتی را برای بستن برنامه تحت عنوان handle اختصاص دهیم می توانیم یک رشته و یا یک عدد را قرار دهیم . در صورت به کار بردن رشته عدد صفر منظور می شود .

- ساختار گردشی - حلقه

چنانچه بخواهیم یک عمل را N بار انجام دهیم یا N متغیر که با هم تصاعد عددی دارند داشته باشیم از ساختار حلقه استفاده می کنیم .حلقه ها بر اساس N به سه دسته تقسیم می شوند

- حلقه نا محدود

[نام حلقه :] Do

[بلوک دستورات]

End Do [نام حلقه]

[نام حلقه :] Do label

[بلوک دستورات]

Label Continue

- حلقه محدود شرطی

[نام حلقه :] Do [Label] [,] While (عبارت شرطی)

[بلوک دستورات]

End Do [نام حلقه]

[نام حلقه :] Do Label [,] While (عبارت شرطی)

[بلوک دستورات]

Label Continue

- حلقه شمارشی

[نام حلقه :] Do [Label] نام متغیر = کران بالا ، کران پایین [، گام]

[بلوک دستورات]

End Do [نام حلقه]

[نام حلقه :] Do Label نام متغیر = کران بالا ، کران پایین [، گام]

[بلوک دستورات]

Label Continue

در برنامه زیر با استفاده از حلقه شمارشی N! محاسبه می شود :

```
Read *,N
Factoriel=1
Do i=1,N
 Factoriel=Factoriel*I
End Do
Print *,N,"!=",Factoriel
End
```

در برنامه زیر مقدار $\sum_{n=0}^{100}\frac{2^n}{n!}$ محاسبه و چاپ می شود :

```
F=1 ; Sum=1
Do i= 1,100
        F=F*I
  Sum=Sum+(2**i)/F
End Do
Print *,Sum
End
```

- دستور خروج از حلقه

دستور Exit برای خروج از حلقه ای است که خود دستور در آن قرار دارد . چنانچه در مقابل این دستور نام حلقه ای ذکر شود ، دستور خروج برای آن حلقه اجرا خواهد شد .

- دستور گردش حلقه

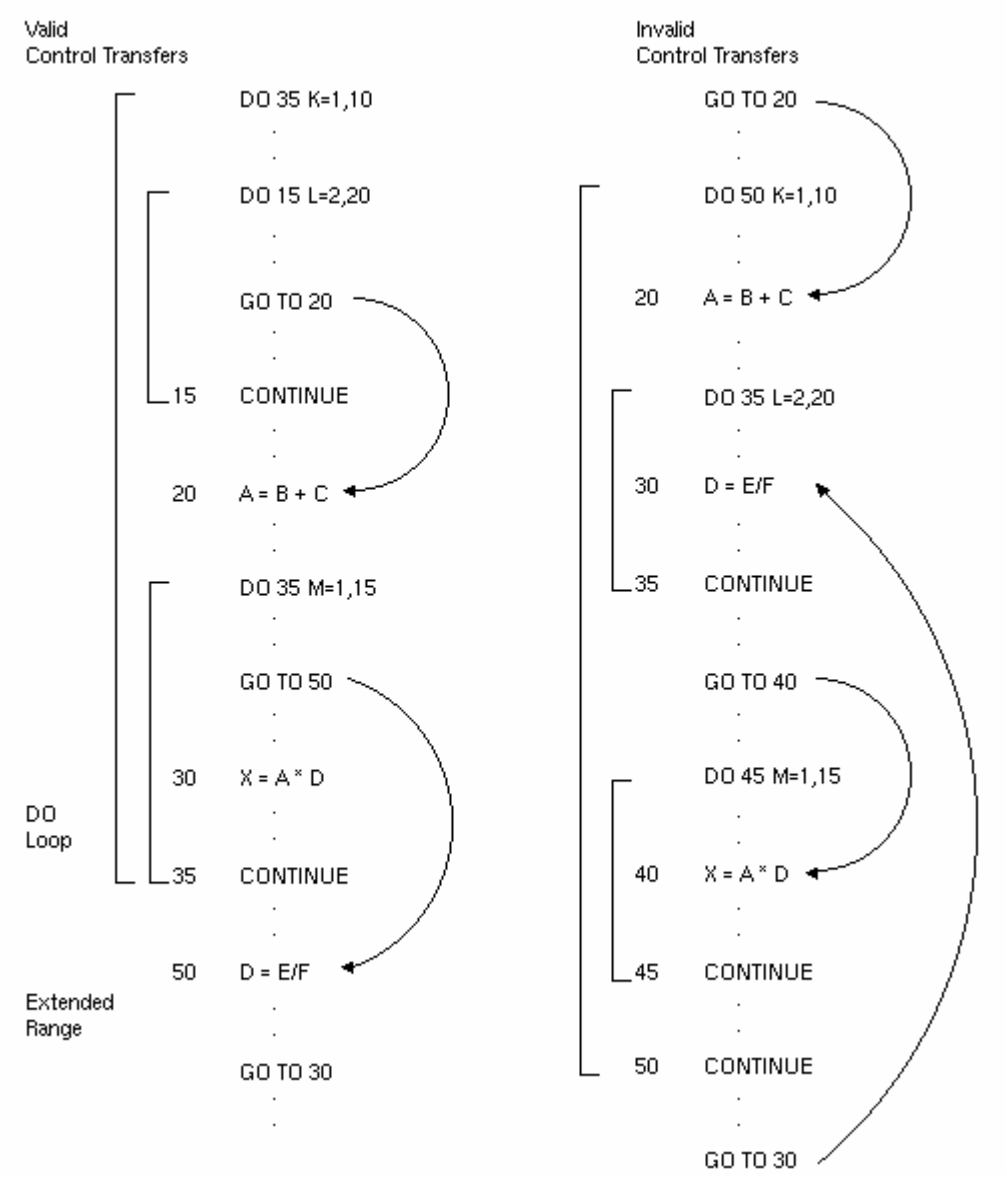
چنانچه فرترن در اجرای برنامه به دستور Cycle برسد دستورات بین Cycle و اولین End Do انجام نخواهد شد .

در برنامه زیر عدد 1 را بر اعداد 100- تا 100 تقسیم کرده ایم و خروجی آن مقدار این عبارت است . همانطور که می دانید تقسیم بر صفر معنی ندارد و برای انجام ندادن تقسیم بر صفر از دستور Cycle استفاده کرده ایم :

```
Do i=-100 , 100
      IF ( i==0) Cycle
           Print *,1./i.
End Do
End
```

در شکل زیر نحوه استفاده از ساختارهای گردش مرکب (تو در تو) نشان داده شده است . ستون سمت راست استفاده نادرست و ستون سمت چپ استفاده درست را نمایش می دهد .

Correctly Nested DO Loops	Incorrectly Nested DO loops
DO 45 K=1,10	DO 15 K=1,10
.	.
.	.
DO 35 L=2,50,2	DO 25 L=1,20
.	.
.	.
35 CONTINUE	15 CONTINUE
.	.
.	.
DO 45 M=1,20	DO 30 M=1,15
.	.
.	.
45 CONTINUE	25 CONTINUE
.	.
.	.
DO 10 I=1,20	30 CONTINUE
.	.
.	.
DO J=1,5	DO 10 I=1,5
.	.
.	.
DO K=1,10	DO J=1,10
.	.
.	.
END DO	10 CONTINUE
.	.
.	.
END DO	END DO
.	
.	
10 CONTINUE	


Valid
Control Transfers
DO 35 K=1,10
DO 15 L=2,20
GO TO 20
15 CONTINUE
20 A = B + C
DO 35 M=1,15
GO TO 50
30 X = A * D
35 CONTINUE
50 D = E/F
GO TO 30
DO
Loop
Extended
Range
Invalid
Control Transfers
GO TO 20
DO 50 K=1,10
20 A = B + C
DO 35 L=2,20
30 D = E/F
35 CONTINUE
GO TO 40
DO 45 M=1,15
40 X = A * D
45 CONTINUE
50 CONTINUE
GO TO 30

فصل پنجم

خواندن و نوشتن

(بهتر است مطالب این بخش را همزمان با مطالب بخش دسترسی به فایل بخوانید .)

برای خواندن اطلاعات از یکی از سه دستور زیر استفاده می شود :

Read (ویژگی های کنترلی خواندن) 1 نام متغیر ، ...

Read قالب ، 1 نام متغیر ، ...

Write (ویژگی های کنترلی خواندن) 1 نام متغیر ، ...

منظور از ویژگی های کنترلی افزودن مشخصات و خصوصیات جدید به عمل خواندن است . در جدول زیر این ویژگی ها آورده شده اند :

کلید واژه	مقدار ویژگی	توضیحات
[unit=]	عدد	عددی که مشخص کننده واحد ورود اطلاعات است . به عبارت دیگر یک اشاره گر از محل ورود اطلاعات می باشد . * نماد ورودی استاندارد است . چنانچه در جایگاه اول داخل پرانتز قرار گیرد نوشتن کلیدواژه لازم نیست .
[FMT=]	قالب	مشخص کننده نحوه خواندن و قرار دادن ورودی در متغیرهاست . * نماد فرمت آزاد یا بدون قالب است . چنانچه در جایگاه دوم داخل پرانتز قرار گیرد نوشتن کلید واژه لازم نیست
Advance	“yes” یا “no”	چنانچه yes باشد در خواندن اطلاعات با توجه به قالب در طول داده به جلو حرکت می کند . در غیر اینصورت پس از خواندن اولین متغیر از بین داده ها برای خواندن اطلاعات بعدی از اول داده شروع می کند
END	Label	چنانچه در خواندن از فایل به آخر آن برسد به Label گفته شده پرش می کند
EOR	Label	چنانچه در خواندن از فایل به آخر رکورد برسد به Label گفته شده پرش می کند .
ERR	Label	چنانچه در حین عملیات خواندن یا نوشتن به خطایی برخورد کند به Label گفته شده پرش می کند .

منظور از رکورد در مطالب بالا یک خط از فایل می باشد .

می توان یک حلقه را نیز در ساختارهای بالا اعمال کرد :

Read (ویژگی های کنترلی خواندن) (...)

... ، (... ، (گام ، کران بالا ، کران پایین = شمارنده 2 ،(گام ، کران بالا ، کران پایین =شمارنده 1،نام متغیر

به مثال زیر توجه کنید :

Read (*,*)((A(i,j),i=1,10),j=1,5))

این دستور معادل دستور زیر است :

```
Do 1 i=1,10
     Do 1  j=1,5
          Read (*,*) A(i,j)
1 Continue
```

فصل ششم

قالب بندی (فرمت)

بخش عظیمی از برنامه نویسی به خواندن و چاپ اطلاعات اختصاص دارد . بنابراین این مهم ایجاب می کند تا بتوان خواندن و نوشتن را سفارشی کرد به عبارت ساده تر بتوان حالات خاصی را به این دو دستور داد از جمله این حالات می توان به اختصاص میدانها و قالب بندی صفحه اشاره کرد . به عنوان مثال می توان به گونه ای برنامه نوشت که عمل چاپ اطلاعات از سطر چهام شروع شود و یا در خواندن اطلاعات 5 حرف اول نادیده فرض شود .

قبل از توضیح میدانها بیاییم میدان را تعریف کنیم . برنامه های فرترن در کنسول غیر گرافیکی اجرا میشود یا به عبارت دیگر در محیط DOS ، ابتدا صفحه داس را تقسیم بندی می کنیم . می دانیم صفحه مشکی رنگی که در زمان اجرای برنامه ظاهر می شود (صفحه DOS) از نقاط نورانی به نام pixel تشکیل شده است . مجموع تعدادی از این پیکسل ها برای نمایش یک کاراکتر استفاده می شود در هر یک از این مجموعه ها تنها یک حرف نمایش داده می شود . حال اگر تعدادی از این مجموعه ی پیکسل ها کنار یکدیگر قرار گیرند تا بتوان تعداد زیادی حرف را نمایش داد یک میدان را تشکیل می دهند البته این جایگاه ها همگی دارای خصوصیت مشترک هستند مثلا همگی از نوع صحیح و یا همگی از نوع رشته ای هستند .

اکنون که با میدان ها آشنا شدیم ، ببینیم چگونه می توان این میدانها را اعمال کرد . اگر به خاطر داشته باشید در مبحث خواندن و نوشتن ، ویژگی با نام قالب به طور مختصر توضیح داده شد . دستور زیر را در نظر بگیرید :

Read (*,*) A

این دستور متغیر A را از صفحه کلید با قالب آزاد (بدون محدودیت) می خواند . حال اگر بخواهیم یک قالب را بر این متغیر اعمال کنیم به یکی از دو روش زیر عمل می کنیم :

```
Read ( *,1) A
1 Format ( F6.2)
```

یا

```
Read (*,"(F6.2)")A
```

چنانچه می بینید در قسمت FMT باید یک شماره label اختصاص داد که در خطی که حاوی این شماره است بلافاصله دستور Format قرار دارد و یا می توان به طور مستقیم میدان را در داخل یک پرانتز و گیومه قرار داد .

دستور Format دستور اجرایی نیست یعنی تا زمانی که به این دستور پرش داده نشود اجرا نخواهد شد پس لزومی ندارد که دستور فرمت در خط بعد از خواندن و یا نوشتن بیاید می توان آن را در هر قسمت از بدنه اصلی برنامه قرار داد .

اکنون باید تمامی میدان ها را شناخت (در مثال های ذکر شده در پایین منظور از □ همان Space است که در حین اجرای برنامه نمایش داده نمی شود)

- میدان I

این میدان به صورت زیر تعریف می شود :

I طول میدان [حداقل طول میدان .]

طول میدان تعداد جایگاه های ذکر شده در بالاست و حداقل طول میدان ، حداقل تعداد ارقام نمایش داده شده در میدان است . چنانچه طول عدد از طول میدان بیشتر باشد ، به اندازه طول میدان ستاره چاپ خواهد شد و اگر طول عدد از طول میدان کمتر باشد میدان از سمت راست پر می شود .

فرمت	عدد	خروجی	توضیحات
I3.3	11	011	حداقل طول میدان 3 است
I3.2	1	□01	
I3.2	4000	***	طول میدان کم است
I2.0	0	□□	جز استثناهاست
I2.1	0	0	
I3.2	11.1	***	در این میدان تنها اعداد صحیح و عبارت منطقی قرار می گیرند

- میدان مبنا ها B, O, Z

در این میدان ها ابتدا عدد را به مبنای مورد نظر برده و سپس مانند میدان I عمل کنید . میدان B عدد را به مبنای دو و O عدد را به مبنای 8 و Z عدد را به 16 می برد .

فرمت	عدد در مبنای 10	عدد در مبنای مورد نظر	خروجی
B4	9	1001	1001
B3	9	1001	***
B5.5	9	1001	01001
O4.3	27	33	□033
Z5	32767	7FFF	□7FFF

اکنون برنامه ای بنویسید که دو عدد را گرفته و عدد اول را در مبنای عدد دوم که یکی از سه عدد 2 یا 8 یا 16 است نمایش دهد :

```
Read *,N,IBase
```

```
Select case (IBase)
    Case (2)
         Print "(B0)",N
    Case (8)
         Print "(O0)",N
    Case (16)
         Print "(Z0)",N
End select
End
```

همانگونه که می بینید طول میدان صفر در نظر گرفته شده است و موجب می شود که طول میدان به اندازه خود خروجی باشد .

– میدان F

این میدان در حالت کلی به صورت زیر است:

تعداد رقم اعشار . طول کلی میدان F

برای محاسبه خروجی این میدان کافیست که ابتدا به اندازه طول کلی میدان جایگاه در نظر گرفته شود سپس از سمت راست به اندازه تعداد رقم اعشار جایگاه خالی گذاشته شود در جایگاه بعد ممیز قرار داده می شود و بقیه جایگاه ها در سمت چپ میدان به قسمت صحیح اختصاص داده می شود . حال چنانچه نتوانید میدان را بسازید در لحظه اجرا با پیغام خطا مواجه خواهید شد . آنچه در مورد این میدان مهم است آنکه طول قسمت صحیح عدد باید بتواند به طور کامل در قسمت صحیح میدان قرار گیرد در غیر اینصورت به تعداد طول کلی میدان ستاره چاپ خواهد شد و اگر طول قسمت اعشاری میدان کمتر از طول قسمت اعشاری عدد باشد تعدادی از رقمهای اعشار که در آن قسمت می توانند قرار گیرند با گرد کردن چاپ می شوند . برای فهم بهتر مطلب به مثالهای زیر توجه کنید :

میدان	عدد	خروجی	توضیحات
F6.2	15.5	□15.50	
F6.2	15.257	□15.26	عدد گرد شده است
F6.2	2115.25	******	در میدان قرار نمی گیرد
F6.2	-10.1	-10.10	
F3.1	12.1E2	***	ابتدا عدد محاسبه شود
F5.2	-.2	-0.20	تعداد ارقام اعشار باید حتما رعایت شود

- میدان نماد علمی

این میدان به وسیله سه مشخصه E ، EN ، ES تعیین می شود . حالت کلی E به صورت زیر است . دو میدان بعد نیز به همین صورت تعیین می شوند .

[طول توان E] طول قسمت اعشاری . طول کلی میدان E

در هر سه این میدانها ابتدا به اندازه طول کلی میدان جایگاه قرار دهید سپس از سمت راست شروع کرده و به اندازه طول توان جایگاه برای توان جدا کنید (چنانچه طول توان نوشته نشده باشد آنرا 2 در نظر بگیرید) سپس یک جایگاه را به علامت مثبت یا منفی اختصاص دهید ، جایگاه بعدی را برای حرف E در نظر بگیرید توجه کنید که اگر در فرمت e نوشته شده باشد شما نیز باید از حرف کوچک آن استفاده کنید و اگر نوشته نشده باشد منظور همان E است . بعد از این مرحله به اندازه طول قسمت اعشار جدا کرده و جایگاه بعدی را به ممیز اختصاص دهید . بقیه جایگاه ها در سمت راست برای قسمت صحیح باقی می ماند . چنانچه نتوانید این میدان را تولید کنید در حین اجرا با پیغام خطای مبنی بر اشتباه بودن میدان مواجه خواهید شد .

حال باید عدد را در میدان قرار دهیم از سمت چپ عدد شروع کرده و تمامی صفرهای موجود در سمت چپ را نادیده می گیریم سپس در مورد میدان E عدد بدست آمده را بعد ممیز می نویسیم و در صورتی که کل عدد در قسمت اعشاری جا نشود آنراً گرد می کنیم . در مورد میدان EN سه رقم عدد بدست آمده را سمت چپ ممیز و بقیه را در سمت راست ممیز قرار می دهیم . در مورد میدان ES یک رقم را در سمت چپ و بقیه را در سمت راست می نویسیم . باید توجه کرد که در صورت جا نشدن عدد باید آنرا گرد کرد . حال نوبت به محاسبه قسمت توان می رسد . با محاسبه میزان کوچک شدن عدد ، توان مناسبی را اختیار می کنیم . ممکن است عدد در این میدان کاملا تغییر کند .اگر عدد مورد نظر منفی بود یک جایگاه در سمت چپ به منفی تعلق می گیرد . چنانچه یکی از قسمتهای بالا قابل اجرا نباشد به اندازه طول کلی میدان ستاره چاپ خواهد شد . اکنون به مثالهای زیر توجه کنید :

فرمت	ورودی	خروجی
E10.3	121.454	□0.121E+03
E10.3	0.0012	□0.120E-02
ES11.3	100.125	□□1.001E+02
EN11.3	1000.125	100.012E+01
EN11.2	475867.222	□475.87E+03

- میدان L

این میدان به صورت زیر تعریف می شود :

طول کلی میدان L

این میدان بسیار ساده بوده و با ایجاد جایگاه ها در آخرین جایگاه یکی از دو کلمه F یا T را قرار می دهد به عنوان نمونه در میدان L3 مقدار .True به صورت T□□ چاپ می شود .

- میدان رشته ای A

این میدان برای قرار دادن متن استفاده می شود :

طول کلی میدان A

چنانچه میدان برای متن مورد نظر بزرگ باشد از سمت راست میدان پر می شود و در سمت چپ جاهای خالی باقی می گذارد و در غیر اینصورت از سمت راست متن به اندازه طول میدان چاپ خواهد شد . اگر طول میدان نوشته نشود ، طول آن به اندازه طول متن خواهد بود .

میدان	ورودی	خروجی
A4	Ali	□Ali
A3	Ali Reza	Ali
A	Ali Reza	Ali Reza

اکنون که این میدان ها را شناختیم می توانیم صفحه DOS را به میدانهای مختلف برای کنترل بیشتر تقسیم کنیم اما برای جابجایی و تنظیم میدان ها تعدادی دستور تحت عنوان کنترل قالب وجود دارد که در زیر به آنها اشاره می کنیم .

فرمان کنترل	مثال	توضیحات
T n	T 10	ایجاد میدان بعدی از ستون 10 خواهد بود
TL n	TL 5	از موقعیت فعلی به اندازه 5 جایگاه به سمت چپ می رود
TR n	TR 4	از موقعیت فعلی به اندازه 4 جایگاه به سمت راست می رود
n X	5 X	5 فاصله خالی را قرار می دهد
[r] /	3/یا ///	سه سطر خالی چاپ می کند
:		ادامه ایجاد میدانها را در صورت نبود متغیر متوقف می کند
S		قرار دادن علامت + اعداد به سیستم عامل بستگی پیدا میکند

SP		علامت + اعداد قرار داده می شود
SS		علامت + اعداد قرار داده نمی شود
k P	2 P	عدد را در 10 بتوان k ضرب می کند
BN		فاصله خالی بین ارقام یک عدد را حذف می کند
BZ		فاصله خالی بین ارقام یک عدد را به صفر تبدیل می کند

برای آگاهی کامل از نحوه اعمال این فرمت ها به ضمیمه مراجعه کنید .

فصل هفتم

دسترسی به فایل

ابتدا باید نکاتی در مورد نام فایل ها در سیستم عامل های مختلف گفته شود . همانطور که می دانید نام فایل متشکل از دو قسمت نام و پسوند است که توسط نقطه از هم جدا می شوند . وجود نام ضروری و وجود پسوند اختیاری است . باید توجه داشت که در نام فایلها باید قوانین مربوط به سیستم عامل موجود بر روی سیستم را رعایت کرد . به عنوان مثال در DOS 4.5 و قبل از آن نام فایل تنها 8 حرف می تواند باشد . در ویندوزها نام فایل می تواند تا 256 کاراکتر را به خود اختصاص دهد . نباید از حروف غیر مجاز در نام فایل ها استفاده کرد .

اکنون که با نام فایل آشنا شدیم ، می خواهیم دستورات باز کردن فایل ها را بیان کنیم :

OPEN (مشخصات باز کردن فایل)

CLOSE (مشخصات بستن فایل)

منظور از مشخصات ، یک سری از کلید واژه ها هستند که به دستور OPEN جهت باز کردن فایل کمک می کنند . اکنون به بررسی تک تک آنها می پردازیم .

ACCESS –

فرض کنید لیستی از اسامی دانشجویان را در اختیار دارید . به دانشجویان گفته می شود که به ترتیب لیست و به طور پشت سرهم بر روی صندلی ها بنشینند . اینک شما برای پیدا کردن نفر بیستم می توانید مستقیما به صندلی شماره 20 بروید زیرا ترتیبی بین لیست و نحوه نشستن دانشجویان وجود دارد . اما اگر به آنها این اجازه داده شود که هر شخص بتواند با هر فاصله دلخواهی از نفر قبل بر روی یک صندلی بنشیند آنگاه برای پیدا کردن نفر بیستم باید به ترتیب از نفر اول شروع کرده و تا پیدا کردن نفر بیستم به جستجوی خود ادامه دهیم . این مثال عینا در مورد فایل ها نیز صادق است . اگر طول هر رکورد کاملا مشخص باشد برای دسترسی به رکورد n ام می توانید با پشت سر گذاشتن طول رکورد *(n-1) حرف به اول رکورد مورد نظر برسید اما اگر طول هر رکورد با رکورد های دیگر متفاوت باشد می بایست تک تک رکوردها را تا رسیدن به رکورد مورد نظر بخوانیم . اگر شیوه اول را دسترسی مستقیم بنامیم و دومی را ترتیبی می توان با نوشتن یکی از دو عبارت "Direct" برای مستقیم و یا "Sequential" برای ترتیبی در مقابل =Access نحوه دسترسی به فایل را مشخص کرد .چنانچه هیچ عبارتی نوشته نشود دسترسی پیش فرض ترتیبی خواهد بود .

Action–

یکی دیگر از اختیاراتی که وجود دارد این است که ما می توانیم فایل را صرفا جهت انجام عمل خواندن و یا فقط نوشتن باز کنیم . کلیدواژه Action دارای سه حالت "Read" ، "Write" ، "ReadWrite" می باشد . که حالت اول برای حالت فقط خواندن و حالت دوم برای حالت فقط نوشتن و حالت سوم برای انجام هر دو عمل است . پیش فرض این دستور حالت سوم در نظر گرفته می شود .

ERR –

چنانچه در حین باز کردن فایل با پیغام خطا مواجه شویم در صورت وجود کلیدواژه ERR ادامه برنامه به شماره Label نوشته شده در جلوی آن ، پرش می کند .

File–

در مقابل این کلمه باید یک متن که مسیر دقیق فایل را بیان می کند قرار داد . چنانچه بخواهیم یک فایل موجود را باز کنیم باید حتما پسوند آنرا در صورت وجود بنویسیم . اگر تنها نام فایل ذکر شود مسیر پیش فرض مسیر قرار گیری برنامه خواهد بود .

Position–

که یکی از سه حالت "ASIS" ، "Rewind" ، "Append" می باشد و محل اشاره گر هارد برای باز کردن و جستو را به ترتیب موقعیت فعلی ، از اول و از آخر را تعیین می کند .حالت اول پیش فرض در نظر گرفته می شود

Status–

اگر مقدار آن "Old" باشد فایل تنها زمانی باز خواهد شد که وجود داشته باشد . اگر فایل مورد نظر وجود نداشته باشد ، خطا رخ خواهد داد . اگر مقدار آن "New" باشد فایل ایجاد خواهد شد و در صورتی که از پیش وجود داشته باشد پیغام خطا ظاهر خواهد شد .

اگر مقدار آن "Unknown" باشد در صورت وجود آنرا باز می کند و در صورت عدم وجود آنرا ساخته و بعد باز می کند . این حالت پیش فرض در نظر گرفته می شود . مقدار بعدی "Replace" است که فایل را ساخته و باز میکند اگر فایل از پیش بر روی هارد موجود باشد آنرا پاک کرده و مجددا می سازد . مقدار "Scratch" یک فایل موقت در مسیر temp می سازد زمانی مورد استفاده قرار می گیرد که باید فایل ساخته شده با بسته شدن برنامه بسته شود است .

Unit –

حال زمانی که تمام مشخصات بالا نوشته شد باید یک عدد به عنوان unit به فایل ساخته شده اختصاص داد تا در هنگام خواندن و نوشتن با این عدد کار کرد . به عبارت دیگر برای جلوگیری از نوشتن مجدد همه موارد بالا از یک عدد استفاده می شود . UNIT جهت معرفی فایل به برنامه استفاده می شود و در تمامی دستوراتی که باید با فایل باز شده ارتباط داشته باشند مورد استفاده قرار می گیرد .

به مثال زیر توجه کنید . 10 خط اطلاعات فایل اول دو بار در فایل دوم نوشته می شوند :

```
Character *(*) A
Open(Unit=2,File="C:\data.dat",ERR=23,Action="Read",Status="Old")
```

```
Open(Unit=3,File="C:\data.out",ERR=24,Action="Write",Status="New")
Do i= 1, 10
    Read (2,*)A
    Write(3,*) (A,j=1,2)
End do

Close(2)
Close(3)
GOTO 25
23 PRINT *,"ERROR OPENING INPUT FILE"
24 PRINT *,"ERROR OPENING OUTPUT FILE"
25 End
```

(خط اول این برنامه در COMPAQ FORTRAN خطا محسوب می شود و باید یک طول استایک برای متغیر رشته ای A تعریف کرد)

همانطور که می بینیدپس از پایان کار ، فایل های باز شده را بستیم . چنانچه این کار را انجام ندهید برنامه خود به خود فایل ها را می بندد .

فصل هشتم

ساختار کلی برنامه

تا به اینجا تمام کارها در بدنه اصلی برنامه نوشته می شد . می دانیم که تعریف متغیر ها در بالا و دستورات دیگر در پایین نوشته می شد و برنامه به کلمه End ختم می شد . حال می توانیم قسمت های دیگری را تحت عنوان زیر برنامه به برنامه خود اضافه کنیم که بعد از کلمه End می توانند قرار گیرند

Function–

در برنامه نویسی تمام توابع مورد نیاز ما نوشته نشده اند و گاهی خود ما نیاز به تعریف توابع جدیدی داریم برای این منظور از ساختار زیر استفاده می کنیم :

Function نام تابع (ورودی – لیست آرگومان ها)

بلوک دستوری

نام تابع = ...

End Function

همانطور که در بحث توابع آماده دیدید ، توابع با گرفتن یک یا چند آرگومان محاسبات خاصی را انجام داده و نتیجه محاسبات را در نام متغیر ذخیره می کنند بنابراین تنها یک خروجی دارند و حداقل یکبار می بایست در متن زیر برنامه عددی به آن اختصاص داده شود . به عنوان مثال در زیر برنامه زیر تابع Sin2 برای محاسبه سینوس یک زاویه از طریق بسط تیلور نوشته شده است و در بدنه اصلی اختلاف دو تابع Sin و Sin2 نوشته می شود .

```
Read *,A
Print *,Sin(A)-Sin2(A)
End
Function Sin2(x)
Sin2=x ; F=1 ; S=-1
Do i=3,20,2
    F=F*(i-1)*i
    Sin2=Sin2+S*x**i/F
    S=S*-1
End do
End Function
```

همانطور که می بینید فراخوانی توابع جدید مانند توابع آماده است . نکته ای که باید رعایت شود این است که ورودی تابع Sin2 از نوع اعشاری تعریف شد پس باید در بدنه اصلی برنامه عدد اعشاری به عنوان آرگومان قرار داده شود .

Subroutine–

اگر بخواهیم بیش از یک خروجی داشته باشیم از subroutine استفاده می کنیم به اینصورت که ابتدا ورودی های خود را به داخل تابع فرستاده و سپس از ورودی ها بعنوان خروجی استفاده می کنیم .
ساختار کلی به صورت زیر است .

Subroutine نام زیر برنامه (لیست ورودی ها و خروجی ها)

بلوک دستورات

End Subroutine

اکنون برنامه بالا را با این ساختار می نویسیم :

```
Read *,A
Call Sin2(A,B)
Print *,Sin(A)-B
End
Subroutine Sin2(x,y)
y=x ; F=1 ; S=-1
Do i=3,20,2
    F=F*(i-1)*i
    y=y+S*x**i/F
    S=S*-1
End do
End Subroutine
```

همانطور که می بینید فراخوانی Subroutine با دستور Call می باشد .

ضمیمه 1 : لیست کلیه توابع مورد نیاز برنامه نویسی

Name	Description	Argument/Function Type
ACOS	**ACOS**(*x*). Returns the arc cosine of *x* in radians between 0 and pi. When **ACOS** is passed as an argument, *x* must be REAL(4).	*x:* Real result: same type as *x*
ACOSD	**ACOSD**(*x*). Returns the arc cosine of *x* in degrees between 0 and 180. When **ACOSD** is passed as an argument, *x* must be REAL(4).	*x:* Real result: same type as *x*
ALOG	**ALOG**(*x*). Returns natural log of *x*.	*x*: REAL(4) result: REAL(4)
ALOG10	**ALOG10**(*x*). Returns common log (base 10) of *x*.	*x*: REAL(4) result: REAL(4)
ASIN	**ASIN**(*x*). Returns arc sine of *x* in radians between ±pi/2. When **ASIN** is passed as an argument, *x* must be REAL(4).	*x*: Real result: same type as *x*
ASIND	**ASIND**(*x*). Returns arc sine of *x* in degrees between ±90°. When **ASIND** is passed as an argument, *x* must be REAL(4).	*x*: Real result: same type as *x*
ATAN	**ATAN**(*x*). Returns arc tangent of *x* in radians between ±pi/2. When **ATAN** is passed as an argument, *x* must be REAL(4).	*x*: Real result: same type as *x*
ATAND	**ATAND**(*x*). Returns arc tangent of *x* in degrees between ±90°. When **ATAND** is passed as an argument, *x* must be REAL(4).	*x*: Real result: same type as *x*
ATAN2	**ATAN2**(*y*, *x*). Returns the arc tangent of *y*/*x* in radians between ±pi . When **ATAN2** is passed as an argument, *y* and *x* must be REAL(4).	*y*: Real *x*: same as *y* result: same type as *y*
ATAN2D	**ATAN2D**(*y*, *x*). Returns the arc tangent of *y*/*x* in degrees between ±180°. When **ATAN2D** is passed as an argument, *y* and *x* must be REAL(4).	*y*: Real *x*: same as *y* result: same type as *y*

CCOS	**CCOS**(*x*). Returns complex cosine of *x*.	*x*: COMPLEX(4) result: COMPLEX(4)
CDCOS	**CDCOS**(*x*). Returns double-precision complex cosine of *x*.	*x*: COMPLEX(8) result: COMPLEX(8)
CDEXP	**CDEXP**(*x*). Returns double-precision complex value of *e***x*.	*x*: COMPLEX(8) result: COMPLEX(8)
CDLOG	**CDLOG**(*x*). Returns double-precision complex natural log of *x*.	*x*: COMPLEX(8) result: COMPLEX(8)
CDSIN	**CDSIN**(*x*). Returns double-precision complex sine of *x*.	*x*: COMPLEX(8) result: COMPLEX(8)
CDSQRT	**CDSQRT**(*x*). Returns double-precision complex square root of *x*.	*x* COMPLEX(8) result: COMPLEX(8)
CEXP	**CEXP**(*x*). Returns complex value of *e***x*.	*x*: COMPLEX(4) result: COMPLEX(4)
CLOG	**CLOG**(*x*). Returns complex natural log of *x*.	*x*: COMPLEX(4) result: COMPLEX(4)
COS	**COS**(*x*). Returns cosine of *x* radians. When **COS** is passed as an argument, *x* must be REAL(4).	*x*: Real or Complex result: same type as *x*
COSD	**COSD**(*x*). Returns cosine of *x* degrees. When **COSD** is passed as an argument, *x* must be REAL(4).	*x*: Real result: same type as *x*
COSH	**COSH**(*x*). Returns the hyperbolic cosine of *x*. When **COSH** is passed as an argument, *x* must be REAL(4).	*x*: Real result: same type as *x*
COTAN	**COTAN** (*x*). Returns cotangent of *x* in radians.	*x*: Real result: same type as *x*
COTAND	**COTAND** (*x*). Returns cotangent of *x* in degrees.	*x*: Real

		result: same type as *x*
CSIN	**CSIN**(*x*). Returns complex sine of *x*.	*x*: COMPLEX(4) result: COMPLEX(4)
CSQRT	**CSQRT**(*x*). Returns complex square root of *x*.	*x*: COMPLEX(4) result: COMPLEX(4)
DACOS	**DACOS**(*x*). Returns double-precision arc cosine of *x* in radians between 0 and pi.	*x*: REAL(8) result: REAL(8)
DACOSD	**DACOSD**(*x*). Returns the arc cosine of *x* in degrees between 0 and 180. When **DACOSD** is passed as an argument, *x* must be REAL(4).	*x*: REAL(8) result: REAL(8)
DASIN	**DASIN**(*x*). Returns double-precision arc sine of *x* in radians between ±pi/2.	*x*: REAL(8) result: REAL(8)
DASIND	**DASIND**(*x*). Returns double-precision arc sine of *x* in degrees between ±90°.	*x*: REAL(8) result: REAL(8)
DATAN	**DATAN**(*x*). Returns double-precision arc tangent of *x* in radians between ±pi/2.	*x*: REAL(8) result: REAL(8)
DATAND	**DATAND**(*x*). Returns double-precision arc tangent of *x* in degrees between ±90°.	*x*: REAL(8) result: REAL(8)
DATAN2	**DATAN2**(*y*, *x*). Returns double-precision arc tangent of *y*/*x* in radians between ±pi.	*y*: REAL(8) *x*: REAL(8) result: REAL(8)
DATAN2D	**DATAN2D**(*y*, *x*). Returns double-precision arc tangent of *y*/*x* in degrees between ±180°.	*y*: REAL(8) *x*: REAL(8) result: REAL(8)
DCOS	**DCOS**(*x*). Returns double-precision cosine of *x* in radians.	*x*: REAL(8) result: REAL(8)

DCOSD	**DCOSD**(*x*). Returns double-precision cosine of *x* in degrees.	*x*: REAL(8) result: REAL(8)
DCOSH	**DCOSH**(*x*). Returns double-precision hyperbolic cosine of *x*.	*x*: REAL(8) result: REAL(8)
DCOTAN	**DCOTAN**(*x*). Returns double-precision cotangent of *x*.	*x*: REAL(8) result: REAL(8)
DEXP	**DEXP**(*x*). Returns double-precision value of *e****x*	*x*: REAL(8) result: REAL(8)
DLOG	**DLOG**(*x*). Returns double-precision natural log of *x*.	*x*: REAL(8) result: REAL(8)
DLOG10	**DLOG10**(*x*). Returns double-precision common log (base 10) of *x*.	*x*: REAL(8) result: REAL(8)
DSIN	**DSIN**(*x*). Returns double-precision sin of *x* in radians.	*x*: REAL(8) result: REAL(8)
DSIND	**DSIND**(*x*). Returns double-precision sin of *x* in degrees.	*x*: REAL(8) result: REAL(8)
DSINH	**DSINH**(*x*). Returns double-precision hyperbolic sine of *x*.	*x*: REAL(8) result: REAL(8)
DSQRT	**DSQRT**(*x*). Returns double-precision square root of *x*.	*x*: REAL(8) result: REAL(8)
DTAN	**DTAN**(*x*). Returns double-precision tangent of *x* in radians.	*x*: REAL(8) result: REAL(8)
DTAND	**DTAND**(*x*). Returns double-precision tangent of *x* in degrees.	*x*: REAL(8) result: REAL(8)
DTANH	**DTANH**(*x*). Returns double-precision hyperbolic tangent of *x*.	*x*: REAL(8)

		result: REAL(8)
EXP	**EXP**(*x*). Returns value of *e***x*. When **EXP** is passed as an argument, *x* must be REAL(4).	*x*: Real or Complex result: same type as *x*
LOG	**LOG**(*x*) Returns the natural log of *x*.	*x*: Real or Complex result: same type as *x*
LOG10	**LOG10**(*x*). Returns the common log (base 10) of *x*.	*x*: Real result: same type as *x*
	SIN(*x*). Returns the sine of *x* radians. When **SIN** is passed as an argument, *x* must be REAL(4).	*x*: Real or Complex result: same type as *x*
SIND	**SIND**(*x*). Returns the sine of *x* degrees. When **SIND** is passed as an argument, *x* must be REAL(4).	*x*: Real result: same type as *x*
SINH	**SINH**(*x*). Returns the hyperbolic sine of *x*. When **SINH** is passed as an argument, *x* must be REAL(4).	*x*: Real result: same type as *x*
SQRT	**SQRT**(*x*). Returns the square root of *x*. When **SQRT** is passed as an argument, *x* must be REAL(4).	*x*: Real or Complex result: same type as *x*
TAN	**TAN**(*x*). Returns the tangent of *x* radians. When **TAN** is passed as an argument, *x* must be REAL(4).	*x*: Real result: same type as *x*
TAND	**TAND**(*x*). Returns the tangent of *x* degrees. When **TAND** is passed as an argument, *x* must be REAL(4).	*x*: Real result: same type as *x*
TANH	**TANH**(*x*). Returns the hyperbolic tangent of *x*. When **TANH** is passed as an argument, *x* must be REAL(4).	*x*: Real result: same type as *x*

ضمیمه 2 : برنامه ها

برنامه 1 : محاسبه ب.م . م دو عدد

```
PROGRAM  GreatestCommonDivisor
   IMPLICIT  NONE

   INTEGER   :: a, b, c

   WRITE(*,*) 'Two positive integers please --> '
   READ(*,*)  a, b
   IF (a < b) THEN        ! since a >= b must be true, they
      c = a               ! are swapped if a < b
      a = b
      b = c
   END IF

   DO                     ! now we have a <= b
      c = MOD(a, b)       !    compute c, the reminder
      IF (c == 0) EXIT    !    if c is zero, we are done.  GCD = b
      a = b               !    otherwise, b becomes a
      b = c               !    and c becomes b
   END DO                 !    go back

   WRITE(*,*) 'The GCD is ', b

END PROGRAM  GreatestCommonDivisor
```

برنامه 2 : محاسبه جذر یک عدد از طریق رابطه نیوتن

```
PROGRAM  SquareRoot
   IMPLICIT  NONE
   REAL    :: Input, X, NewX, Tolerance
   INTEGER :: Count
   READ(*,*)  Input, Tolerance
   Count = 0
   X     = Input
   DO
      Count = Count + 1
      NewX  = 0.5*(X + Input/X)
      IF (ABS(X - NewX) < Tolerance)  EXIT
      X = NewX
   END DO
   WRITE(*,*)  'After ', Count, ' iterations:'
   WRITE(*,*)  '  The estimated square root is ', NewX
   WRITE(*,*)  '  The square root from SQRT() is ', SQRT(Input)
   WRITE(*,*)  '  Absolute error = ', ABS(SQRT(Input) - NewX)

END PROGRAM  SquareRoot
```

برنامه 3 : یافتن تمامی عوامل اول یک عدد

```
PROGRAM  Factorize
   IMPLICIT  NONE

   INTEGER  :: Input
   INTEGER  :: Divisor
```

```
   INTEGER  :: Count

   WRITE(*,*)  'This program factorizes any integer >= 2 --> '
   READ(*,*)   Input

   Count = 0
   DO
      IF (MOD(Input,2) /= 0 .OR. Input == 1)  EXIT
      Count = Count + 1
      WRITE(*,*)  'Factor # ', Count, ': ', 2
      Input = Input / 2
   END DO

   Divisor = 3
   DO
      IF (Divisor > Input) EXIT
      DO
         IF (MOD(Input,Divisor) /= 0 .OR. Input == 1)  EXIT
         Count = Count + 1
         WRITE(*,*)  'Factor # ', Count, ': ', Divisor
         Input = Input / Divisor
      END DO
      Divisor = Divisor + 2
   END DO

END PROGRAM  Factorize
```

برنامه 4: نمایش مثلث بالایی یک ماتریس 10 در 10

```
PROGRAM  UpperTriangularMatrix
   IMPLICIT   NONE
   INTEGER, PARAMETER                :: SIZE = 10
   INTEGER, DIMENSION(1:SIZE,1:SIZE) :: Matrix
   INTEGER                           :: Number
   INTEGER                           :: Position
   INTEGER                           :: i, j
   CHARACTER(LEN=100)                :: Format

   READ(*,"(I5)")  Number
   DO i = 1, Number
      READ(*,"(10I5)")  (Matrix(i,j), j = 1, Number)
   END DO

   WRITE(*,"(1X,A)")  "Input Matrix:"
   DO i = 1, Number
      WRITE(*,"(1X,10I5)")  (Matrix(i,j), j = 1, Number)
   END DO

   WRITE(*,"(/1X,A)") "Upper Triangular Part:"
   Position = 2
   DO i = 1, Number
      WRITE(Format,"(A,I2.2,A)")  "(T", Position, ", 10I5)"
      WRITE(*,Format)  (Matrix(i,j), j = i, Number)
      Position = Position + 5
   END DO
END PROGRAM  UpperTriangularMatrix
```

برنامه 5: چاپ جدول ضرب

```
PROGRAM  Multiplication_Table
   IMPLICIT  NONE
   INTEGER, PARAMETER :: MAX = 9
   INTEGER            :: i, j
   CHARACTER(LEN=80)  :: FORMAT

   FORMAT = "(9(2X, I1, A, I1, A, I2))"
   DO i = 1, MAX
      WRITE(*,FORMAT) (i, '*', j, '=', i*j, j = 1, MAX)
   END DO
END PROGRAM  Multiplication_Table
```

برنامه 6: مرتب کردن داده ها

```
PROGRAM  Sorting
   IMPLICIT  NONE
   INTEGER, PARAMETER :: MAX_SIZE = 100
   INTEGER, DIMENSION(1:MAX_SIZE) :: InputData
   INTEGER                        :: ActualSize
   INTEGER                        :: i

   READ(*,*)  ActualSize, (InputData(i), i = 1, ActualSize)
   WRITE(*,*) "Input Array:"
   WRITE(*,*) (InputData(i), i = 1, ActualSize)

   CALL  Sort(InputData, ActualSize)

   WRITE(*,*)
   WRITE(*,*) "Sorted Array:"
   WRITE(*,*) (InputData(i), i = 1, ActualSize)

CONTAINS

   INTEGER FUNCTION  FindMinimum(x, Start, End)
      IMPLICIT  NONE
      INTEGER, DIMENSION(1:), INTENT(IN) :: x
      INTEGER, INTENT(IN)                :: Start, End
      INTEGER                            :: Minimum
      INTEGER                            :: Location
      INTEGER                            :: i

      Minimum  = x(Start)
      Location = Start
      DO i = Start+1, End
         IF (x(i) < Minimum) THEN
            Minimum  = x(i)
            Location = i
         END IF
      END DO
      FindMinimum = Location
   END FUNCTION  FindMinimum

   SUBROUTINE  Swap(a, b)
      IMPLICIT  NONE
      INTEGER, INTENT(INOUT) :: a, b
      INTEGER                :: Temp

      Temp = a
```

```
      a     = b
      b     = Temp
   END SUBROUTINE  Swap


   SUBROUTINE  Sort(x, Size)
      IMPLICIT  NONE
      INTEGER, DIMENSION(1:), INTENT(INOUT) :: x
      INTEGER, INTENT(IN)                   :: Size
      INTEGER                               :: i
      INTEGER                               :: Location

      DO i = 1, Size-1
         Location = FindMinimum(x, i, Size)
         CALL  Swap(x(i), x(Location))        !
      END DO
   END SUBROUTINE  Sort

END PROGRAM  Sorting
```

برنامه 7 : محاسبه اعداد آرمسترانگ سه رقمی

```
PROGRAM  ArmstrongNumber
   IMPLICIT  NONE

   INTEGER :: a, b, c
   INTEGER :: abc, a3b3c3
   INTEGER :: Count

   Count = 0
   DO a = 0, 9
      DO b = 0, 9
         DO c = 0, 9
            abc    = a*100 + b*10 + c
            a3b3c3 = a**3 + b**3 + c**3
            IF (abc == a3b3c3) THEN
               Count = Count + 1
               WRITE(*,*)  'Armstrong number ', Count, ': ', abc
            END IF
         END DO
      END DO
   END DO

END PROGRAM  ArmstrongNumber
```

برنامه 8 : محاسبه تمامی حالات ایجاد 1000 ریالی توسط واحد های پولی

```
integer o
open(2,file="c:\1000.txt",status="replace")
write(2,*) "1000 Rls   500 Rls   200 Rls   100 Rls   50 Rls   20 Rls   10 Rls"
write(2,*) "--------  -------  -------  -------  ------  ------  ------"
do 1 i=0,1                                                          !1000 rls
      do 1  j=0,2                                                   !500 rls
            do 1 k=0,5                                              !200 rls
                  do 1 l=0,10                                       !100 rls
                        do 1 m=0,20                                 !50 rls
                              do 1 n=0,50                           !20 rls
```

```
                                        do 1 o=0,100        !10 rls
                                                if                                  &
(i*1000+j*500+k*200+l*100+m*50+n*20+o*10==1000) then
                                                        write(2,30) i,j,k,l,m,n,o
                                                end if

1 continue
30 format (3x,I1,10x,I1,9x,I1,9x,I2,7x,I2,7x,I2,6x,I3)
end
```

برنامه 9 : محاسبه ماکزیمم و مینی موم اعداد تا زمانی که صفر وارد نشده است .

```
Read *,A
Nmax=A ; Nmin=A
Do
Read *,A
Nmax=Max(A,Nmax) ; Nmin=Min(A,Nmin)
IF (A==0.) Goto 11
End do
11 print *,"Max=",Nmax,"Min=",Nmin
End
```

برنامه 10 : تحلیل خرپا با اعمال نیروهای عمودی به آن (ساده)

```
real ,allocatable :: f(:,:)
real ,allocatable :: p(:)
real A
integer N
character stat
read *,N,A
allocate(P(n+2))
allocate(f(-1:N+2,N+3))
do i= 1 , n+2
      print *,"Please enter P",i,":"
      read *,P(i)
      pi=pi+p(i)
end do
Do j= 2,n+2
      mp=mp+p(j)*(j-1)*A/2
end do
t=(n-1)/2
g=floor(T)
c2=mp/(g+1)
c1=-c2+pi
f(1,2)=c1*(1/sin(3.141592/3))
f(1,3)=-f(1,2)*cos(3.141592/3)
f(0,2)=0
f(n+1,n+3)=0
f(n+2,n+1)=0
f(n+2,n+2)=0
do j=2, n+1
      f(j,j+1)=-f(j-1,j)+p(j)
      f(j,j+2)=f(j-2,j)+f(j-1,j)*cos(3.141592/3)-
f(j,j+1)*cos(3.141592/3)
end do
print *,"-----------------------------------------------"

print *,"Calculation process completed"
print *,"c1=",C1
print *,"c2=",C2
print *,"=",50/SIN(3.141592/3)
```

```
print *,"Now is the time to Show th information"

1 read *,Nj
print *,"-----------------------------------------------"
if (F(nj,nj+2)<0) then
 STAT="T"
else
 STAT="C"
END IF
print *,"F(",Nj,",",Nj+2,")=",ABS(f(NJ,NJ+2)) ,"(",STAT,")"

if (F(nj,nj+1)<0) then
 STAT="T"
else
 STAT="C"
END IF
print *,"F(",Nj,",",Nj+1,")=",ABS(f(NJ,NJ+1)),"(",STAT,")"

if (F(nj-2,nj)<0) then
 STAT="T"
else
 STAT="C"
END IF
print *,"F(",Nj-2,",",Nj,")=",ABS(f(NJ-2,NJ)),"(",STAT,")"

if (F(nj-1,nj)<0) then
 STAT="T"
else
 STAT="C"
END IF
print *,"F(",Nj-1,",",Nj,")=",ABS(f(NJ-1,NJ)),"(",STAT,")"

if (P(nj)<0) then
 STAT="T"
else
 STAT="C"
END IF
print *,"P(",Nj,")=",ABS(P(NJ)),"(",STAT,")"
goto 1
end
```

برنامه 11 : جمع و ضرب دو ماتریس N در N

```
program MAT_Calculation
character c
real ,allocatable,dimension(:,:)::A,B,M,S
1 print *,"Please enter the N that belongs to matrices A & B:"
read *,N
allocate (a(N,N))
allocate (B(N,N))
allocate (S(N,N))
allocate (M(N,N))
do i= 1,N
      do J=1,N
            print *,"please enter the ",i,"&",j,"object of matrices"
            read *,a(i,J)
            read *,b(i,j)
            s(i,j)=a(i,j)+b(i,j)
      end do
end do
```

```
do t= 1 , N
    g=0
      do while (g<N)
            g=g+1
                  do j=1,n
                        tm=tm+A(t,j)*b(j,g)
                  end do
      end do
      m(t,g)=tm
      tm=0
end do
do i=1,n
      WRITE(*,11)  (M(i,j), j = 1, n)
end do
print *,"Do you want to do again?"
read *,C
if (c=="N" .or. c=="NO" .or. c=="n" .or. c=="no")  goto 12
goto 1
12 end
```

برنامه 12 : تشخیص اینکه آیا یک آرایه مقلوب است .

```
PROGRAM  Palindrome
   IMPLICIT  NONE

   INTEGER, PARAMETER :: LENGTH = 30     ! maximum array size
   INTEGER, DIMENSION(1:LENGTH) :: x     ! the array
   INTEGER            :: Size            ! actual array size (input)
   INTEGER            :: Head            ! pointer moving forward
   INTEGER            :: Tail            ! pointer moving backward
   INTEGER            :: i               ! running index

   READ(*,*) Size, (x(i), i = 1, Size)   ! read in the input array
   WRITE(*,*)  "Input array:"            ! display the input
   WRITE(*,*)  (x(i), i = 1, Size)

   Head = 1                              ! scan from the beginning
   Tail = Size                           ! scan from the end
   DO                                    ! checking array
      IF (Head >= Tail)  EXIT            ! exit if two pointers meet
      IF (x(Head) /= x(Tail))  EXIT  !exit if two elements not equal
      Head = Head + 1              ! equal.  Head moves forward
      Tail = Tail - 1              ! and Tail moves backward
   END DO                          ! until done

   WRITE(*,*)
   IF (Head >= Tail) THEN          ! if Head cross Tail, then we have
      WRITE(*,*) "The input array is a palindrome"
   ELSE
      WRITE(*,*) "The input array is NOT a palindrome"
   END IF

END PROGRAM  Palindrome
```

برنامه 13 : رسم مثلث بالایی یک ماتریس

```
PROGRAM  UpperTriangularMatrix
   IMPLICIT   NONE
```

```
   INTEGER, PARAMETER                :: SIZE = 10
   INTEGER, DIMENSION(1:SIZE,1:SIZE) :: Matrix
   INTEGER                           :: Number
   INTEGER                           :: Position
   INTEGER                           :: I, j
   CHARACTER(LEN=100)                :: Format
   READ(*,"(I5)")  Number
   DO i = 1, Number
      READ(*,"(10I5)")  (Matrix(i,j), j = 1, Number)
   END DO
   WRITE(*,"(1X,A)")  "Input Matrix:"
   DO i = 1, Number
      WRITE(*,"(1X,10I5)")  (Matrix(i,j), j = 1, Number)
   END DO
   WRITE(*,"(/1X,A)") "Upper Triangular Part:"
   Position = 2
   DO i = 1, Number
      WRITE(Format,"(A,I2.2,A)")  "(T", Position, ", 10I5)"
      WRITE(*,Format)  (Matrix(i,j), j = i, Number)
      Position = Position + 5
   END DO
END PROGRAM  UpperTriangularMatrix
```

برنامه 14 : تبدیل عدد تاریخ به سال و ماه و روز

```
PROGRAM  YYYYMMDDConversion
   IMPLICIT  NONE

   INTERFACE                              ! interface block
      SUBROUTINE  Conversion(Number, Year, Month, Day)
         INTEGER, INTENT(IN)  :: Number
         INTEGER, INTENT(OUT) :: Year, Month, Day
      END SUBROUTINE  Conversion
   END INTERFACE

   INTEGER :: YYYYMMDD, Y, M, D

   DO                                     ! loop until a zero is seen
      WRITE(*,*)  "A YYYYMMDD (e.g.,19971027) please (0 to stop)-> "
      READ(*,*)   YYYYMMDD                ! read in the value
      IF (YYYYMMDD == 0)  EXIT            ! if 0, then bail out

      CALL  Conversion(YYYYMMDD, Y, M, D)    ! do conversation

      WRITE(*,*)  "Year  = ", Y           ! display results
      WRITE(*,*)  "Month = ", M
      WRITE(*,*)  "Day   = ", D
      WRITE(*,*)
   END DO
END PROGRAM  YYYYMMDDConversion

! ------------------------------------------------------------------
! SUBROUTINE  Conversion():
!    This external subroutine takes an integer input Number in the
! form of YYYYMMDD and convert it to Year, Month and Day.
! ------------------------------------------------------------------

SUBROUTINE  Conversion(Number, Year, Month, Day)
   IMPLICIT  NONE
```

```
   INTEGER, INTENT(IN)  :: Number
   INTEGER, INTENT(OUT) :: Year, Month, Day

   Year  = Number / 10000
   Month = MOD(Number, 10000) / 100
   Day   = MOD(Number, 100)
END SUBROUTINE  Conversion
```

برنامه 15: محاسبه مکان و سرعت پرتابه

$$
\begin{aligned}
x &= ut\cos(a) \\
y &= ut\sin(a) - \frac{gt^2}{2} \\
V_x &= u\cos(a) \\
V_y &= u\sin(a) - gt
\end{aligned}
$$

$$
\begin{aligned}
V &= \sqrt{V_x^2 + V_y^2} \\
\tan(\theta) &= \frac{V_x}{V_y}
\end{aligned}
$$

```
! ----------------------------------------------------------------
! Given t, the time since launch, u, the launch velocity, a, the
! initial angle of launch (in degree),and g, the acceleration due to
! gravity, this program computes the position (x and y coordinates)
! and the velocity (magnitude and direction) of a projectile.
! ----------------------------------------------------------------

PROGRAM  Projectile
   IMPLICIT    NONE

   REAL, PARAMETER :: g  = 9.8       ! acceleration due to gravity
   REAL, PARAMETER :: PI = 3.1415926 ! you knew this.  didn't you

   REAL            :: Angle          ! launch angle in degree
   REAL            :: Time           ! time to flight
   REAL            :: Theta          ! direction at time in degree
   REAL            :: U              ! launch velocity
   REAL            :: V              ! resultant velocity
   REAL            :: Vx             ! horizontal velocity
   REAL            :: Vy             ! vertical velocity
   REAL            :: X              ! horizontal displacement
   REAL            :: Y              ! vertical displacement

   READ(*,*)  Angle, Time, U

   Angle = Angle * PI / 180.0           ! convert to radian
   X     = U * COS(Angle) * Time
   Y     = U * SIN(Angle) * Time - g*Time*Time / 2.0
   Vx    = U * COS(Angle)
   Vy    = U * SIN(Angle) - g * Time
   V     = SQRT(Vx*Vx + Vy*Vy)
   Theta = ATAN(Vy/Vx) * 180.0 / PI

   WRITE(*,*)  'Horizontal displacement : ', X
   WRITE(*,*)  'Vertical displacement   : ', Y
```

```
   WRITE(*,*)  'Resultant velocity      : ', V
   WRITE(*,*)  'Direction (in degree)   : ', Theta

END PROGRAM  Projectile
```

برنامه 16 : محاسبه مساحت زیر نمودار تابع داده شده

```
      program area

C*********************************************************************
*
C  Compute area under the curve y = x**2 + 1 given user defined
C     stop and stepsize
C*********************************************************************

      real start, stop, delta, sum

      print *, 'Enter the interval endpoints and number of',
     +         ' subintervals:'
      read *, start, stop, n

      delta = (stop - start) / n    ! INCREMENT SIZE

      sum = 0
      x = start + delta / 2
      print *, x
      print *, x, stop, delta
      do 10 rcnt = x, stop, delta
         height = rcnt**2+1
       area = height * delta
       sum = sum + area
       print *, height, delta, rcnt, area, sum
 10   continue

      print *, 'Appx area using ',n,' subintervals is ',sum

      stop
      end
```

برنامه 17 :یافتن داده خاص در بین آرایه ها

```
! ------------------------------------------------------------------
! PROGRAM  TableLookUp
!    Given an array and a input value, this program can determine if
! the value if in the table.  If it is, the array location where the
! value is stored is returned.
! ------------------------------------------------------------------

PROGRAM  TableLookUp
   IMPLICIT  NONE
   INTEGER, PARAMETER              :: TableSize = 100
   INTEGER, DIMENSION(1:TableSize) :: Table
   INTEGER                         :: ActualSize
   INTEGER                         :: Key
   INTEGER                         :: Location
```

```
   INTEGER                          :: i
   INTEGER                          :: end_of_input

   READ(*,*)  ActualSize          ! read in the actual size and table
   READ(*,*)  (Table(i), i = 1, ActualSize)
   WRITE(*,*) "Input Table:"
   WRITE(*,*) (Table(i), i = 1, ActualSize)
   WRITE(*,*)
   DO                             ! keep reading in a key value
      WRITE(*,*)  "A search key please --> "
      READ(*,*,IOSTAT=end_of_input)  Key
      IF (end_of_input < 0)  EXIT     ! EXIT of end-of-file reached
      Location = LookUp(Table, ActualSize, Key)! do a table look up
      IF (Location > 0) THEN            ! display the search result
         WRITE(*,*)  "Key value ", Key, " location ", Location
      ELSE
         WRITE(*,*)  "Key value ", Key, " is not found"
      END IF
   END DO
   WRITE(*,*)
   WRITE(*,*) "Table lookup operation completes"

CONTAINS

! ------------------------------------------------------------------
! INTEGER FUNCTION  LookUp():
! Given an array x() and a key value Data, this function determines
! if Data is a member of x().  If it is, the index where Data can be
! found is returned; otherwise, it returns 0.
! ------------------------------------------------------------------

   INTEGER FUNCTION  LookUp(x, Size, Data)
      IMPLICIT  NONE
      INTEGER, DIMENSION(1:), INTENT(IN) :: x
      INTEGER, INTENT(IN)                :: Size
      INTEGER, INTENT(IN)                :: Data
      INTEGER                            :: i

      LookUp = 0                      ! assume not found
      DO i = 1, Size                  ! check each array element
         IF (x(i) == Data) THEN       !   is it equal to Data?
            LookUp = i                ! YES, found.  Record location
            EXIT                      !   and bail out
         END IF
      END DO
   END FUNCTION  LookUp

END PROGRAM  TableLookUp
```

برنامه 18 : رسم نمودار میله ای

```
PROGRAM  VerticalBarChart
   IMPLICIT  NONE
   CHARACTER(LEN=*), PARAMETER  :: Part1 = "(1X, I5, A,"
   CHARACTER(LEN=*), PARAMETER  :: Part2 = "A, A, I2, A)"
   CHARACTER(LEN=2)             :: Repetition
   CHARACTER(LEN=10), PARAMETER :: InputFormat = "(I5/(5I5))"
   INTEGER                      :: Number, i, j
```

```
   INTEGER, DIMENSION(1:100)    :: Data

   READ(*,InputFormat)  Number, (Data(i), i=1, Number)
   DO i = 1, Number
      IF (Data(i) /= 0) THEN
         WRITE(Repetition,"(I2)")  Data(i)
         WRITE(*,Part1 // Repetition // Part2)  Data(i), " |", &
("*", j=1,Data(i)), " (", Data(i), ")"
      ELSE
         WRITE(*,"(1X, I5, A, I2, A)")  Data(i), " | (", Data(i)&
, ")"
      END IF
   END DO
END PROGRAM  VerticalBarChart
```

برنامه 19 : نمایش ساعت و تاریخ

```
PROGRAM CLOCK
! Program asking the computer for date and time
IMPLICIT NONE
CHARACTER (LEN=8) DATE        ! date in format ccyymmdd
CHARACTER (LEN=10) TIME       ! time in format hhmmss.sss
CHARACTER (LEN=5) ZONE        ! time zone (rel to UTC) as Shhmm
INTEGER VALUES(8)             ! year, month, day, mins from UTC,
                              ! hours, min, sec, msec
CHARACTER (LEN=8) TIMESTRING    ! time in the format hh:mm:ss
CHARACTER (LEN=10) DATESTRING   ! date in the format dd-mm-yyyy
                               ! Ask the system for the date and time
CALL DATE_AND_TIME( DATE, TIME, ZONE, VALUES )
                                    ! Convert to desired format
TIMESTRING = TIME( 1:2 ) // ':' // TIME( 3:4 ) // ':' // TIME( 5:6 )
DATESTRING = DATE( 7:8 ) // '-' // DATE( 5:6 ) // '-' // DATE( 1:4 )
                              ! Output the desired time and date
PRINT *, 'It is ', TIMESTRING, ' on ', DATESTRING
END PROGRAM CLOCK
```

برنامه 20 : جدول زوایا و روابط مثلثاتی آنها

```
PROGRAM TRIG_TABLE
! Compiles a table of SIN, COS, TAN against angle in DEGREES
IMPLICIT NONE
INTEGER DEG ! angle in degrees
REAL RAD ! angle in radians
REAL PI ! mathematical pi
CHARACTER (LEN=*), PARAMETER :: FMTHEAD = '( 1X, A3, 3( 2X, A7 ) )'
CHARACTER (LEN=*), PARAMETER :: FMTDATA = '( 1X, I3, 3( 2X, F7.4 ) )'
! formats for headings and data
PI = 4.0 * ATAN( 1.0 )
WRITE ( *, FMTHEAD ) 'Deg', 'Sin', 'Cos', 'Tan'
DO DEG = 0, 80, 10
RAD = DEG * PI / 180.0
WRITE ( *, FMTDATA ) DEG, SIN( RAD ), COS( RAD ), TAN( RAD )
END DO
END PROGRAM TRIG_TABLE
```

برنامه 21 :برنامه جدول توانی

```
PROGRAM EXPTABLE
! Program tabulates EXP(X)
IMPLICIT NONE
INTEGER :: NSTEP = 15 ! number of steps
REAL :: XMIN = 0.0, XMAX = 3.0 ! interval limits
REAL DELTAX ! step size
REAL X ! current X value
INTEGER I ! a counter
! Format specifiers
CHARACTER (LEN=*), PARAMETER :: FMT1 = '( 1X, A4 , 2X, A10 )'
CHARACTER (LEN=*), PARAMETER :: FMT2 = '( 1X, F4.2, 2X, 1PE10.3 )'
DELTAX = ( XMAX - XMIN ) / NSTEP ! calculate step size
WRITE ( *, FMT1 ) 'X', 'EXP' ! write headers
DO I = 0, NSTEP
X = XMIN + I * DELTAX ! set X value
WRITE ( *, FMT2 ) X, EXP( X ) ! write data
END DO
END PROGRAM EXPTABLE
```

برنامه 22:برنامه آنالیز یک متن

```
PROGRAM ANALYSE_TEXT
IMPLICIT NONE
INTEGER :: IO = 0 ! holds i/o status
INTEGER :: NLETTERS = 0 ! number of letters read
INTEGER :: NWORDS = 0 ! number of words read
CHARACTER CH, LAST_CH ! successive characters
CHARACTER (LEN=*), PARAMETER :: ALPHABET = &
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
CHARACTER, PARAMETER :: SPACE=' '
LAST_CH = SPACE
! Open the text file
OPEN ( 10, FILE = 'text.dat' )
! Read characters one-by-one until end of file is reached
DO WHILE ( IO /= -1 ) ! IO=-1 means EOF
! Read one character
READ ( 10, '( A1 )', IOSTAT = IO, ADVANCE = 'NO' ) CH
IF ( IO == 0 ) THEN ! a character has been read
PRINT *, 'Character = ', CH
! Is it a new word?
IF (LAST_CH == SPACE .AND. CH /= SPACE ) NWORDS = NWORDS + 1
! Is it a letter of the alphabet or something else?
IF ( INDEX( ALPHABET, CH ) /= 0 ) NLETTERS = NLETTERS + 1
LAST_CH = CH ! update last character
ELSE ! end of line or end of file
PRINT *, 'IO = ', IO
LAST_CH = SPACE
END IF
END DO
! Close the text file
CLOSE (10)
! Output the analysis
PRINT *, 'Number of letters = ', NLETTERS
PRINT *, 'Number of words = ', NWORDS
END PROGRAM ANALYSE_TEXT
```

برنامه 23: تبدیل مختصات کارتزین به قطبی

```
PROGRAM COORDINATES
! Program to convert from Cartesian to polar coordinates
IMPLICIT NONE
```

```
EXTERNAL POLARS
REAL X, Y
REAL R, THETA
PRINT *, 'Input coordinates X and Y'
READ *, X, Y
CALL POLARS( X, Y, R, THETA )
PRINT *, 'R, THETA =', R, THETA
END PROGRAM COORDINATES
!=========================
SUBROUTINE POLARS( X, Y, R, THETA )
! Subroutine transforming input (X, Y) to output (R, THETA)
IMPLICIT NONE
REAL, INTENT(IN) :: X, Y ! cartesian coordinates (input)
REAL, INTENT(OUT) :: R, THETA ! polar coordinates (output)
REAL, PARAMETER :: PI = 3.141593 ! the constant pi
R = SQRT( X ** 2 + Y ** 2 ) ! radius
THETA = ATAN2( Y, X ) ! inverse tangent between -pi and pi
IF ( THETA < 0.0 ) THETA = THETA + 2.0 * PI
! angle between 0 and 2 pi
THETA = THETA * 180.0 / PI ! convert to degrees
END SUBROUTINE POLARS
```

برنامه 24: جابجایی مقدار متغیرها با یکدیگر

```
PROGRAM EXAMPLE
! Program to swap two numbers
IMPLICIT NONE
EXTERNAL SWAP
INTEGER I, J
PRINT *, 'Input integers I and J'
READ *, I, J
CALL SWAP( I, J )
PRINT *, 'Swapped numbers are ', I, J
END PROGRAM EXAMPLE
!=========================================
SUBROUTINE SWAP( M, N )
IMPLICIT NONE
INTEGER M, N ! numbers to be swapped
INTEGER TEMP ! temporary storage
TEMP = M ! store number before changing it
M = N
N = TEMP
END SUBROUTINE SWAP
```

برنامه 25:نمایش نام دانشجوبانی که معدلشان بین دو نمره داده شده است .

```
PROGRAM Student_Average
INTEGER N
REAL , ALLOCATABLE :: Score(:)
REAL Min,MAX
CHARACTER , ALLOCATABLE,DIMENSION(:) :: Name(:)*20
PRINT *,"Please enter the number of students :"
READ (*,"(I)") N
ALLOCATE(Score(N))
ALLOCATE(Name(N))
DO I= 1 , N
    PRINT *,"Enter the name  of the student number ",I
    READ (*,"(A20)") Name(I)
```

```
    PRINT *,"Enter the  score of the student number ",I
    READ (*,"(F5.2)") Score(I)

END DO
PRINT *,"Please enter the min & max score "
READ (*,"(F5.2,F5.2)") Min,Max
IF (Min>Max) THEN
   A=Max
                Max=Min
   Min=A
END IF
DO J=1,N
    IF (Score(J)>=Min .AND. Score(J)<=Max ) THEN
        PRINT "(A20,F5.2)" , Name(J),Score(J)
    END IF
END DO
END PROGRAM Student_Average
```

برنامه 26:محاسبه مساحت مثلث

```
program Triangle
real a, b, c
print *, 'Enter the lengths of the three sides of the triangle'
read *, a, b, c
print *, 'Triangle''s area: ', triangleArea( a, b, c )
contains
function triangleArea( a, b, c )
real triangleArea
real, intent( in ) :: a, b, c
real theta
real height
theta = acos( ( a**2 + b**2 - c**2 ) / ( 2.0 * a * b ) )
height = a * sin( theta )
triangleArea = 0.5 * b * height
end function triangleArea
end program Triangle
```

برنامه 27: محاسبه میانگین ، واریانس و انحراف از معیار

```
PROGRAM EXAMPLE
! Program computes mean, variance and standard deviation
IMPLICIT NONE
EXTERNAL STATS ! subroutine to be used
INTEGER NVAL ! number of values
REAL, ALLOCATABLE :: X(:) ! data values
REAL MEAN, VARIANCE, STANDARD_DEVIATION ! statistics
INTEGER N ! a counter
! Open data file
OPEN ( 10, FILE = 'stats.dat' )
! Read the number of points and set aside enough memory
READ ( 10, * ) NVAL
ALLOCATE ( X(NVAL) )
! Read data values
READ ( 10, * ) ( X(N), N = 1, NVAL )
CLOSE ( 10 )
! Compute statistics
CALL STATS( NVAL, X, MEAN, VARIANCE, STANDARD_DEVIATION )
```

```
! Output results
PRINT *, 'Mean = ', MEAN
PRINT *, 'Variance = ', VARIANCE
PRINT *, 'Standard deviation = ', STANDARD_DEVIATION
! Recover computer memory
DEALLOCATE ( X )
END PROGRAM EXAMPLE
!=====================================================
SUBROUTINE STATS( N, X, M, VAR, SD )
! This works out the sample mean, variance and standard deviation
IMPLICIT NONE
INTEGER, INTENT(IN) :: N ! array size
REAL, INTENT(IN) :: X(N) ! data values
REAL, INTENT(OUT) :: M, VAR, SD ! statistics
! Calculate statistics using array operation SUM
M = SUM( X ) / N ! mean
VAR = SUM( X * X ) / N - M ** 2 ! variance
SD = SQRT( VAR ) ! standard deviation
END SUBROUTINE STATS
```

برنامه28:یافتن ریشه های معادله بوسیله روش نصف کردن

$$f(x) = \sqrt{\frac{\pi}{2}}e^{ax} + \frac{x}{a^2 + x^2} \qquad a = 0.8475$$

```
! ------------------------------------------------------------------
!    This program solves equations with the Bisection Method.  Given
! a function f(x) = 0.  The bisection method starts with two values,
! a and b such that f(a) and f(b) have opposite signs.  That is,
! f(a)*f(b) < 0.  Then, it is guaranteed that f(x)=0 has a root in
! the range of a and b.  This program reads in a and b (Left and
!Right
! in this program) and find the root in [a,b].
!    In the following, function f() is REAL FUNCTION Funct() and
! solve() is the function for solving the equation.
! ------------------------------------------------------------------

PROGRAM  Bisection
   IMPLICIT  NONE

   REAL, PARAMETER :: Tolerance = 0.00001
   REAL            :: Left,  fLeft
   REAL            :: Right, fRight
   REAL            :: Root

   WRITE(*,*)  'This program can solves equation F(x) = 0'
   WRITE(*,*)  'Please enter two values Left and Right such that '
   WRITE(*,*)  'F(Left) and F(Right) have opposite signs.'
   WRITE(*,*)
   WRITE(*,*)  'Left and Right please --> '
   READ(*,*)   Left, Right       ! read in Left and Right

   fLeft  = Funct(Left)          ! compute their function values
   fRight = Funct(Right)
   WRITE(*,*)
   WRITE(*,*)  'Left = ', Left, '    f(Left) = ', fLeft
```

```
   WRITE(*,*)  'Right = ', Right, '   f(Right) = ', fRight
   WRITE(*,*)
   IF (fLeft*fRight > 0.0)  THEN
      WRITE(*,*)  '*** ERROR: f(Left)*f(Right) must be negative ***'
   ELSE
      Root = Solve(Left, Right, Tolerance)
      WRITE(*,*)  'A root is ', Root
   END IF

CONTAINS

! ------------------------------------------------------------------
! REAL FUNCTION  Funct()
!    This is for function f(x).  It takes a REAL formal argument and
! returns the value of f() at x.  The following is sample function
! with a root in the range of -10.0 and 0.0.  You can change the
! expression with your own function.
! ------------------------------------------------------------------

   REAL FUNCTION  Funct(x)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: x
      REAL, PARAMETER  :: PI = 3.1415926
      REAL, PARAMETER  :: a  = 0.8475

      Funct = SQRT(PI/2.0)*EXP(a*x) + x/(a*a + x*x)

   END FUNCTION  Funct

! ------------------------------------------------------------------
! REAL FUNCTION  Solve()
!    This function takes Left - the left end, Right - the right end,
! and Tolerance - a tolerance value such that f(Left)*f(Right) < 0
! and find a root in the range of Left and Right.
!    This function works as follows.  Because of INTENT(IN), this
! function cannot change the values of Left and Right and therefore
! the values of Left and Right are saved to a and b.
!    Then, the middle point c=(a+b)/2 and its function value f(c)
! is computed.  If f(a)*f(c) < 0, then a root is in [a,c]; otherwise,
! a root is in [c,b].  In the former case, replacing b and f(b) with
! c and f(c), we still maintain that a root in [a,b].  In the latter,
! replacing a and f(a) with c and f(c) will keep a root in [a,b].
! This process will continue until |f(c)| is less than Tolerance and
! hence c can be considered as a root.
! ------------------------------------------------------------------

   REAL FUNCTION  Solve(Left, Right, Tolerance)
      IMPLICIT  NONE
      REAL, INTENT(IN) :: Left, Right, Tolerance
      REAL             :: a, Fa, b, Fb, c, Fc

      a = Left                          ! save Left and Right
      b = Right

      Fa = Funct(a)                     ! compute the function values
      Fb = Funct(b)
      IF (ABS(Fa) < Tolerance) THEN     ! if f(a) is already small
         Solve = a                      ! then a is a root
      ELSE IF (ABS(Fb) < Tolerance) THEN     ! is f(b) is small
         Solve = b                      ! then b is a root
      ELSE                              ! otherwise,
```

```
         DO                                  ! iterate ....
            c  = (a + b)/2.0                 !   compute the middle point
            Fc = Funct(c)                    !   and its function value
            IF (ABS(Fc) < Tolerance) THEN     ! is it very small?
               Solve = c                     ! yes, c is a root
               EXIT
            ELSE IF (Fa*Fc < 0.0) THEN       ! do f(a)*f(c) < 0 ?
               b  = c                        ! replace b with c
               Fb = Fc                       ! and f(b) with f(c)
            ELSE                             ! then f(c)*f(b) < 0 holds
               a  = c                        ! replace a with c
               Fa = Fc                       ! and f(a) with f(c)
            END IF
         END DO                              ! go back and do it again
      END IF
   END FUNCTION  Solve

END PROGRAM  Bisection
```

برنامه 29: مربع جادویی

```
!  This program prints a magic squares array, an n by n matrix in
!  each integer 1, 2, ..., n*n appears exactly once and all columns,
!  rows, and diagonals sum to the same number.
!  Here is the result of a sample run:

! Order of magic squares matrix? 7
!    30   39   48    1   10   19   28
!    38   47    7    9   18   27   29
!    46    6    8   17   26   35   37
!     5   14   16   25   34   36   45
!    13   15   24   33   42   44    4
!    21   23   32   41   43    3   12
!    22   31   40   49    2   11   20

module stdtypes

!  symbolic name for kind type of 4 byte integers

  integer, parameter, public :: i4 = selected_int_kind (9)

!  one-byte storage of logical values. if unavailable, use default
!  logical by uncommenting default logical definition above.

  integer (kind = i4), parameter, public :: lg = 1_i4

end module stdtypes

module indexCheckM
  use stdtypes

  private
  public :: indexChecker

contains

  function indexChecker (row, col, rowdim, coldim) result(indexCheck)
```

```
  integer (kind = i4), intent (in) :: row, col, rowdim, coldim
  logical (kind = lg)  :: indexCheck
  if (row >= 1 .and. row <= rowdim .and. col >= 1 .and. col <=
coldim) then
    indexCheck = .true.
  else
    indexCheck = .false.
  end if
  end function  indexChecker

end module indexCheckM

program magicSquares

  use stdtypes
  use indexCheckM

  integer (kind = i4) :: matrixOrder, ios
  integer (kind = i4), dimension (:,:), pointer :: matrix
  integer (kind = i4) :: row, col, prow, pcol, k
  character (len = 32) :: rowformat

  write (unit = *, fmt = "(a)", iostat = ios, advance = "no") &
         "Order of magic squares matrix? "
  read (unit = *, fmt = *, iostat = ios) matrixOrder

  if (modulo(matrixOrder, 2) == 0) then
    print *, "Order of magic square matrix must be odd"
    stop
  end if

  allocate(matrix(matrixOrder, matrixOrder))
  matrix = 0

  row = 1
  col = (matrixOrder - 1)/2 + 1
  matrix(row, col) = 1

  do k = 2, matrixOrder*matrixOrder
    prow = row - 1
    pcol = col + 1
    if (indexChecker(prow, pcol, matrixOrder, matrixOrder)) then
      if (matrix(prow, pcol) == 0) then
        row = prow
        col = pcol
      else
        row = row + 1
      end if
    else if (prow < 1 .and. indexChecker(1, pcol, matrixOrder,
matrixOrder)) then
      row = matrixOrder
      col = pcol
    else if(indexChecker(prow, 1, matrixOrder, matrixOrder) .and.
pcol > matrixOrder) then
      row = prow
      col = 1
    else if (prow == 0 .and. pcol == matrixOrder + 1) then
      row = row + 1
    end if
    matrix(row, col) = k
  end do
```

```
  write (unit = rowformat, fmt = "(i16)", iostat = ios)
matrixOrder*matrixOrder
  k = len_trim(adjustl(rowformat)) + 3
  write (unit = rowformat, fmt = "(a1, i4, a1, i2, a1)", iostat =
ios) &
       "(", matrixOrder, "I", k, ")"

  do k = 1, matrixOrder
    write (unit = *, fmt = rowformat, iostat = ios) matrix(k,
1:matrixOrder)
  end do

end program magicSquares
```

برنامه 30: نمایش تفاوت داده ها

```
c ------------------------------------------------------------------
c Show how the same set of bits can be intepreted differently
c   types of variables
c Instructor: Nam Sun Wang
c ------------------------------------------------------------------
      character  a*4
      integer*2  i2
      real*8     x8
      complex    complx
      logical    logic
      equivalence (a, i2, i4, x4, x8, complx, logic)

c A "magic" number in decimal, hexadecimal, and binary notation
c     i4 = 1735287127
c     i4 = #676E6157
      i4 = 2#01100111011011100110000101010111

      print *, 'Interpretation of 01100111011011100110000101010111'
      print *, 'As a character:      ', a
      print *, 'As a 2-byte integer: ', i2
      print *, 'As a 4-byte integer: ', i4
      print *, 'As a 4-byte real:    ', x4
      print *, 'As a 8-byte real:    ', x8
      print *, 'As a 8-byte complex: ', complx
      print *, 'As a 4-byte logical: ', logic

      end
```

مثالهای آنالیز عددی

برنامه 1 : یافتن ریشه های معادله به کمک روش نصف کردن

```
PROGRAM BISECTION
!
! This program uses the bisection method to find the root of
! f(x)=exp(x)*ln(x)-x*x=0.
```

```
  IMPLICIT NONE
  INTEGER :: ISTEP
  REAL :: A,B,DL,DX,X0,X1,F,FX
!
  DL = 1.0E-06
  A  = 1.0
  B  = 2.0
  DX = B - A
  ISTEP = 0
  DO WHILE (ABS(DX).GT.DL)
    X0 = (A+B)/2.0
    IF ((FX(A)*FX(X0)).LT.0) THEN
      B  = X0
      DX = B-A
    ELSE
      A  = X0
      DX = B-A
    END IF
    ISTEP = ISTEP+1
  END DO
  WRITE (6,"(I4,2F16.8)") ISTEP,X0,DX
END PROGRAM BISECTION
!
FUNCTION FX(X) RESULT (F)
  IMPLICIT NONE
  REAL :: F
  REAL, INTENT (IN) :: X
!
  F = EXP(X)*ALOG(X)-X*X
END FUNCTION FX
```

برنامه 2: یافتن ریشه های معادله به کمک روش نیوتن

```
PROGRAM NEWTON
!
! This program uses the Newton method to find the root of
! f(x)=exp(x)*ln(x)-x*x=0.
!
  IMPLICIT NONE
  INTEGER :: ISTEP
  REAL :: A,B,DL,DX,X0,X1,F,FX,DF,DFX
!
  DL = 1.0E-06
  A  = 1.0
  B  = 2.0
  DX = B-A
  X0 = (A+B)/2.0
  ISTEP = 0
  DO WHILE (ABS(DX).GT.DL)
    X1 = X0-FX(X0)/DFX(X0)
    DX = X1-X0
    X0 = X1
    ISTEP = ISTEP+1
  END DO
  WRITE (6,"(I4,2F16.8)") ISTEP,X0,DX
END PROGRAM NEWTON
!
FUNCTION FX(X) RESULT (F)
  IMPLICIT NONE
```

```
  REAL :: F
  REAL, INTENT (IN) :: X
!
  F = EXP(X)*ALOG(X)-X*X
END FUNCTION FX
!
FUNCTION DFX (X) RESULT (DF)
  IMPLICIT NONE
  REAL :: DF
  REAL, INTENT (IN) :: X
!
  DF = EXP(X)*(ALOG(X)+1.0/X)-2.0*X
END FUNCTION DFX
```

برنامه 3 : یافتن ریشه های معادله به کمک روش سکانت

```
PROGRAM ROOT
!
! Main program to use the Secant Method to find the root of
! f(x)=exp(x)*ln(x)-x*x=0.
!
  IMPLICIT NONE
  INTEGER :: ISTEP
  REAL :: A,B,DL,DX,X0
!
  DL = 1.0E-06
  A  = 1.0
  B  = 2.0
  DX = (B-A)/10.0
  X0 = (A+B)/2.0
  CALL SECANT (DL,X0,DX,ISTEP)

  WRITE (6,"(I4,2F16.8)") ISTEP,X0,DX
END PROGRAM ROOT
!
SUBROUTINE SECANT (DL,X0,DX,ISTEP)
!
! Subroutine for the root of f(x)=0 with the secant method.

  IMPLICIT NONE
  INTEGER, INTENT (INOUT) :: ISTEP
  REAL, INTENT (INOUT) :: X0,DX
  REAL :: X1,X2,D,F,FX
  REAL, INTENT (IN) :: DL
!
  ISTEP = 0
  X1 = X0+DX
  DO WHILE (ABS(DX).GT.DL)
    D  = FX(X1)-FX(X0)
    X2 = X1-FX(X1)*(X1-X0)/D
    X0 = X1
    X1 = X2
    DX = X1-X0
    ISTEP = ISTEP+1
  END DO
END SUBROUTINE SECANT
!
FUNCTION FX(X) RESULT (F)
  IMPLICIT NONE
```

```
  REAL :: F
  REAL, INTENT (IN) :: X
!
  F = EXP(X)*ALOG(X)-X*X
END FUNCTION FX
```

برنامه 4: محاسبه انتگرال به روش سیمپسون

```
PROGRAM INTEGRAL
!
! Main program for evaluation of an integral with integrand
! sin(x) in the region of [0,pi/2].

  IMPLICIT NONE
  INTEGER, PARAMETER :: N=9
  INTEGER :: I
  REAL :: PI,H,S
  REAL, DIMENSION (N) :: X,F
!
  PI = 4.0*ATAN(1.0)
  H  = PI/(2*(N-1))
  DO I = 1, N
    X(I) = H*(I-1)
    F(I) = SIN(X(I))
  END DO
  CALL SIMP (N,H,F,S)
  WRITE (6, "(F16.8)") S
END PROGRAM INTEGRAL
!
SUBROUTINE SIMP (N,H,FI,S)
!
! Subroutine for integration over f(x) with the Simpson rule.  FI:
! integrand f(x); H: interval; S: integral.

  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL, INTENT (IN) :: H
  REAL :: S0,S1,S2
  REAL, INTENT (OUT) :: S
  REAL, INTENT (IN), DIMENSION (N) :: FI
!
  S  = 0.0
  S0 = 0.0
  S1 = 0.0
  S2 = 0.0
  DO I = 2, N-1, 2
    S1 = S1+FI(I-1)
    S0 = S0+FI(I)
    S2 = S2+FI(I+1)
  END DO
  S = H*(S1+4.0*S0+S2)/3.0
!
! If N is even, add the last slice separately
!
  IF (MOD(N,2).EQ.0) S = S &
     +H*(5.0*FI(N)+8.0*FI(N-1)-FI(N-2))/12.0
END SUBROUTINE SIMP
```

برنامه 5: محاسبه دترمینان ماتریس

```
SUBROUTINE DTRM (A,N,D,INDX)
!
! Subroutine for evaluating the determinant of a matrix using
! the partial-pivoting Gaussian elimination scheme.

  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I,J,MSGN
  INTEGER, INTENT (OUT), DIMENSION (N) :: INDX
  REAL, INTENT (OUT) :: D
  REAL, INTENT (INOUT), DIMENSION (N,N) :: A
!
  CALL ELGS(A,N,INDX)
!
  D = 1.0
  DO I = 1, N
    D = D*A(INDX(I),I)
  END DO
!
  MSGN = 1
  DO I = 1, N
    DO WHILE (I.NE.INDX(I))
          MSGN = -MSGN
          J = INDX(I)
          INDX(I) = INDX(J)
          INDX(J) = J
    END DO
  END DO
  D = MSGN*D
END SUBROUTINE DTRM
!
SUBROUTINE ELGS (A,N,INDX)
!
! Subroutine to perform the partial-pivoting Gaussian elimination.
! A(N,N) is the original matrix in the input and transformed matrix
! plus the pivoting element ratios below the diagonal in the output.
! INDX(N) records the pivoting order.
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I,J,K,ITMP
  INTEGER, INTENT (OUT), DIMENSION (N) :: INDX
  REAL :: C1,PI,PI1,PJ
  REAL, INTENT (INOUT), DIMENSION (N,N) :: A
  REAL, DIMENSION (N) :: C
!
! Initialize the index
!
  DO I = 1, N
    INDX(I) = I
  END DO
!
! Find the rescaling factors, one from each row
!
  DO I = 1, N
    C1= 0.0
    DO J = 1, N
```

```
      C1 = AMAX1(C1,ABS(A(I,J)))
    END DO
    C(I) = C1
  END DO
!
! Search the pivoting (largest) element from each column
!
  DO J = 1, N-1
    PI1 = 0.0
    DO I = J, N
      PI = ABS(A(INDX(I),J))/C(INDX(I))
      IF (PI.GT.PI1) THEN
        PI1 = PI
        K   = I
      ENDIF
    END DO
!
! Interchange the rows via INDX(N) to record pivoting order
!
    ITMP    = INDX(J)
    INDX(J) = INDX(K)
    INDX(K) = ITMP
    DO I = J+1, N
      PJ  = A(INDX(I),J)/A(INDX(J),J)
!
! Record pivoting ratios below the diagonal
!
      A(INDX(I),J) = PJ
!
! Modify other elements accordingly
!
      DO K = J+1, N
        A(INDX(I),K) = A(INDX(I),K)-PJ*A(INDX(J),K)
      END DO
    END DO
  END DO
!
END SUBROUTINE ELGS
```

برنامه 6: حل معادلات خطی

```
PROGRAM EX43
! An example of solving linear equation set A(N,N)*X(N) = B(N)
! with the partial-pivoting Gaussian elimination scheme.  The
! numerical values are for the Wheatstone bridge example discussed
! in Section 4.1 in the book with all resistances being 100 ohms
! and the voltage 200 volts.
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=3
  INTEGER :: I,J
  INTEGER, DIMENSION (N) :: INDX
  REAL, DIMENSION (N) :: X,B
  REAL, DIMENSION (N,N) :: A
  DATA B /200.0,0.0,0.0/, &
       ((A(I,J), J=1,N),I=1,N) /100.0,100.0,100.0,-100.0, &
                               300.0,-100.0,-100.0,-100.0, 300.0/
!
  CALL LEGS (A,N,B,X,INDX)
!
```

```
  WRITE (6, "(F16.8)") (X(I), I=1,N)
END PROGRAM EX43


SUBROUTINE LEGS (A,N,B,X,INDX)
!
! Subroutine to solve the equation A(N,N)*X(N) = B(N) with the
! partial-pivoting Gaussian elimination scheme.
! Copyright (c) Tao Pang 2001.
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I,J
  INTEGER, INTENT (OUT), DIMENSION (N) :: INDX
  REAL, INTENT (INOUT), DIMENSION (N,N) :: A
  REAL, INTENT (INOUT), DIMENSION (N) :: B
  REAL, INTENT (OUT), DIMENSION (N) :: X
!
  CALL ELGS (A,N,INDX)
!
  DO I = 1, N-1
    DO J = I+1, N
      B(INDX(J)) = B(INDX(J))-A(INDX(J),I)*B(INDX(I))
    END DO
  END DO
!
  X(N) = B(INDX(N))/A(INDX(N),N)
  DO I = N-1, 1, -1
    X(I) = B(INDX(I))
    DO J = I+1, N
      X(I) = X(I)-A(INDX(I),J)*X(J)
    END DO
    X(I) =  X(I)/A(INDX(I),I)
  END DO
!
END SUBROUTINE LEGS
!
SUBROUTINE ELGS (A,N,INDX)
!
! Subroutine to perform the partial-pivoting Gaussian elimination.
! A(N,N) is the original matrix in the input and transformed matrix
! plus the pivoting element ratios below the diagonal in the output.
! INDX(N) records the pivoting order.

  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I,J,K,ITMP
  INTEGER, INTENT (OUT), DIMENSION (N) :: INDX
  REAL :: C1,PI,PI1,PJ
  REAL, INTENT (INOUT), DIMENSION (N,N) :: A
  REAL, DIMENSION (N) :: C
!
! Initialize the index
!
  DO I = 1, N
    INDX(I) = I
  END DO
!
! Find the rescaling factors, one from each row
!
  DO I = 1, N
```

```
      C1= 0.0
      DO J = 1, N
        C1 = AMAX1(C1,ABS(A(I,J)))
      END DO
      C(I) = C1
    END DO
!
! Search the pivoting (largest) element from each column
!
    DO J = 1, N-1
      PI1 = 0.0
      DO I = J, N
        PI = ABS(A(INDX(I),J))/C(INDX(I))
        IF (PI.GT.PI1) THEN
          PI1 = PI
          K   = I
        ENDIF
      END DO
!
! Interchange the rows via INDX(N) to record pivoting order
!
      ITMP    = INDX(J)
      INDX(J) = INDX(K)
      INDX(K) = ITMP
      DO I = J+1, N
        PJ  = A(INDX(I),J)/A(INDX(J),J)
!
! Record pivoting ratios below the diagonal
!
        A(INDX(I),J) = PJ
!
! Modify other elements accordingly
!
        DO K = J+1, N
          A(INDX(I),K) = A(INDX(I),K)-PJ*A(INDX(J),K)
        END DO
      END DO
    END DO
!
END SUBROUTINE ELGS
```

برنامه 7: معکوس یک ماتریس

```
SUBROUTINE MIGS (A,N,X,INDX)
!
! Subroutine to invert matrix A(N,N) with the inverse stored
! in X(N,N) in the output.  Copyright (c) Tao Pang 2001.
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I,J,K
  INTEGER, INTENT (OUT), DIMENSION (N) :: INDX
  REAL, INTENT (IN), DIMENSION (N,N):: A
  REAL, INTENT (OUT), DIMENSION (N,N):: X
  REAL, DIMENSION (N,N) :: B
!
  DO I = 1, N
    DO J = 1, N
      B(I,J) = 0.0
    END DO
```

```
  END DO
  DO I = 1, N
    B(I,I) = 1.0
  END DO
!
  CALL ELGS (A,N,INDX)
!
  DO I = 1, N-1
    DO J = I+1, N
      DO K = 1, N
        B(INDX(J),K) = B(INDX(J),K)-A(INDX(J),I)*B(INDX(I),K)
      END DO
    END DO
  END DO
!
  DO I = 1, N
    X(N,I) = B(INDX(N),I)/A(INDX(N),N)
    DO J = N-1, 1, -1
      X(J,I) = B(INDX(J),I)
      DO K = J+1, N
        X(J,I) = X(J,I)-A(INDX(J),K)*X(K,I)
      END DO
      X(J,I) =  X(J,I)/A(INDX(J),J)
    END DO
  END DO
END SUBROUTINE MIGS
!
SUBROUTINE ELGS (A,N,INDX)
!
! Subroutine to perform the partial-pivoting Gaussian elimination.
! A(N,N) is the original matrix in the input and transformed matrix
! plus the pivoting element ratios below the diagonal in the output.
! INDX(N) records the pivoting order.  Copyright (c) Tao Pang 2001.
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I,J,K,ITMP
  INTEGER, INTENT (OUT), DIMENSION (N) :: INDX
  REAL :: C1,PI,PI1,PJ
  REAL, INTENT (INOUT), DIMENSION (N,N) :: A
  REAL, DIMENSION (N) :: C
!
! Initialize the index
!
  DO I = 1, N
    INDX(I) = I
  END DO
!
! Find the rescaling factors, one from each row
!
  DO I = 1, N
    C1= 0.0
    DO J = 1, N
      C1 = AMAX1(C1,ABS(A(I,J)))
    END DO
    C(I) = C1
  END DO
!
! Search the pivoting (largest) element from each column
!
  DO J = 1, N-1
```

```
      PI1 = 0.0
      DO I = J, N
        PI = ABS(A(INDX(I),J))/C(INDX(I))
        IF (PI.GT.PI1) THEN
          PI1 = PI
          K   = I
        ENDIF
      END DO
!
! Interchange the rows via INDX(N) to record pivoting order
!
      ITMP    = INDX(J)
      INDX(J) = INDX(K)
      INDX(K) = ITMP
      DO I = J+1, N
        PJ  = A(INDX(I),J)/A(INDX(J),J)
!
! Record pivoting ratios below the diagonal
!
        A(INDX(I),J) = PJ
!
! Modify other elements accordingly
!
        DO K = J+1, N
          A(INDX(I),K) = A(INDX(I),K)-PJ*A(INDX(J),K)
        END DO
      END DO
    END DO
!
END SUBROUTINE ELGS
```

برنامه 8 : مشتق تابع

```
PROGRAM DERIVATIVES
!
! Main program for derivatives of f(x) = sin(x).  F1: f';
! F2: f";  D1: error in f'; and D2: error in f".
!
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=101
  INTEGER :: I
  REAL :: PI,H
  REAL, DIMENSION (N) :: X,F,F1,D1,F2,D2
!
  PI = 4.0*ATAN(1.0)
  H  = PI/(2*100)
  DO I = 1, N
    X(I) = H*(I-1)
    F(I) = SIN(X(I))
  END DO
  CALL THREE(N,H,F,F1,F2)
  DO I = 1, N
    D1(I) = F1(I)-COS(X(I))
    D2(I) = F2(I)+SIN(X(I))
    WRITE (6, "(5F10.6)") X(I),F1(I),D1(I),F2(I),D2(I)
  END DO
END PROGRAM DERIVATIVES
```

```
!
SUBROUTINE THREE (N,H,FI,F1,F2)
!
! Subroutine for 1st and 2nd order derivatives with the three-point
! formulas. Extrapolations are made at the boundaries.  FI: input
! f(x); H: interval; F1: f'; and F2: f".
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL, INTENT (IN) :: H
  REAL, INTENT (IN), DIMENSION (N) :: FI
  REAL, INTENT (OUT), DIMENSION (N) :: F1,F2
!
! f' and f" from three-point formulas
!
  DO I = 2, N-1
    F1(I) = (FI(I+1)-FI(I-1))/(2.*H)
    F2(I) = (FI(I+1)-2.0*FI(I)+FI(I-1))/(H*H)
  END DO
!
! Linear extrapolation for the boundary points
!
  F1(1) = 2.0*F1(2)-F1(3)
  F1(N) = 2.0*F1(N-1)-F1(N-2)
  F2(1) = 2.0*F2(2)-F2(3)
  F2(N) = 2.0*F2(N-1)-F2(N-2)
END SUBROUTINE THREE
```

برنامه 9: مسائل مقدار مرزی – روش سکانت و رانج کاتا

```
PROGRAM SHOOTING
!
! Program for the boundary value problem with the shooting
! method.  The Runge-Kutta and secant methods are used.

  IMPLICIT NONE
  INTEGER, PARAMETER :: N=101,M=5
  REAL :: DK11,DK21,DK12,DK22,DK13,DK23,DK14,DK24
  REAL :: DL,XL,XU,H,D,YL,YU,X0,DX,X1,X2,F0,F1
  REAL :: Y1,Y2,G1,G1F,G2,G2F
  REAL, DIMENSION (2,N) :: Y
!
! Initialization of the problem
!
  DL = 1.0E-06
  XL = 0.0
  XU = 1.0
  H  = (XU-XL)/(N-1)
  D  = 0.1
  YL = 0.0
  YU = 1.0
  X0 = (YU-YL)/(XU-XL)
  DX = 0.01
  X1 = X0+DX
!
! The secant search for the root
!
  Y(1,1) = YL
```

```
  DO WHILE (ABS(D).GT.DL)
!
C The!Runge-Kutta calculation of the first trial solution
!
    Y(2,1) = X0
    DO I = 1, N-1
      X  = XL+H*I
      Y1 = Y(1,I)
      Y2 = Y(2,I)
      DK11 = H*G1F(Y1,Y2,X)
      DK21 = H*G2F(Y1,Y2,X)
      DK12 = H*G1F((Y1+DK11/2.0),(Y2+DK21/2.0),(X+H/2.0))
      DK22 = H*G2F((Y1+DK11/2.0),(Y2+DK21/2.0),(X+H/2.0))
      DK13 = H*G1F((Y1+DK12/2.0),(Y2+DK22/2.0),(X+H/2.0))
      DK23 = H*G2F((Y1+DK12/2.0),(Y2+DK22/2.0),(X+H/2.0))
      DK14 = H*G1F((Y1+DK13),(Y2+DK23),(X+H))
      DK24 = H*G2F((Y1+DK13),(Y2+DK23),(X+H))
      Y(1,I+1) = Y(1,I)+(DK11+2.0*(DK12+DK13)+DK14)/6.0
      Y(2,I+1) = Y(2,I)+(DK21+2.0*(DK22+DK23)+DK24)/6.0
    END DO
    F0 = Y(1,N)-1.0
!
! Runge-Kutta calculation of the second trial solution
!
    Y(2,1) = X1
    DO I = 1, N-1
      X  = XL+H*I
      Y1 = Y(1,I)
      Y2 = Y(2,I)
      DK11 = H*G1(Y1,Y2,X)
      DK21 = H*G2(Y1,Y2,X)
      DK12 = H*G1((Y1+DK11/2.0),(Y2+DK21/2.0),(X+H/2.0))
      DK22 = H*G2((Y1+DK11/2.0),(Y2+DK21/2.0),(X+H/2.0))
      DK13 = H*G1((Y1+DK12/2.0),(Y2+DK22/2.0),(X+H/2.0))
      DK23 = H*G2((Y1+DK12/2.0),(Y2+DK22/2.0),(X+H/2.0))
      DK14 = H*G1((Y1+DK13),(Y2+DK23),(X+H))
      DK24 = H*G2((Y1+DK13),(Y2+DK23),(X+H))
      Y(1,I+1) = Y(1,I)+(DK11+2.0*(DK12+DK13)+DK14)/6.0
      Y(2,I+1) = Y(2,I)+(DK21+2.0*(DK22+DK23)+DK24)/6.0
    END DO
    F1 = Y(1,N)-1.0
!
    D  = F1-F0
    X2 = X1-F1*(X1-X0)/D
    X0 = X1
    X1 = X2
  END DO
  WRITE (6,"(2F16.8)") (H*(I-1), Y(1,I),I=1,N,M)
END
!
FUNCTION G1F (Y1,Y2,T) RESULT (G1)
  IMPLICIT NONE
  REAL :: Y1,Y2,T,G1
!
   G1 = Y2
END FUNCTION G1F
!
FUNCTION G2F (Y1,Y2,T) RESULT (G2)
  IMPLICIT NONE
  REAL :: PI,Y1,Y2,T,G2
!
```

```
  PI = 4.0*ATAN(1.0)
  G2 =-PI*PI*(Y1+1.0)/4.0
END FUNCTION G2F
```

برنامه 10 : دترمینان چندجمله ای

```
SUBROUTINE TDPL(A,B,N,X,P)
!
! Subroutine to generate determinant polynomial P_N(X).
!
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL, INTENT (IN) :: X
  REAL :: P0
  REAL, INTENT (IN), DIMENSION (N) :: A,B
  REAL, INTENT (OUT), DIMENSION (N) :: P
!
  P0  = 1.0
  IF (N.LT.1) STOP 'The dimension is less than 1.'
  P(1) = A(1)-X
  IF (N.GE.2) P(2) = (A(2)-X)*P(1)-B(1)*B(1)*P0
  IF (N.GE.3) THEN
    DO I = 2, N-1
      P(I+1) = (A(I+1)-X)*P(I)-B(I)*B(I)*P(I-1)
    END DO
  END IF
END SUBROUTINE TDPL
```

برنامه 11 : اتگرال تابع

```
MODULE CSEED
  INTEGER :: ISEED
END MODULE CSEED
!
PROGRAM MCDS
!
! Integration with the direct sampling Monte Carlo scheme.  The
integrand
! is f(x) = x*x.
!
  USE CSEED
  IMPLICIT NONE
  INTEGER, PARAMETER :: M=1000000
  INTEGER :: time,STIME,I
  INTEGER, DIMENSION (9) :: T
  REAL :: SUM1,SUM2,S,DS,X,F,FX,R,RANF
!
! Initial seed from the system time and forced to be odd
!
  STIME = time(%REF(0))
  CALL gmtime(STIME,T)
  ISEED = T(6)+70*(T(5)+12*(T(4)+31*(T(3)+23*(T(2)+59*T(1)))))
  IF (MOD(ISEED,2).EQ.0) ISEED = ISEED-1
!
  SUM1 = 0.0
```

```
  SUM2 = 0.0
  DO I = 1, M
    X = RANF()
    SUM1 = SUM1+FX(X)
    SUM2 = SUM2+FX(X)**2
  END DO
  S  = SUM1/M
  DS = SQRT(ABS(SUM2/M-(SUM1/M)**2)/M)
  WRITE(6,"(2F16.8)") S,DS
END PROGRAM MCDS
!
FUNCTION FX(X) RESULT (F)
  IMPLICIT NONE
  REAL :: X,F
!
  F = X*X
END FUNCTION FX
!
FUNCTION RANF() RESULT (R)
!
! Uniform random number generator x(n+1) = a*x(n) mod c with
! a=7**5 and c = 2**(31)-1.
!
  USE CSEED
  IMPLICIT NONE
  INTEGER :: IH,IL,IT,IA,IC,IQ,IR
  DATA IA/16807/,IC/2147483647/,IQ/127773/,IR/2836/
  REAL :: R
!
  IH = ISEED/IQ
  IL = MOD(ISEED,IQ)
  IT = IA*IL-IR*IH
  IF(IT.GT.0) THEN
    ISEED = IT
  ELSE
    ISEED = IC+IT
  END IF
  R = ISEED/FLOAT(IC)
END FUNCTION RANF
```

برنامه 12:

```
SUBROUTINE RLXN (FN,DN,S,N,P,H)
!
! Subroutine performing one iteration of Relaxation for the one-
dimensional
! stationary diffusion equation. DN is the diffusion coefficient
shifted
! half step towards x=0.
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL, INTENT (IN) :: H,P
  REAL :: H2,Q
  REAL, INTENT (IN), DIMENSION (N) :: DN,S
  REAL, INTENT (INOUT), DIMENSION (N) :: FN
!
  H2 = H*H
```

```
  Q  = 1.0-P
  DO I = 2, N-1
    FN(I) = Q*FN(I)+P*(DN(I+1)*FN(I+1)+DN(I)*FN(I-
1)+H2*S(I))/(DN(I+1)+DN(I))
  END DO
END SUBROUTINE RLXN
```

برنامه 13 : محاسبه طول پیوند سدیم کلرید

```
MODULE CB
  REAL :: E2,A0,R0
END MODULE CB
!
PROGRAM BOND
!
! Main program to calculate the bond length of NaCl.
!
!
  USE CB
  IMPLICIT NONE
  INTEGER :: ISTEP
  REAL :: DL,X0,DX
!
  A0 = 1090.0
  R0 = 0.33
  E2 = 14.4
  DL = 1.0E-06
  X0 = 2.0
  DX = 0.1
  CALL M_SECANT (DL,X0,DX,ISTEP)
  WRITE (6,"(I4,2F16.8)") ISTEP,X0,DX
END PROGRAM BOND
!
SUBROUTINE M_SECANT (DL,X0,DX,ISTEP)
!
! Subroutine for the root of f(x) = dg(x)/dx = 0 with the
! secant method with the search toward the maximum of g(x).
!
!
  IMPLICIT NONE
  INTEGER, INTENT (OUT) :: ISTEP
  REAL, INTENT (IN) :: DL
  REAL, INTENT (INOUT) :: X0,DX
  REAL :: G0,G1,G2,X1,X2,D,G,GX,F,FX
!
  ISTEP = 0
  G0 = GX(X0)
  X1 = X0+DX
  G1 = GX(X1)
  IF(G1.LT.G0) X1 = X0-DX
  DO WHILE (ABS(DX).GT.DL)
    D  = FX(X1)-FX(X0)
    DX = -(X1-X0)*FX(X1)/D
    X2 = X1+DX
    G2 = GX(X2)
    IF(G2.LT.G1) X2 = X1-DX
    X0 = X1
    X1 = X2
    G1 = G2
```

```
    ISTEP = ISTEP+1
  END DO
  X0 = X2
END SUBROUTINE M_SECANT
!
FUNCTION GX(X) RESULT(G)
  USE CB
  IMPLICIT NONE
  REAL :: X,G
!
  G  = E2/X-A0*EXP(-X/R0)
END FUNCTION GX
!
FUNCTION FX(X) RESULT(F)
  USE CB
  IMPLICIT NONE
  REAL :: X,F
!
  F  = E2/(X*X)-A0*EXP(-X/R0)/R0
END FUNCTION FX
```

برنامه 14 :

```
MODULE CSEED
  INTEGER ISEED
END MODULE CSEED
!
SUBROUTINE RMSG (N,XS,A)
!
! Subroutine for generating a random matrix in the Gaussian
! orthogonal ensemble with XS as the standard deviation of
! the off-diagonal elements.
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I,J
  REAL, INTENT (IN) :: XS
  REAL :: G1,G2
  REAL, INTENT (OUT), DIMENSION (N,N) :: A
!
  DO I = 1, N
    CALL GRNF (G1,G2)
    A(I,I) = SQRT(2.0)*G1*XS
  END DO
!
  DO I = 1, N
    DO J = I+1, N
      CALL GRNF(G1,G2)
      A(I,J) = G1*XS
      A(J,I) = A(I,J)
    END DO
  END DO
END SUBROUTINE RMSG
!
SUBROUTINE GRNF (X,Y)
!
! Two Gaussian random numbers generated from two uniform random
! numbers.
!
```

```
  IMPLICIT NONE
  REAL, INTENT (OUT) :: X,Y
  REAL :: PI,R1,R2,R,RANF
!
  PI = 4.0*ATAN(1.0)
  R1 = -ALOG(1.0-RANF())
  R2 = 2.0*PI*RANF()
  R1 = SQRT(2.0*R1)
  X  = R1*COS(R2)
  Y  = R1*SIN(R2)
END SUBROUTINE GRNF
!
FUNCTION RANF() RESULT (R)
!
! Uniform random number generator x(n+1) = a*x(n) mod c with
! a=7**5 and c = 2**(31)-1.
!
  USE CSEED
  IMPLICIT NONE
  INTEGER :: IH,IL,IT,IA,IC,IQ,IR
  DATA IA/16807/,IC/2147483647/,IQ/127773/,IR/2836/
  REAL :: R
!
  IH = ISEED/IQ
  IL = MOD(ISEED,IQ)
  IT = IA*IL-IR*IH
  IF(IT.GT.0) THEN
    ISEED = IT
  ELSE
    ISEED = IC+IT
  END IF
  R = ISEED/FLOAT(IC)
END FUNCTION RANF
```

برنامه 15:

```
SUBROUTINE ELGS (A,N,INDX)
!
! Subroutine to perform the partial-pivoting Gaussian elimination.
! A(N,N) is the original matrix in the input and transformed matrix
! plus the pivoting element ratios below the diagonal in the output.
! INDX(N) records the pivoting order.  Copyright (c) Tao Pang 2001.
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I,J,K,ITMP
  INTEGER, INTENT (OUT), DIMENSION (N) :: INDX
  REAL :: C1,PI,PI1,PJ
  REAL, INTENT (INOUT), DIMENSION (N,N) :: A
  REAL, DIMENSION (N) :: C
!
! Initialize the index
!
  DO I = 1, N
    INDX(I) = I
  END DO
!
! Find the rescaling factors, one from each row
```

```
!
  DO I = 1, N
    C1= 0.0
    DO J = 1, N
      C1 = AMAX1(C1,ABS(A(I,J)))
    END DO
    C(I) = C1
  END DO
!
! Search the pivoting (largest) element from each column
!
  DO J = 1, N-1
    PI1 = 0.0
    DO I = J, N
      PI = ABS(A(INDX(I),J))/C(INDX(I))
      IF (PI.GT.PI1) THEN
        PI1 = PI
        K   = I
      ENDIF
    END DO
!
! Interchange the rows via INDX(N) to record pivoting order
!
    ITMP    = INDX(J)
    INDX(J) = INDX(K)
    INDX(K) = ITMP
    DO I = J+1, N
      PJ  = A(INDX(I),J)/A(INDX(J),J)
!
! Record pivoting ratios below the diagonal
!
      A(INDX(I),J) = PJ
!
! Modify other elements accordingly
!
      DO K = J+1, N
        A(INDX(I),K) = A(INDX(I),K)-PJ*A(INDX(J),K)
      END DO
    END DO
  END DO
!
END SUBROUTINE ELGS
```

برنامه 16:

```
SUBROUTINE FFT2D (FR,FI,N1,N2,M1,M2)
!
! Subroutine for the two-dimensional fast Fourier transform
! with N=N1*N2 and N1=2**M1 and N2=2**M2.
!
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N1,N2,M1,M2
  INTEGER :: I,J
  REAL, INTENT (INOUT), DIMENSION  (N1,N2) :: FR,FI
!
! Transformation on the second index
!
  DO I = 1, N1
```

```
      CALL FFT (FR(I,1),FI(I,1),N2,M2)
    END DO
!
! Transformation on the first index
!
    DO J = 1, N2
      CALL FFT (FR(1,J),FI(1,J),N1,M1)
    END DO
  END SUBROUTINE FFT2D
!
    SUBROUTINE FFT (AR,AI,N,M)
!
! An example of the fast Fourier transform subroutine with N = 2**M.
! AR and AI are the real and imaginary part of data in the input and
! corresponding Fourier coefficients in the output.
!
!
    IMPLICIT NONE
    INTEGER, INTENT (IN) :: N,M
    INTEGER :: N1,N2,I,J,K,L,L1,L2
    REAL :: PI,A1,A2,Q,U,V
    REAL, INTENT (INOUT), DIMENSION (N) :: AR,AI
!
    PI = 4.0*ATAN(1.0)
    N2 = N/2
!
    N1 = 2**M
    IF(N1.NE.N) STOP 'Indices do not match'
!
! Rearrange the data to the bit reversed order
!
    L = 1
    DO K = 1, N-1
      IF (K.LT.L) THEN
        A1    = AR(L)
        A2    = AI(L)
        AR(L) = AR(K)
        AR(K) = A1
        AI(L) = AI(K)
        AI(K) = A2
      END IF
      J   = N2
      DO WHILE (J.LT.L)
        L = L-J
        J = J/2
      END DO
      L = L+J
    END DO
!
! Perform additions at all levels with reordered data
!
    L2 = 1
    DO L = 1, M
      Q  =  0.0
      L1 =  L2
      L2 =  2*L1
      DO K = 1, L1
        U   =  COS(Q)
        V   = -SIN(Q)
        Q   =  Q + PI/L1
        DO J = K, N, L2
```

```
          I     =  J + L1
          A1    =  AR(I)*U-AI(I)*V
          A2    =  AR(I)*V+AI(I)*U
          AR(I) =  AR(J)-A1
          AR(J) =  AR(J)+A1
          AI(I) =  AI(J)-A2
          AI(J) =  AI(J)+A2
        END DO
      END DO
    END DO
  END SUBROUTINE FFT
```

برنامه 17 : چند جمله ای لژاندر

```
SUBROUTINE LGND (LMAX,X,P)
!
! Subroutine to generate Legendre polynomials P_L(X)
! for L = 0,1,...,LMAX with given X.
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: LMAX
  INTEGER :: L
  REAL, INTENT (IN) :: X
  REAL, INTENT (OUT), DIMENSION (0:LMAX) :: P
!
  P(0) = 1.0
  P(1) = X
  DO L = 1, LMAX-1
    P(L+1) = ((2.0*L+1)*X*P(L)-L*P(L-1))/(L+1)
  END DO
END SUBROUTINE LGND
```

برنامه 18 :

```
SUBROUTINE NMRV (N,H,Q,S,U)
!
! The Numerov algorithm for the equation u"(x)+q(x)u(x)=s(x)
! as given in Eqs. (3.77)-(3.80) in the book.
!
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL, INTENT (IN) :: H
  REAL :: G,C0,C1,C2,D,UTMP
  REAL, INTENT (IN), DIMENSION (N) :: Q,S
  REAL, INTENT (OUT), DIMENSION (N) :: U
!
  G = H*H/12.0
!
  DO I = 2, N-1
    C0 = 1.0+G*((Q(I-1)-Q(I+1))/2.0+Q(I))
    C1 = 2.0-G*(Q(I+1)+Q(I-1)+8.0*Q(I))
    C2 = 1.0+G*((Q(I+1)-Q(I-1))/2.0+Q(I))
    D  = G*(S(I+1)+S(I-1)+10.0*S(I))
    UTMP   = C1*U(I)-C0*U(I-1)+D
    U(I+1) = UTMP/C2
  END DO
```

```
END SUBROUTINE NMRV
```

برنامه 19:

```
PROGRAM MILL
!
! Program to fit the Millikan experimental data to a linear curve
! p(x) = a*x+b directly.  One can find a and b from partial D/partial
! a = 0 and partial D/partial b = 0 with D = sum (p(x_i)-f(x_i))**2.
! The result is a = (c1*c3-c4*n)/(c1**2-c2*n) and b = (c1*c4-c2*c3)
! /(c1**2-c2*n) with n being the number of points, c1 = sum x_i, c2
! = sum x_i**2, c3 = sum f(x_i), and c4 = sum x_i*f(x_i).
!
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=15
  INTEGER :: I
  REAL :: C1,C2,C3,C4,C,A,B
  REAL, DIMENSION (N) :: X,F
  DATA X /4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0, &
          12.0,13.0,14.0,15.0,16.0,17.0,18.0/
  DATA F /6.558,8.206,9.880,11.50,13.14,14.81,16.40,18.04, &
          19.68,21.32,22.96,24.60,26.24,27.88,29.52/
!
  C1 = 0.0
  C2 = 0.0
  C3 = 0.0
  C4 = 0.0
  DO I = 1, N
    C1 = C1+X(I)
    C2 = C2+X(I)*X(I)
    C3 = C3+F(I)
    C4 = C4+F(I)*X(I)
  END DO
  C = C1*C1-C2*N
  A = (C1*C3-C4*N)/C
  B = (C1*C4-C2*C3)/C
  WRITE (6, "('The fundamental charge is 'F6.4,'+-'F6.4)") A,ABS(B)
END PROGRAM MILL
```

برنامه 20:

```
PROGRAM INTERPOLATION2
!
! Main program for the Lagrange interpolation with the
! upward and downward correction method.
!
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=5
  REAL :: X,F,DF
  REAL, DIMENSION (N) :: XI,FI
  DATA XI/0.0,0.5,1.0,1.5,2.0/, &
       FI/1.0,0.938470,0.765198,0.511828,0.223891/
!
  X = 0.9
  CALL UPDOWN (N,XI,FI,X,F,DF)
  WRITE (6,"(3F16.8)") X,F,DF
END PROGRAM INTERPOLATION2
!
```

```
SUBROUTINE UPDOWN (N,XI,FI,X,F,DF)
!
! Subroutine performing the Lagrange interpolation with the
! upward and downward correction method.  F: interpolated
! value.  DF: error estimated.
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: NMAX=21
  INTEGER, INTENT (IN) :: N
  INTEGER :: I,J,I0,J0,IT,K
  REAL, INTENT (IN) :: X
  REAL, INTENT (OUT) :: F,DF
  REAL :: DX,DXT,DT
  REAL, INTENT (IN), DIMENSION (N) :: XI,FI
  REAL, DIMENSION (NMAX,NMAX) :: DP,DM
!
  IF (N.GT.NMAX) STOP 'Dimension of the data set is too large.'
    DX = ABS(XI(N)-XI(1))
    DO  I = 1, N
      DP(I,I) = FI(I)
      DM(I,I) = FI(I)
      DXT = ABS(X-XI(I))
      IF (DXT.LT.DX) THEN
        I0 = I
        DX = DXT
      END IF
    END DO
    J0 = I0
!
! Evaluate correction matrices
!
  DO I = 1, N-1
    DO J = 1, N-I
      K = J+I
      DT =(DP(J,K-1)-DM(J+1,K))/(XI(K)-XI(J))
      DP(J,K) = DT*(XI(K)-X)
      DM(J,K) = DT*(XI(J)-X)
    END DO
  END DO
!
! Update the approximation
!
  F = FI(I0)
  IT = 0
  IF(X.LT.XI(I0)) IT = 1
  DO I = 1, N-1
    IF ((IT.EQ.1).OR.(J0.EQ.N)) THEN
      I0 = I0-1
      DF = DP(I0,J0)
      F  = F+DF
      IT = 0
      IF (J0.EQ.N) IT = 1
    ELSE IF ((IT.EQ.0).OR.(I0.EQ.1)) THEN
      J0 = J0+1
      DF = DM(I0,J0)
      F  = F+DF
      IT = 1
      IF (I0.EQ.1) IT = 0
    END IF
  END DO
  DF = ABS(DF)
```

```
END SUBROUTINE UPDOWN
```

برنامه 21:

```
PROGRAM MILLIKAN
!
! Main program for a linear fit of the Millikan experimental
! data on the fundamental charge e_0 from e_n = e_0*n + de.
!
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=15,M=2
  INTEGER :: I
  REAL :: SUM0,SUMT,E0,DE
  REAL, DIMENSION (N) :: X,F
  REAL, DIMENSION (M) :: A
  REAL, DIMENSION (M,N) :: U
  DATA X /4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0, &
          12.0,13.0,14.0,15.0,16.0,17.0,18.0/
  DATA F /6.558,8.206,9.880,11.50,13.14,14.81,16.40,18.04, &
          19.68,21.32,22.96,24.60,26.24,27.88,29.52/
!
  CALL PFIT (N,M,X,F,A,U)
  SUM0 = 0.0
  SUMT = 0.0
  DO I = 1, N
    SUM0 = SUM0+U(1,I)**2
    SUMT = SUMT+X(I)*U(1,I)**2
  END DO
  E0   = A(2)
  DE   = A(1)-A(2)*SUMT/SUM0
  WRITE (6,"(2F16.8)") E0,DE
END PROGRAM MILLIKAN
!
SUBROUTINE PFIT (N,M,X,F,A,U)
!
! Subroutine generating orthonormal polynomials U(M,N) up to
! (M-1)th order and coefficients A(M), for the least squares
! approximation of the function F(N) at X(N). Other variables
! used: G(K) for g_k, H(K) for h_k, S(K) for <u_k|u_k>.
!
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: NMAX=101,MMAX=101
  INTEGER, INTENT (IN) :: N,M
  INTEGER :: I,J
  REAL :: TMP
  REAL, INTENT (IN), DIMENSION (N) :: X,F
  REAL, INTENT (OUT), DIMENSION (M) :: A
  REAL, INTENT (OUT), DIMENSION (M,N) :: U
  REAL, DIMENSION (MMAX) :: G,H,S
!
  IF(N.GT.NMAX) STOP 'Too many points'
  IF(M.GT.MMAX) STOP 'Order too high'
!
! Set up the zeroth order polynomial u_0
!
  DO I = 1, N
    U(1,I) = 1.0
  END DO
  DO I = 1, N
```

```
      TMP  = U(1,I)*U(1,I)
      S(1) = S(1)+TMP
      G(1) = G(1)+X(I)*TMP
      A(1) = A(1)+U(1,I)*F(I)
    END DO
    G(1) = G(1)/S(1)
    H(1) = 0.0
    A(1) = A(1)/S(1)
!
! Set up the first order polynomial u_1
!
    DO I = 1, N
      U(2,I) = X(I)*U(1,I)-G(1)*U(1,I)
      S(2)   = S(2)+U(2,I)**2
      G(2)   = G(2)+X(I)*U(2,I)**2
      H(2)   = H(2) + X(I)*U(2,I)*U(1,I)
      A(2)   = A(2)+U(2,I)*F(I)
    END DO
    G(2) = G(2)/S(2)
    H(2) = H(2)/S(1)
    A(2) = A(2)/S(2)
!
! Higher order polynomials u_k from the recursive relation
!
    IF(M.GE.3) THEN
      DO I = 2, M-1
        DO J = 1, N
          U(I+1,J) = X(J)*U(I,J)-G(I)*U(I,J)-H(I)*U(I-1,J)
          S(I+1)   = S(I+1) + U(I+1,J)**2
          G(I+1)   = G(I+1) + X(J)*U(I+1,J)**2
          H(I+1)   = H(I+1) + X(J)*U(I+1,J)*U(I,J)
          A(I+1)   = A(I+1) + U(I+1,J)*F(J)
        END DO
        G(I+1) = G(I+1)/S(I+1)
        H(I+1) = H(I+1)/S(I)
        A(I+1) = A(I+1)/S(I+1)
      END DO
    END IF
  END SUBROUTINE PFIT
```

برنامه 22:

```
  SUBROUTINE PFIT (N,M,X,F,A,U)
!
! Subroutine generating orthonormal polynomials U(M,N) up to
! (M-1)th order and coefficients A(M), for the least squares
! approximation of the function F(N) at X(N). Other variables
! used: G(K) for g_k, H(K) for h_k, S(K) for <u_k|u_k>.
!
!
    IMPLICIT NONE
    INTEGER, PARAMETER :: NMAX=101,MMAX=101
    INTEGER, INTENT (IN) :: N,M
    INTEGER :: I,J
    REAL :: TMP
    REAL, INTENT (IN), DIMENSION (N) :: X,F
    REAL, INTENT (OUT), DIMENSION (M) :: A
    REAL, INTENT (OUT), DIMENSION (M,N) :: U
    REAL, DIMENSION (MMAX) :: G,H,S
```

```
!
  IF(N.GT.NMAX) STOP 'Too many points'
  IF(M.GT.MMAX) STOP 'Order too high'
!
! Set up the zeroth order polynomial u_0
!
  DO I = 1, N
    U(1,I) = 1.0
  END DO
  DO I = 1, N
    TMP  = U(1,I)*U(1,I)
    S(1) = S(1)+TMP
    G(1) = G(1)+X(I)*TMP
    A(1) = A(1)+U(1,I)*F(I)
  END DO
  G(1) = G(1)/S(1)
  H(1) = 0.0
  A(1) = A(1)/S(1)
!
! Set up the first order polynomial u_1
!
  DO I = 1, N
    U(2,I) = X(I)*U(1,I)-G(1)*U(1,I)
    S(2)   = S(2)+U(2,I)**2
    G(2)   = G(2)+X(I)*U(2,I)**2
    H(2)   = H(2)+X(I)*U(2,I)*U(1,I)
    A(2)   = A(2)+U(2,I)*F(I)
  END DO
  G(2) = G(2)/S(2)
  H(2) = H(2)/S(1)
  A(2) = A(2)/S(2)
!
! Higher order polynomials u_k from the recursive relation
!
  IF(M.GE.3) THEN
    DO I = 2, M-1
      DO J = 1, N
        U(I+1,J) = X(J)*U(I,J)-G(I)*U(I,J)-H(I)*U(I-1,J)
        S(I+1)   = S(I+1) + U(I+1,J)**2
        G(I+1)   = G(I+1) + X(J)*U(I+1,J)**2
        H(I+1)   = H(I+1) + X(J)*U(I+1,J)*U(I,J)
        A(I+1)   = A(I+1) + U(I+1,J)*F(J)
      END DO
      G(I+1) = G(I+1)/S(I+1)
      H(I+1) = H(I+1)/S(I)
      A(I+1) = A(I+1)/S(I+1)
    END DO
  END IF
END SUBROUTINE PFIT
```

برنامه 23:

```
PROGRAM INTERPOLATION
!
! Main program for the Lagrange interpolation with the Aitken method.
!
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=5
  REAL :: X,F,DF
  REAL, DIMENSION (N) :: XI,FI
```

```
  DATA XI/0.0,0.5,1.0,1.5,2.0/, &
       FI/1.0,0.938470,0.765198,0.511828,0.223891/
!
  X = 0.9
  CALL AITKEN (N,XI,FI,X,F,DF)
  WRITE (6,"(3F16.8)") X,F,DF
END PROGRAM INTERPOLATION
!
SUBROUTINE AITKEN (N,XI,FI,X,F,DF)
!
! Subroutine performing the Lagrange interpolation with the
! Aitken method. F: interpolated value. DF: error estimated.
!
!
  INTEGER, PARAMETER :: NMAX=21
  INTEGER, INTENT (IN) :: N
  INTEGER :: I,J
  REAL, INTENT (IN) :: X
  REAL, INTENT (OUT) :: F,DF
  REAL :: X1,X2,F1,F2
  REAL, INTENT (IN), DIMENSION (N):: XI,FI
  REAL, DIMENSION (NMAX):: FT
!
  IF (N.GT.NMAX) STOP 'Dimension of the data is too large.'
  DO I = 1, N
    FT(I) = FI(I)
  END DO
!
  DO I = 1, N-1
    DO J = 1, N-I
      X1 = XI(J)
      X2 = XI(J+I)
      F1 = FT(J)
      F2 = FT(J+1)
      FT(J) = (X-X1)/(X2-X1)*F2+(X-X2)/(X1-X2)*F1
    END DO
  END DO
  F = FT(1)
  DF = (ABS(F-F1)+ABS(F-F2))/2.0
END SUBROUTINE AITKEN
```

برنامه 24:

```
MODULE CB
  REAL :: Q,B,W
END MODULE CB
!
PROGRAM PENDULUM
!
! Program for the power spectra of a driven pendulum under damping
with
! the fourth order Runge-Kutta algorithm. Given parameters: Q, B, and
W
! (omega_0).
!
  USE CB
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=65536,L=128,M=16,MD=16
  INTEGER :: I,J
  REAL :: PI,F1,H,OD,T,Y1,Y2,G1,GX1,G2,GX2
  REAL :: DK11,DK21,DK12,DK22,DK13,DK23,DK14,DK24
```

```
  REAL, DIMENSION (N) :: AR,AI,WR,WI,O
  REAL, DIMENSION (2,N) :: Y

!
  PI = 4.0*ATAN(1.0)
  F1 = 1.0/SQRT(FLOAT(N))
  W  = 2.0/3.0
  H  = 2.0*PI/(L*W)
  OD = 2.0*PI/(N*H*W*W)
  Q  = 0.5
  B  = 1.15
  Y(1,1) = 0.0
  Y(2,1) = 2.0
!
! Runge-Kutta algorithm to integrate the equation
!
  DO I = 1, N-1
    T  = H*I
    Y1 = Y(1,I)
    Y2 = Y(2,I)
    DK11 = H*GX1(Y1,Y2,T)
    DK21 = H*GX2(Y1,Y2,T)
    DK12 = H*GX1((Y1+DK11/2.0),(Y2+DK21/2.0),(T+H/2.0))
    DK22 = H*GX2((Y1+DK11/2.0),(Y2+DK21/2.0),(T+H/2.0))
    DK13 = H*GX1((Y1+DK12/2.0),(Y2+DK22/2.0),(T+H/2.0))
    DK23 = H*GX2((Y1+DK12/2.0),(Y2+DK22/2.0),(T+H/2.0))
    DK14 = H*GX1((Y1+DK13),(Y2+DK23),(T+H))
    DK24 = H*GX2((Y1+DK13),(Y2+DK23),(T+H))
    Y(1,I+1) = Y(1,I)+(DK11+2.0*(DK12+DK13)+DK14)/6.0
    Y(2,I+1) = Y(2,I)+(DK21+2.0*(DK22+DK23)+DK24)/6.0
!
! Bring theta back to region [-pi,pi]
!
     IF (ABS(Y(1,I+1)).GT.PI) THEN
       Y(1,I+1) = Y(1,I+1) - 2.*PI*ABS(Y(1,I+1))/Y(1,I+1)
     END IF
  END DO
!
  DO I = 1, N
    AR(I) = Y(1,I)
    WR(I) = Y(2,I)
    AI(I) = 0.0
    WI(I) = 0.0
  END DO
  CALL FFT (AR,AI,N,M)
  CALL FFT (WR,WI,N,M)
!
  DO I = 1, N
    O(I)  =  (I-1)*OD
    AR(I) =  (F1*AR(I))**2+(F1*AI(I))**2
    WR(I) =  (F1*WR(I))**2+(F1*WI(I))**2
    AR(I) = ALOG10(AR(I))
    WR(I) = ALOG10(WR(I))
  END DO
  WRITE(6,"(3F16.10)") (O(I),AR(I),WR(I),I=1,(L*MD),4)
END PROGRAM PENDULUM
!
  SUBROUTINE FFT (AR,AI,N,M)
!
! An example of the fast Fourier transform subroutine with N = 2**M.
! AR and AI are the real and imaginary part of data in the input and
```

```
! corresponding Fourier coefficients in the output.
!
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N,M
  INTEGER :: N1,N2,I,J,K,L,L1,L2
  REAL :: PI,A1,A2,Q,U,V
  REAL, INTENT (INOUT), DIMENSION (N) :: AR,AI
!
  PI = 4.0*ATAN(1.0)
  N2 = N/2
!
  N1 = 2**M
  IF(N1.NE.N) STOP 'Indices do not match'
!
! Rearrange the data to the bit reversed order
!
  L = 1
  DO K = 1, N-1
    IF (K.LT.L) THEN
      A1    = AR(L)
      A2    = AI(L)
      AR(L) = AR(K)
      AR(K) = A1
      AI(L) = AI(K)
      AI(K) = A2
    END IF
    J   = N2
    DO WHILE (J.LT.L)
      L = L-J
      J = J/2
    END DO
    L = L+J
  END DO
!
! Perform additions at all levels with reordered data
!
  L2 = 1
  DO L = 1, M
    Q  =  0.0
    L1 =  L2
    L2 =  2*L1
    DO K = 1, L1
      U   =  COS(Q)
      V   = -SIN(Q)
      Q   =  Q + PI/L1
      DO J = K, N, L2
        I     =  J + L1
        A1    =  AR(I)*U-AI(I)*V
        A2    =  AR(I)*V+AI(I)*U
        AR(I) =  AR(J)-A1
        AR(J) =  AR(J)+A1
        AI(I) =  AI(J)-A2
        AI(J) =  AI(J)+A2
      END DO
    END DO
  END DO
END SUBROUTINE FFT
!
FUNCTION GX1 (Y1,Y2,T) RESULT (G1)
!
```

```
  G1 = Y2
END FUNCTION GX1
!
FUNCTION GX2 (Y1,Y2,T) RESULT (G2)
  USE CB
!
  G2 = -Q*Y2-SIN(Y1)+B*COS(W*T)
END FUNCTION GX2
```

برنامه 25:

```
PROGRAM SCHR
!
! Main program for solving the eigenvalue problem of the
! one-dimensional Schroedinger equation.
!
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=501,M=5,IMAX=100
  INTEGER :: I,IM,NL,NR,ISTEP
  REAL :: DL,H2M,EA,EB,E,DE,XL0,XR0,H,C
  REAL :: XL,XR,FACT,F0,F1,E1,SUM,V,VX
  REAL, DIMENSION (N) :: UL,UR,QL,QR,S
!
  DL    =  1.0E-06
  H2M   =  0.5
  EA    =  2.4
  EB    =  2.7
  E     =  EA
  DE    =  0.1
  XL0   = -10.0
  XR0   =  10.0
  H     =  (XR0-XL0)/(N-1)
  C     =  1.0/H2M
  UL(1) =  0.0
  UL(2) =  0.01
  UR(1) =  0.0
  UR(2) =  0.01
!
! Set up the potential q(x) and source s(x)
!
  DO I = 1, N
    XL    = XL0+(I-1)*H
    XR    = XR0-(I-1)*H
    QL(I) = C*(E-VX(XL))
    QR(I) = C*(E-VX(XR))
    S(I)  = 0.0
  END DO
!
! Find the matching point at the right turning point
!
  DO I = 1, N-1
    IF(((QL(I)*QL(I+1)).LE.0).AND.(QL(I).GT.0)) IM = I
  END DO
!
! Numerov algorithm from left to right and vice versa
!
  NL = IM+1
  NR = N-IM+2
  CALL NMRV2 (NL,H,QL,S,UL)
  CALL NMRV2 (NR,H,QR,S,UR)
```

```
!
! Rescale the left solution
!
  FACT = UR(NR-1)/UL(IM)
  DO I = 1, NL
    UL(I) = FACT*UL(I)
  END DO
!
  F0 = UR(NR)+UL(NL)-UR(NR-2)-UL(NL-2)
  F0 = F0/(2.0*H*UR(NR-1))
!
! Bisection method for the root
!
  ISTEP = 0
  DO WHILE ((ABS(DE).GT.DL).AND.(ISTEP.LT.IMAX))
    E1 = E
    E  = (EA+EB)/2.0
    DO I = 1, N
      QL(I) = QL(I)+C*(E-E1)
      QR(I) = QR(I)+C*(E-E1)
    END DO
!
! Find the matching point at the right turning point
!
    DO I = 1, N-1
      IF(((QL(I)*QL(I+1)).LE.0).AND.(QL(I).GT.0)) IM = I
    END DO
!
! Numerov algorithm from left to right and vice versa
!
    NL = IM+1
    NR = N-IM+2
    CALL NMRV2 (NL,H,QL,S,UL)
    CALL NMRV2 (NR,H,QR,S,UR)
!
! Rescale the left solution
!
    FACT = UR(NR-1)/UL(IM)
    DO I = 1, NL
          UL(I) = FACT*UL(I)
    END DO
!
    F1 = UR(NR)+UL(NL)-UR(NR-2)-UL(NL-2)
    F1 = F1/(2.0*H*UR(NR-1))
!
    IF ((F0*F1).LT.0) THEN
      EB = E
      DE = EB-EA
    ELSE
      EA = E
      DE = EB-EA
      F0 = F1
    END IF
    ISTEP = ISTEP+1
  END DO
!
  SUM = 0.0
  DO I = 1, N
    IF(I.GT.IM) UL(I) = UR(N-I)
    SUM = SUM+UL(I)*UL(I)
  END DO
```

```
!
  WRITE(6,"(2I4)") ISTEP,IMAX
  WRITE(6,"(4F20.8)") E,DE,F0,F1
!
  SUM=SQRT(H*SUM)
  DO I = 1, N, M
    XL = XL0+(I-1)*H
    UL(I) = UL(I)/SUM
    WRITE(15,"(4F20.8)") XL,UL(I)
    WRITE(16,"(4F20.8)") XL,VX(XL)
  END DO
END PROGRAM SCHR
!
FUNCTION VX (X) RESULT (V)
  REAL :: A,B,X,V
!
  A = 1.0
  B = 4.0
  V = 3.0-A*A*B*(B-1.0)/(COSH(A*X)**2)/2.0
END FUNCTION VX
!
SUBROUTINE NMRV2 (N,H,Q,S,U)
!
! The Numerov algorithm for the equation u"(x)+q(x)u(x)=s(x)
! as given in Eqs. (3.82)-(3.85) in the book.
!
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL,INTENT (IN) :: H
  REAL :: G,C0,C1,C2,D,UTMP
  REAL, INTENT (IN), DIMENSION (N) :: Q,S
  REAL, INTENT (INOUT), DIMENSION (N) :: U
!
  G = H*H/12.0
!
  DO I = 2, N-1
    C0 = 1.0+G*Q(I-1)
    C1 = 2.0-10.0*G*Q(I)
    C2 = 1.0+G*Q(I+1)
    D  = G*(S(I+1)+S(I-1)+10.0*S(I))
    UTMP   = C1*U(I)-C0*U(I-1)+D
    U(I+1) = UTMP/C2
  END DO
END SUBROUTINE NMRV2
```

برنامه 26:

```
PROGRAM WAVE
!
! Program for the eigenvalue problem with a combination of the
! bisection method and the Numerov algorithm for u" = -k**2*u
! with boundary conditions u(0)=u(1)=0.
!
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=101
  INTEGER :: I,ISTEP
  REAL :: DL,H,AK,BK,DK,EK,F0,F1
  REAL, DIMENSION (N) :: Q,S,U
```

```
!
! Initialization of the problem
!
  DL = 1.0E-06
  H  = 1.0/(N-1)
  AK = 2.0
  BK = 4.0
  DK = 1.0
  EK = AK
  U(1) = 0.0
  U(2) = 0.01
  ISTEP = 0
!
  DO I = 1,N
    S(I) = 0.0
    Q(I) = EK*EK
  END DO
  CALL NMRV (N,H,Q,S,U)
  F0 = U(N)
!
! Bisection method for the root
!
  DO WHILE (ABS(DK).GT.DL)
    EK = (AK+BK)/2.0
    DO I = 1,N
      Q(I) = EK*EK
    END DO
    CALL NMRV (N,H,Q,S,U)
    F1 = U(N)
    IF ((F0*F1).LT.0) THEN
      BK = EK
      DK = BK-AK
    ELSE
      AK = EK
      DK = BK-AK
      F0 = F1
    END IF
    ISTEP = ISTEP+1
  END DO
  WRITE (6,"(I4,3F16.8)") ISTEP,EK,DK,F1
END PROGRAM WAVE
!
SUBROUTINE NMRV (N,H,Q,S,U)
!
! The Numerov algorithm for the equation u"(x)+q(x)u(x)=s(x)
! as given in Eqs. (3.77)-(3.80) in the book.
!
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL, INTENT (IN) :: H
  REAL :: G,C0,C1,C2,D,UTMP
  REAL, INTENT (IN), DIMENSION (N) :: Q,S
  REAL, INTENT (INOUT), DIMENSION (N) :: U
!
  G = H*H/12.0
!
  DO I = 2, N-1
    C0 = 1.0+G*((Q(I-1)-Q(I+1))/2.0+Q(I))
    C1 = 2.0-G*(Q(I+1)+Q(I-1)+8.0*Q(I))
```

```
    C2 = 1.0+G*((Q(I+1)-Q(I-1))/2.0+Q(I))
    D  = G*(S(I+1)+S(I-1)+10.0*S(I))
    UTMP   = C1*U(I)-C0*U(I-1)+D
    U(I+1) = UTMP/C2
  END DO
END SUBROUTINE NMRV
```

برنامه 27:

```
SUBROUTINE NMRV2 (N,H,Q,S,U)
!
! The Numerov algorithm for the equation u"(x)+q(x)u(x)=s(x)
! as given in Eqs. (3.82)-(3.85) in the book.
!
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL,INTENT (IN) :: H
  REAL :: G,C0,C1,C2,D,UTMP
  REAL, INTENT (IN), DIMENSION (N) :: Q,S
  REAL, INTENT (INOUT), DIMENSION (N) :: U
!
  G = H*H/12.0
!
  DO I = 2, N-1
    C0 = 1.0+G*Q(I-1)
    C1 = 2.0-10.0*G*Q(I)
    C2 = 1.0+G*Q(I+1)
    D  = G*(S(I+1)+S(I-1)+10.0*S(I))
    UTMP   = C1*U(I)-C0*U(I-1)+D
    U(I+1) = UTMP/C2
  END DO
END SUBROUTINE NMRV2
```

برنامه 28:

```
MODULE CSEED
  INTEGER :: ISEED
END MODULE CSEED
!
SUBROUTINE PERCOLATION (L,N,M,P)
!
! Subroutine to create an N*M percolation network.
!
!
  INTEGER, INTENT (IN) :: N,M
  REAL, INTENT (IN) :: P
  REAL:: R,RANF
  LOGICAL, INTENT (OUT), DIMENSION (N,M) :: L
!
  DO I = 1, N
    DO J = 1, M
      R = RANF()
      IF(R.LT.P) THEN
        L(I,J) = .TRUE.
      ELSE
        L(I,J) = .FALSE.
      END IF
    END DO
  END DO
```

```
END SUBROUTINE PERCOLATION
!
FUNCTION RANF() RESULT (R)
!
! Uniform random number generator x(n+1) = a*x(n) mod c with
! a=7**5 and c = 2**(31)-1.
!
  USE CSEED
  IMPLICIT NONE
  INTEGER :: IH,IL,IT,IA,IC,IQ,IR
  DATA IA/16807/,IC/2147483647/,IQ/127773/,IR/2836/
  REAL :: R
!
  IH = ISEED/IQ
  IL = MOD(ISEED,IQ)
  IT = IA*IL-IR*IH
  IF(IT.GT.0) THEN
    ISEED = IT
  ELSE
    ISEED = IC+IT
  END IF
  R = ISEED/FLOAT(IC)
END FUNCTION RANF
```

برنامه 29:

```
MODULE CSEED
  INTEGER :: ISEED
END MODULE CSEED
!
SUBROUTINE GRNF (X,Y)
!
! Two Gaussian random numbers generated from two uniform random
! numbers.
!
  IMPLICIT NONE
  REAL, INTENT (OUT) :: X,Y
  REAL :: PI,R1,R2,R,RANF
!
  PI = 4.0*ATAN(1.0)
  R1 = -ALOG(1.0-RANF())
  R2 = 2.0*PI*RANF()
  R1 = SQRT(2.0*R1)
  X  = R1*COS(R2)
  Y  = R1*SIN(R2)
END SUBROUTINE GRNF
!
FUNCTION RANF() RESULT (R)
!
! Uniform random number generator x(n+1) = a*x(n) mod c with
! a=7**5 and c = 2**(31)-1.
!
  USE CSEED
  IMPLICIT NONE
  INTEGER :: IH,IL,IT,IA,IC,IQ,IR
  DATA IA/16807/,IC/2147483647/,IQ/127773/,IR/2836/
  REAL :: R
!
  IH = ISEED/IQ
  IL = MOD(ISEED,IQ)
  IT = IA*IL-IR*IH
```

```
  IF(IT.GT.0) THEN
    ISEED = IT
  ELSE
    ISEED = IC+IT
  END IF
  R = ISEED/FLOAT(IC)
END FUNCTION RANF
```

برنامه 30:

```
MODULE CSEED
  INTEGER :: ISEED
END MODULE CSEED
!
FUNCTION ERNF() RESULT (E)
!
! Exponential random number generator from a uniform random
! number generator.
!
  REAL E,R,RANF
!
  E = -ALOG(1.0-RANF())
END FUNCTION ERNF
!
FUNCTION RANF() RESULT (R)
!
! Uniform random number generator x(n+1) = a*x(n) mod c with
! a=7**5 and c = 2**(31)-1.
!
  USE CSEED
  IMPLICIT NONE
  INTEGER :: IH,IL,IT,IA,IC,IQ,IR
  DATA IA/16807/,IC/2147483647/,IQ/127773/,IR/2836/
  REAL :: R
!
  IH = ISEED/IQ
  IL = MOD(ISEED,IQ)
  IT = IA*IL-IR*IH
  IF(IT.GT.0) THEN
    ISEED = IT
  ELSE
    ISEED = IC+IT
  END IF
  R = ISEED/FLOAT(IC)
END FUNCTION RANF
```

برنامه 31:

```
MODULE CSEED
  INTEGER :: ISEED
END MODULE CSEED
!
FUNCTION RANF() RESULT (R)
!
! Uniform random number generator x(n+1) = a*x(n) mod c with
! a=7**5 and c = 2**(31)-1.
!
  USE CSEED
  IMPLICIT NONE
  INTEGER :: IH,IL,IT,IA,IC,IQ,IR
  DATA IA/16807/,IC/2147483647/,IQ/127773/,IR/2836/
```

```
  REAL :: R
!
  IH = ISEED/IQ
  IL = MOD(ISEED,IQ)
  IT = IA*IL-IR*IH
  IF(IT.GT.0) THEN
    ISEED = IT
  ELSE
    ISEED = IC+IT
  END IF
  R = ISEED/FLOAT(IC)
END FUNCTION RANF
```

برنامه 32:

```
MODULE CB
  REAL :: B,E,A
END MODULE CB
!
PROGRAM SCATTERING
!
! This is the main program for the scattering problem.
!
!
  USE CB
  IMPLICIT NONE
  INTEGER, PARAMETER :: M=21,N=10001
  INTEGER I,J,ISTEP
  REAL :: DL,B0,DB,DX,X0,X,DX0,F,FX,FB,FBX,G1,G2,RU,RUTH,SI
  REAL, DIMENSION (N) :: FI
  REAL, DIMENSION (M) :: THETA,SIG,SIG1
!
  DL = 1.E-06
  B0 = 0.01
  DB = 0.5
  DX = 0.01
  E  = 1.0
  A  = 100.0
  DO I = 1, M
    B  = B0+(I-1)*DB
!
! Calculate the first term of theta
!
    DO J = 1, N
      X = B+DX*J
      FI(J) = 1.0/(X*X*SQRT(FBX(X)))
    END DO
    CALL SIMP(N,DX,FI,G1)
!
! Find r_m from 1-b*b/(r*r)-U/E=0
!
    X0  = B
    DX0 = DX
    CALL SECANT (DL,X0,DX0,ISTEP)
!
! Calculate the second term of theta
!
    DO J = 1, N
      X = X0+DX*J
      FI(J) = 1.0/(X*X*SQRT(FX(X)))
    END DO
```

```
    CALL SIMP (N,DX,FI,G2)
    THETA(I) = 2.0*B*(G1-G2)
    END DO
!
! Calculate d_theta/d_b
!
    CALL THREE (M,DB,THETA,SIG,SIG1)
!
! Put the cross section in log form with the exact result of
! the Coulomb scattering (RUTH)
!
    DO I = M, 1, -1
      B      = B0+(I-1)*DB
      SIG(I) = B/ABS(SIG(I))/SIN(THETA(I))
      RUTH   = 1.0/SIN(THETA(I)/2.0)**4/16.0
      SI     = ALOG(SIG(I))
      RU     = ALOG(RUTH)
      WRITE (6,"(3F16.8)") THETA(I),SI,RU
    END DO
END PROGRAM SCATTERING
!
SUBROUTINE SIMP (N,H,FI,S)
!
! Subroutine for integration over f(x) with the Simpson rule.  FI:
! integrand f(x); H: interval; S: integral.
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL, INTENT (IN) :: H
  REAL :: S0,S1,S2
  REAL, INTENT (OUT) :: S
  REAL, INTENT (IN), DIMENSION (N) :: FI
!
  S  = 0.0
  S0 = 0.0
  S1 = 0.0
  S2 = 0.0
  DO I = 2, N-1, 2
    S1 = S1+FI(I-1)
    S0 = S0+FI(I)
    S2 = S2+FI(I+1)
  END DO
  S = H*(S1+4.0*S0+S2)/3.0
!
! If N is even, add the last slice separately
!
  IF (MOD(N,2).EQ.0) S = S &
     +H*(5.0*FI(N)+8.0*FI(N-1)-FI(N-2))/12.0
END SUBROUTINE SIMP
!
SUBROUTINE SECANT (DL,X0,DX,ISTEP)
!
! Subroutine for the root of f(x)=0 with the secant method.
!
!
  IMPLICIT NONE
  INTEGER, INTENT (INOUT) :: ISTEP
  REAL, INTENT (INOUT) :: X0,DX
  REAL :: X1,X2,D,F,FX
  REAL, INTENT (IN) :: DL
```

```
!
  ISTEP = 0
  X1 = X0+DX
  DO WHILE (ABS(DX).GT.DL)
    D  = FX(X1)-FX(X0)
    X2 = X1-FX(X1)*(X1-X0)/D
    X0 = X1
    X1 = X2
    DX = X1-X0
    ISTEP = ISTEP+1
  END DO
END SUBROUTINE SECANT
!
SUBROUTINE THREE (N,H,FI,F1,F2)
!
! Subroutine for 1st and 2nd order derivatives with the three-point
! formulas. Extrapolations are made at the boundaries.  FI: input
! f(x); H: interval; F1: f'; and F2: f".
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL, INTENT (IN) :: H
  REAL, INTENT (IN), DIMENSION (N) :: FI
  REAL, INTENT (OUT), DIMENSION (N) :: F1,F2
!
! f' and f" from three-point formulas
!
  DO I = 2, N-1
    F1(I) = (FI(I+1)-FI(I-1))/(2.*H)
    F2(I) = (FI(I+1)-2.0*FI(I)+FI(I-1))/(H*H)
  END DO
!
! Linear extrapolation for the boundary points
!
  F1(1) = 2.0*F1(2)-F1(3)
  F1(N) = 2.0*F1(N-1)-F1(N-2)
  F2(1) = 2.0*F2(2)-F2(3)
  F2(N) = 2.0*F2(N-1)-F2(N-2)
END SUBROUTINE THREE
!
FUNCTION FX(X) RESULT (F)
  USE CB
  IMPLICIT NONE
  REAL :: X,F,U,UX
!
  F = 1.0-B*B/(X*X)-UX(X)/E
END FUNCTION FX
!
FUNCTION FBX(X) RESULT (FB)
  USE CB
  IMPLICIT NONE
  REAL :: X,FB
!
    FB = 1.0-B*B/(X*X)
END FUNCTION FBX
!
FUNCTION UX(X) RESULT (U)
  USE CB
  IMPLICIT NONE
  REAL :: X,U
```

```
!
  U = 1.0/X*EXP(-X/A)
END FUNCTION UX
```

برنامه 33:

```
PROGRAM EULER_CONST
  INCLUDE 'mpif.h'
  INTEGER :: N,K,IERR,IRANK,IPROC,IFINISH
  REAL*8, PARAMETER :: SUM25=0.5772156649015328606065 12D0
  REAL*8 :: SUMI,SUM
!
  CALL MPI_INIT (IERR)
  CALL MPI_COMM_RANK (MPI_COMM_WORLD,IRANK,IERR)
  CALL MPI_COMM_SIZE (MPI_COMM_WORLD,IPROC,IERR)
!
  IF (IRANK.EQ.0) THEN
    PRINT*, 'Enter total number of terms in the series: '
    READ (5,*) N
  END IF
!
! Broadcast the total number of terms to every process
!
  CALL MPI_BCAST (N,1,MPI_INTEGER,0,MPI_COMM_WORLD,IERR)
  K = (N/IPROC)
  SUMI = 0.D0
!
  IF (IRANK.NE.(IPROC-1)) then
    DO I = IRANK*K+1, (IRANK+1)*K
      SUMI = SUMI+1.D0/DFLOAT(I)
    END DO
  ELSE
    DO I = IRANK*K+1, N
      SUMI = SUMI + 1.D0/DFLOAT(I)
    END DO
  END IF
!
! Collect the sums from every process
!
  CALL MPI_REDUCE (SUMI,SUM,1,MPI_DOUBLE_PRECISION, &
                   MPI_SUM,0,MPI_COMM_WORLD,IERR)
  IF (IRANK.EQ.0) THEN
    SUM = SUM-DLOG(DFLOAT(N))
    PRINT*, 'The evaluated Euler constant is ', SUM, &
            'with the estimated error of ', DABS(SUM-SUM25)
  END IF
!
  CALL MPI_FINALIZE (IFINISH)
END PROGRAM EULER_CONST
```

برنامه 34:

```
PROGRAM TALK_0_TO_1
  INCLUDE 'mpif.h'
  INTEGER :: IRANK,IPROC,ITAG,ISEND,IRECV,IERR,IM,ID,IFINISH
  INTEGER, DIMENSION (MPI_STATUS_SIZE) :: ISTAT
  CHARACTER*40 :: HELLO
!
  ITAG  = 730
  ID    = 40
  ISEND = 0
```

```
  IRECV = 1
  CALL MPI_INIT (IERR)
  CALL MPI_COMM_RANK (MPI_COMM_WORLD,IRANK,IERR)
  CALL MPI_COMM_SIZE (MPI_COMM_WORLD,IPROC,IERR)
  PRINT*, IRANK, IPROC
  CALL MPI_BARRIER (MPI_COMM_WORLD,IERR)
  IF (IRANK.EQ.ISEND) THEN
    HELLO = 'I am process 0, who are you ?'
    IM = 29
    CALL MPI_SEND (HELLO,IM,MPI_CHARACTER,IRECV, &
                   ITAG,MPI_COMM_WORLD,IERR)
    PRINT*, 'I sent the message: ', HELLO
  ELSE IF (IRANK.EQ.IRECV) THEN
    CALL MPI_RECV (HELLO,ID,MPI_CHARACTER,ISEND, &
                    ITAG,MPI_COMM_WORLD,ISTAT,IERR)
    PRINT*, 'I got your message which is: ', HELLO
  END IF
  CALL MPI_FINALIZE(IFINISH)
END PROGRAM TALK_0_TO_1
```

برنامه 35:

```
MODULE CSEED
  INTEGER :: ISEED
END MODULE CSEED
!
SUBROUTINE PERCOLATION (L,N,M,P)
!
! Subroutine to create an N*M percolation network.
!
!
  INTEGER, INTENT (IN) :: N,M
  REAL, INTENT (IN) :: P
  REAL:: R,RANF
  LOGICAL, INTENT (OUT), DIMENSION (N,M) :: L
!
  DO I = 1, N
    DO J = 1, M
      R = RANF()
      IF(R.LT.P) THEN
        L(I,J) = .TRUE.
      ELSE
        L(I,J) = .FALSE.
      END IF
    END DO
  END DO
END SUBROUTINE PERCOLATION
!
FUNCTION RANF() RESULT (R)
!
! Uniform random number generator x(n+1) = a*x(n) mod c with
! a=7**5 and c = 2**(31)-1.
!
  USE CSEED
  IMPLICIT NONE
  INTEGER :: IH,IL,IT,IA,IC,IQ,IR
  DATA IA/16807/,IC/2147483647/,IQ/127773/,IR/2836/
  REAL :: R
!
  IH = ISEED/IQ
  IL = MOD(ISEED,IQ)
```

```
  IT = IA*IL-IR*IH
  IF(IT.GT.0) THEN
    ISEED = IT
  ELSE
    ISEED = IC+IT
  END IF
  R = ISEED/FLOAT(IC)
END FUNCTION RANF
```

برنامه 36:

```
MODULE CSEED
  INTEGER :: ISEED
END MODULE CSEED
!
SUBROUTINE GRNF (X,Y)
!
! Two Gaussian random numbers generated from two uniform random
! numbers.
!
  IMPLICIT NONE
  REAL, INTENT (OUT) :: X,Y
  REAL :: PI,R1,R2,R,RANF
!
  PI = 4.0*ATAN(1.0)
  R1 = -ALOG(1.0-RANF())
  R2 = 2.0*PI*RANF()
  R1 = SQRT(2.0*R1)
  X  = R1*COS(R2)
  Y  = R1*SIN(R2)
END SUBROUTINE GRNF
!
FUNCTION RANF() RESULT (R)
!
! Uniform random number generator x(n+1) = a*x(n) mod c with
! a=7**5 and c = 2**(31)-1.
!
  USE CSEED
  IMPLICIT NONE
  INTEGER :: IH,IL,IT,IA,IC,IQ,IR
  DATA IA/16807/,IC/2147483647/,IQ/127773/,IR/2836/
  REAL :: R
!
  IH = ISEED/IQ
  IL = MOD(ISEED,IQ)
  IT = IA*IL-IR*IH
  IF(IT.GT.0) THEN
    ISEED = IT
  ELSE
    ISEED = IC+IT
  END IF
  R = ISEED/FLOAT(IC)
END FUNCTION RANF
```

برنامه 37:

```
MODULE CSEED
  INTEGER :: ISEED
END MODULE CSEED
!
FUNCTION ERNF() RESULT (E)
```

```
!
! Exponential random number generator from a uniform random
! number generator.
!
  REAL E,R,RANF
!
  E = -ALOG(1.0-RANF())
END FUNCTION ERNF
!
FUNCTION RANF() RESULT (R)
!
! Uniform random number generator x(n+1) = a*x(n) mod c with
! a=7**5 and c = 2**(31)-1.
!
  USE CSEED
  IMPLICIT NONE
  INTEGER :: IH,IL,IT,IA,IC,IQ,IR
  DATA IA/16807/,IC/2147483647/,IQ/127773/,IR/2836/
  REAL :: R
!
  IH = ISEED/IQ
  IL = MOD(ISEED,IQ)
  IT = IA*IL-IR*IH
  IF(IT.GT.0) THEN
    ISEED = IT
  ELSE
    ISEED = IC+IT
  END IF
  R = ISEED/FLOAT(IC)
END FUNCTION RANF
```

برنامه 38:

```
MODULE CSEED
  INTEGER :: ISEED
END MODULE CSEED
!
FUNCTION RANF() RESULT (R)
!
! Uniform random number generator x(n+1) = a*x(n) mod c with
! a=7**5 and c = 2**(31)-1.
!
  USE CSEED
  IMPLICIT NONE
  INTEGER :: IH,IL,IT,IA,IC,IQ,IR
  DATA IA/16807/,IC/2147483647/,IQ/127773/,IR/2836/
  REAL :: R
!
  IH = ISEED/IQ
  IL = MOD(ISEED,IQ)
  IT = IA*IL-IR*IH
  IF(IT.GT.0) THEN
    ISEED = IT
  ELSE
    ISEED = IC+IT
  END IF
  R = ISEED/FLOAT(IC)
END FUNCTION RANF
```

برنامه 39:

```
MODULE CB
  REAL :: B,E,A
END MODULE CB
!
PROGRAM SCATTERING
!
! This is the main program for the scattering problem.
!
!
  USE CB
  IMPLICIT NONE
  INTEGER, PARAMETER :: M=21,N=10001
  INTEGER I,J,ISTEP
  REAL :: DL,B0,DB,DX,X0,X,DX0,F,FX,FB,FBX,G1,G2,RU,RUTH,SI
  REAL, DIMENSION (N) :: FI
  REAL, DIMENSION (M) :: THETA,SIG,SIG1
!
  DL = 1.E-06
  B0 = 0.01
  DB = 0.5
  DX = 0.01
  E  = 1.0
  A  = 100.0
  DO I = 1, M
    B  = B0+(I-1)*DB
!
! Calculate the first term of theta
!
    DO J = 1, N
      X = B+DX*J
      FI(J) = 1.0/(X*X*SQRT(FBX(X)))
    END DO
    CALL SIMP(N,DX,FI,G1)
!
! Find r_m from 1-b*b/(r*r)-U/E=0
!
    X0  = B
    DX0 = DX
    CALL SECANT (DL,X0,DX0,ISTEP)
!
! Calculate the second term of theta
!
    DO J = 1, N
      X = X0+DX*J
      FI(J) = 1.0/(X*X*SQRT(FX(X)))
    END DO
    CALL SIMP (N,DX,FI,G2)
    THETA(I) = 2.0*B*(G1-G2)
    END DO
!
! Calculate d_theta/d_b
!
    CALL THREE (M,DB,THETA,SIG,SIG1)
!
! Put the cross section in log form with the exact result of
! the Coulomb scattering (RUTH)
!
    DO I = M, 1, -1
      B      = B0+(I-1)*DB
      SIG(I) = B/ABS(SIG(I))/SIN(THETA(I))
      RUTH   = 1.0/SIN(THETA(I)/2.0)**4/16.0
```

```
      SI     = ALOG(SIG(I))
      RU     = ALOG(RUTH)
      WRITE (6,"(3F16.8)") THETA(I),SI,RU
    END DO
END PROGRAM SCATTERING
!
SUBROUTINE SIMP (N,H,FI,S)
!
! Subroutine for integration over f(x) with the Simpson rule.  FI:
! integrand f(x); H: interval; S: integral.
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL, INTENT (IN) :: H
  REAL :: S0,S1,S2
  REAL, INTENT (OUT) :: S
  REAL, INTENT (IN), DIMENSION (N) :: FI
!
  S  = 0.0
  S0 = 0.0
  S1 = 0.0
  S2 = 0.0
  DO I = 2, N-1, 2
    S1 = S1+FI(I-1)
    S0 = S0+FI(I)
    S2 = S2+FI(I+1)
  END DO
  S = H*(S1+4.0*S0+S2)/3.0
!
! If N is even, add the last slice separately
!
  IF (MOD(N,2).EQ.0) S = S &
     +H*(5.0*FI(N)+8.0*FI(N-1)-FI(N-2))/12.0
END SUBROUTINE SIMP
!
SUBROUTINE SECANT (DL,X0,DX,ISTEP)
!
! Subroutine for the root of f(x)=0 with the secant method.
!
!
  IMPLICIT NONE
  INTEGER, INTENT (INOUT) :: ISTEP
  REAL, INTENT (INOUT) :: X0,DX
  REAL :: X1,X2,D,F,FX
  REAL, INTENT (IN) :: DL
!
  ISTEP = 0
  X1 = X0+DX
  DO WHILE (ABS(DX).GT.DL)
    D  = FX(X1)-FX(X0)
    X2 = X1-FX(X1)*(X1-X0)/D
    X0 = X1
    X1 = X2
    DX = X1-X0
    ISTEP = ISTEP+1
  END DO
END SUBROUTINE SECANT
!
SUBROUTINE THREE (N,H,FI,F1,F2)
!
```

```
! Subroutine for 1st and 2nd order derivatives with the three-point
! formulas. Extrapolations are made at the boundaries.  FI: input
! f(x); H: interval; F1: f'; and F2: f".
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL, INTENT (IN) :: H
  REAL, INTENT (IN), DIMENSION (N) :: FI
  REAL, INTENT (OUT), DIMENSION (N) :: F1,F2
!
! f' and f" from three-point formulas
!
  DO I = 2, N-1
    F1(I) = (FI(I+1)-FI(I-1))/(2.*H)
    F2(I) = (FI(I+1)-2.0*FI(I)+FI(I-1))/(H*H)
  END DO
!
! Linear extrapolation for the boundary points
!
  F1(1) = 2.0*F1(2)-F1(3)
  F1(N) = 2.0*F1(N-1)-F1(N-2)
  F2(1) = 2.0*F2(2)-F2(3)
  F2(N) = 2.0*F2(N-1)-F2(N-2)
END SUBROUTINE THREE
!
FUNCTION FX(X) RESULT (F)
  USE CB
  IMPLICIT NONE
  REAL :: X,F,U,UX
!
  F = 1.0-B*B/(X*X)-UX(X)/E
END FUNCTION FX
!
FUNCTION FBX(X) RESULT (FB)
  USE CB
  IMPLICIT NONE
  REAL :: X,FB
!
    FB = 1.0-B*B/(X*X)
END FUNCTION FBX
!
FUNCTION UX(X) RESULT (U)
  USE CB
  IMPLICIT NONE
  REAL :: X,U
!
  U = 1.0/X*EXP(-X/A)
END FUNCTION UX
```

برنامه 40:

```
PROGRAM EULER_CONST
  INCLUDE 'mpif.h'
  INTEGER :: N,K,IERR,IRANK,IPROC,IFINISH
  REAL*8, PARAMETER :: SUM25=0.5772156649015328606065 12D0
  REAL*8 :: SUMI,SUM
!
  CALL MPI_INIT (IERR)
  CALL MPI_COMM_RANK (MPI_COMM_WORLD,IRANK,IERR)
  CALL MPI_COMM_SIZE (MPI_COMM_WORLD,IPROC,IERR)
```

```
!
  IF (IRANK.EQ.0) THEN
    PRINT*, 'Enter total number of terms in the series: '
    READ (5,*) N
  END IF
!
! Broadcast the total number of terms to every process
!
  CALL MPI_BCAST (N,1,MPI_INTEGER,0,MPI_COMM_WORLD,IERR)
  K = (N/IPROC)
  SUMI = 0.D0
!
  IF (IRANK.NE.(IPROC-1)) then
    DO I = IRANK*K+1, (IRANK+1)*K
      SUMI = SUMI+1.D0/DFLOAT(I)
    END DO
  ELSE
    DO I = IRANK*K+1, N
      SUMI = SUMI + 1.D0/DFLOAT(I)
    END DO
  END IF
!
! Collect the sums from every process
!
  CALL MPI_REDUCE (SUMI,SUM,1,MPI_DOUBLE_PRECISION, &
                   MPI_SUM,0,MPI_COMM_WORLD,IERR)
  IF (IRANK.EQ.0) THEN
    SUM = SUM-DLOG(DFLOAT(N))
    PRINT*, 'The evaluated Euler constant is ', SUM, &
            'with the estimated error of ', DABS(SUM-SUM25)
  END IF
!
  CALL MPI_FINALIZE (IFINISH)
END PROGRAM EULER_CONST
```

برنامه 41:

```
PROGRAM TALK_0_TO_1
  INCLUDE 'mpif.h'
  INTEGER :: IRANK,IPROC,ITAG,ISEND,IRECV,IERR,IM,ID,IFINISH
  INTEGER, DIMENSION (MPI_STATUS_SIZE) :: ISTAT
  CHARACTER*40 :: HELLO
!
  ITAG  = 730
  ID    = 40
  ISEND = 0
  IRECV = 1
  CALL MPI_INIT (IERR)
  CALL MPI_COMM_RANK (MPI_COMM_WORLD,IRANK,IERR)
  CALL MPI_COMM_SIZE (MPI_COMM_WORLD,IPROC,IERR)
  PRINT*, IRANK, IPROC
  CALL MPI_BARRIER (MPI_COMM_WORLD,IERR)
  IF (IRANK.EQ.ISEND) THEN
    HELLO = 'I am process 0, who are you ?'
    IM = 29
    CALL MPI_SEND (HELLO,IM,MPI_CHARACTER,IRECV, &
                   ITAG,MPI_COMM_WORLD,IERR)
    PRINT*, 'I sent the message: ', HELLO
  ELSE IF (IRANK.EQ.IRECV) THEN
    CALL MPI_RECV (HELLO,ID,MPI_CHARACTER,ISEND, &
                   ITAG,MPI_COMM_WORLD,ISTAT,IERR)
```

```
    PRINT*, 'I got your message which is: ', HELLO
  END IF
  CALL MPI_FINALIZE(IFINISH)
END PROGRAM TALK_0_TO_1
```

برنامه 42:

```
MODULE ORDER_AN_ARRAY
  PRIVATE EXCHANGE
!
  CONTAINS
!
  SUBROUTINE REARRANGE (A)
    IMPLICIT NONE
    REAL, INTENT(INOUT) :: A(:)
    LOGICAL, ALLOCATABLE :: MASK(:)
    INTEGER :: I, N
    INTEGER, DIMENSION(1) :: K
    N = SIZE (A)
    ALLOCATE (MASK(N))
    MASK = .TRUE.
    DO I = 0, N-1
      MASK(N-I) = .FALSE.
       K  = MAXLOC(A,MASK)
      CALL EXCHANGE(A(K(1)),A(N-I))
    END DO
  END SUBROUTINE REARRANGE
!
  SUBROUTINE EXCHANGE (X,Y)
    IMPLICIT NONE
    REAL, INTENT(INOUT):: X,Y
    REAL TX
    TX = X; X = Y; Y = TX
  END SUBROUTINE EXCHANGE
!
END MODULE ORDER_AN_ARRAY
!
PROGRAM RANDOM_ARRAY_ORDERED
  USE ORDER_AN_ARRAY
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N = 100
  REAL, DIMENSION(N) :: A
  INTEGER :: I
!
  CALL RANDOM_NUMBER (A)
  CALL REARRANGE (A)
  WRITE(6, "(F10.8)") (A(I),I=1,N)
END PROGRAM RANDOM_ARRAY_ORDERED
```

برنامه 43:

```
PROGRAM ARRAY_EXAMPLE
  IMPLICIT NONE
  REAL :: TWO_PI
  REAL, ALLOCATABLE :: A(:,:), B(:,:), C(:,:), D(:)
  INTEGER :: N,M,L,I
!
  TWO_PI = 8.0*ATAN(1.0)
  READ "(3I4)", N, M, L
  ALLOCATE (A(N,M)); ALLOCATE (B(L,N))
```

```
  ALLOCATE (C(L,M)); ALLOCATE (D(M))
  CALL RANDOM_NUMBER (A); CALL RANDOM_NUMBER (B);
  A = SIN(TWO_PI*A); B = COS(TWO_PI*B)
  C = MATMUL(B,A)
  DO      I = 1, M
    D(I) = DOT_PRODUCT(A(:,I),B(I,:))
  END DO
  PRINT "(8F10.6)", D
END PROGRAM ARRAY_EXAMPLE
```

برنامه 44:

```
PROGRAM GALERKIN
!
! This program solves the one-dimensional Poisson equation with the
! Galerkin method as described in the text.
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=99
  INTEGER :: I
  REAL :: PI,XL,H,D,E,B0,B1,XIM,XI,XIP
  REAL, DIMENSION (N) :: B,A,Y,W,U
!
  PI  =  4.0*ATAN(1.0)
  XL  =  1.0
  H   =  XL/(N+1)
  D   =  2.0/H
  E   = -1.0/H
  B0  =  PI/H
  B1  =  1.0/H
!
! Find the elements in L and U
!
  W(1) =  D
  U(1) =  E/D
  DO I = 2, N
    W(I) = D-E*U(I-1)
    U(I) = E/W(I)
  END DO
!
! Assign the array B
!
  DO I = 1, N
    XIM  = H*(I-1)
    XI   = H*I
    XIP  = H*(I+1)
    B(I) = B0*COS(PI*XI)*(XIM+XIP-2.0*XI) &
           +B1*(2.0*SIN(PI*XI)-SIN(PI*XIM)-SIN(PI*XIP))
  END DO
!
! Find the solution
!
  Y(1) = B(1)/W(1)
  DO I = 2, N
    Y(I) = (B(I)-E*Y(I-1))/W(I)
  END DO
!
  A(N) = Y(N)
  DO I = N-1,1,-1
    A(I) = Y(I)-U(I)*A(I+1)
  END DO
```

```
!
  WRITE (6,"(2F16.8)") (I*H,A(I), I=1,N)
END PROGRAM GALERKIN
```

برنامه 45:

```
PROGRAM G_WATER
!
! This program solves the groundwater dynamics problem in the
! rectangular geometry through the relaxation method.
!
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: NX=101,NY=51,ISKX=10,ISKY=5,ITMX=5
  INTEGER :: I,J,ISTP
  REAL :: PI,A0,B0,H0,CH,SX,SY,HX,HY,P,X,Y
  REAL, DIMENSION (NX,NY) :: PHI,CK,SN
!
  PI = 4.0*ATAN(1.0)
  A0 = 1.0
  B0 = -0.04
  H0 = 200.0
  CH = -20.0
  SX = 1000.0
  SY = 500.0
  HX = SX/(NX-1)
  HY = SY/(NY-1)
  P  = 0.5
!
! Set up boundary conditions and initial guess of the solution
!
  DO I = 1, NX
    X = (I-1)*HX
    DO J = 1, NY
      Y = (J-1)*HY
      SN(I,J) = 0.0
      CK(I,J) = A0+B0*Y
      PHI(I,J) = H0+CH*COS(PI*X/SX)*Y/SY
    END DO
  END DO
!
  DO ISTP = 1, ITMX
!
! Ensure the boundary conditions by the 4-point formula
!
    DO J = 1, NY
      PHI(1,J)  = (4.0*PHI(2,J)-PHI(3,J))/3.0
      PHI(NX,J) = (4.0*PHI(NX-1,J)-PHI(NX-2,J))/3.0
    END DO
!
    CALL RX2D (PHI,CK,SN,NX,NY,P,HX,HY)
  END DO
!
  DO I = 1, NX, ISKX
    X = (I-1)*HX
    DO J = 1, NY, ISKY
      Y = (J-1)*HY
      WRITE (6,"(3F16.8)") X,Y,PHI(I,J)
    END DO
  END DO
END PROGRAM G_WATER
```

```
!
SUBROUTINE RX2D (FN,DN,S,NX,NY,P,HX,HY)
!
! Subroutine performing one iteration of the relaxation for
! the two-dimensional Poisson equation.
!
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: NX,NY
  INTEGER :: I,J
  REAL, INTENT (IN) :: HX,HY,P
  REAL :: HX2,A,B,Q,CIP,CIM,CJP,CJM
  REAL, INTENT (IN), DIMENSION (NX,NY) :: DN,S
  REAL, INTENT (INOUT), DIMENSION (NX,NY) :: FN
!
  HX2 = HX*HX
  A = HX2/(HY*HY)
  B = 1.0/(4.0*(1.0+A))
  Q = 1.0-P
!
  DO I = 2, NX-1
    DO J = 2, NY-1
      CIP = B*(DN(I+1,J)/DN(I,J)+1.0)
      CIM = B*(DN(I-1,J)/DN(I,J)+1.0)
      CJP = A*B*(DN(I,J+1)/DN(I,J)+1.0)
      CJM = A*B*(DN(I,J-1)/DN(I,J)+1.0)
      FN(I,J) = Q*FN(I,J)+P*(CIP*FN(I+1,J)+CIM*FN(I-1,J) &
               +CJP*FN(I,J+1)+CJM*FN(I,J-1)+HX2*S(I,J))
    END DO
  END DO
END SUBROUTINE RX2D
```

برنامه 46:

```
SUBROUTINE BSSL (BJ,BY,N,X)
!
! Subroutine to generate J_n(x) and Y_n(x) with given x and
! up to N=NMAX-NTEL.
!
  INTEGER, PARAMETER :: NMAX=30,NTEL=5
  INTEGER, INTENT (IN) :: N
  INTEGER :: I,J,K
  REAL, INTENT (IN) :: X
  REAL :: PI,GAMMA,SUM,SUM1
  REAL, INTENT (OUT), DIMENSION (0:N) :: BJ,BY
  REAL, DIMENSION (0:NMAX) :: B1
!
  PI = 4.0*ATAN(1.0)
  GAMMA = 0.5772156649
!
  B1(NMAX)   = 0.0
  B1(NMAX-1) = 1.0
!
! Generating J_n(x)
!
  SUM = 0.0
  DO I = NMAX-1, 1, -1
    B1(I-1) = 2*I*B1(I)/X-B1(I+1)
    IF (MOD(I,2).EQ.0) SUM = SUM+2.0*B1(I)
  END DO
  SUM = SUM+B1(0)
!
```

```
  DO I = 0, N
    BJ(I) = B1(I)/SUM
  END DO
!
! Generating Y_n(x) starts here
!
  SUM1 = 0.0
  DO K = 1, NMAX/2
    SUM1 = SUM1+(-1)**K*B1(2*K)/K
  END DO
!
  SUM1 = -4.0*SUM1/(PI*SUM)
  BY(0) = 2.0*(ALOG(X/2.0)+GAMMA)*BJ(0)/PI+SUM1
  BY(1) = (BJ(1)*BY(0)-2.0/(PI*X))/BJ(0)
  DO I  = 1, N-1
    BY(I+1) = 2*I*BY(I)/X-BY(I-1)
  END DO
END SUBROUTINE BSSL
```

برنامه 47:

```
  SUBROUTINE FFT (AR,AI,N,M)
!
! An example of the fast Fourier transform subroutine with N = 2**M.
! AR and AI are the real and imaginary part of data in the input and
! corresponding Fourier coefficients in the output.
!
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N,M
  INTEGER :: N1,N2,I,J,K,L,L1,L2
  REAL :: PI,A1,A2,Q,U,V
  REAL, INTENT (INOUT), DIMENSION (N) :: AR,AI
!
  PI = 4.0*ATAN(1.0)
  N2 = N/2
!
  N1 = 2**M
  IF(N1.NE.N) STOP 'Indices do not match'
!
! Rearrange the data to the bit reversed order
!
  L = 1
  DO K = 1, N-1
    IF (K.LT.L) THEN
      A1    = AR(L)
      A2    = AI(L)
      AR(L) = AR(K)
      AR(K) = A1
      AI(L) = AI(K)
      AI(K) = A2
    END IF
    J   = N2
    DO WHILE (J.LT.L)
      L = L-J
      J = J/2
    END DO
    L = L+J
  END DO
!
! Perform additions at all levels with reordered data
```

```
!
  L2 = 1
  DO L = 1, M
    Q  =  0.0
    L1 =  L2
    L2 =  2*L1
    DO K = 1, L1
      U   =  COS(Q)
      V   = -SIN(Q)
      Q   =  Q + PI/L1
      DO J = K, N, L2
        I     =  J + L1
        A1    =  AR(I)*U-AI(I)*V
        A2    =  AR(I)*V+AI(I)*U
        AR(I) =  AR(J)-A1
        AR(J) =  AR(J)+A1
        AI(I) =  AI(J)-A2
        AI(J) =  AI(J)+A2
      END DO
    END DO
  END DO
END SUBROUTINE FFT
```

برنامه 48:

```
PROGRAM DFT_EXAMPLE
!
! Example of the discrete Fourier transform with function f(x) =
! x(1-x) in [0,1]. The inverse transform is also performed for
! comparison.
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=128,M=8
  INTEGER :: I
  REAL :: F0,H,X
  REAL, DIMENSION (N) :: FR,FI,GR,GI
!
  F0 = 1.0/SQRT(FLOAT(N))
  H  = 1.0/(N-1)
!
  DO I = 1, N
    X = H*(I-1)
    FR(I) = X*(1.0-X)
    FI(I) = 0.0
  END DO
!
  CALL DFT (FR,FI,GR,GI,N)
  DO I = 1, N
    GR(I) = F0*GR(I)
    GI(I) = F0*GI(I)
  END DO
!
! Perform inverse Fourier transform
!
  DO I = 1, N
    GI(I) = -GI(I)
  END DO
  CALL DFT (GR,GI,FR,FI,N)
  DO I = 1, N
    FR(I) =  F0*FR(I)
    FI(I) = -F0*FI(I)
```

```
  END DO
  WRITE (6,"(2F16.8)") (H*(I-1),FR(I),I=1,N,M)
  WRITE (6,"(2F16.8)") H*(N-1),FR(N)
END PROGRAM DFT_EXAMPLE
!
SUBROUTINE DFT (FR,FI,GR,GI,N)
!
! Subroutine to perform the discrete Fourier transform with
! FR and FI as the real and imaginary parts of the input and
! GR and GI as the corresponding  output.
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I,J
  REAL :: PI,X,Q
  REAL, INTENT (IN), DIMENSION (N) :: FR,FI
  REAL, INTENT (OUT), DIMENSION (N) :: GR,GI
!
  PI = 4.0*ATAN(1.0)
  X  = 2*PI/N
!
  DO I = 1, N
    GR(I) = 0.0
    GI(I) = 0.0
    DO J = 1, N
      Q = X*(J-1)*(I-1)
      GR(I) = GR(I)+FR(J)*COS(Q)+FI(J)*SIN(Q)
      GI(I) = GI(I)+FI(J)*COS(Q)-FR(J)*SIN(Q)
    END DO
  END DO
END SUBROUTINE DFT
```

برنامه 49:

```
PROGRAM S_L_LEGENDRE
!
! Main program for solving the Legendre equation with the simplest
! algorithm for the Sturm-Liouville equation and the bisection method
! for the root search.
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=501
  INTEGER :: ISTEP
  REAL :: DL,H,AK,BK,DK,EK,F0,F1
  REAL, DIMENSION (N) :: U
!
! Initialization of the problem
!
  DL = 1.0E-06
  H  = 2.0/(N-1)
  AK = 0.5
  BK = 1.5
  DK = 0.5
  EK = AK
  U(1)  = -1.0
  U(2)  = -1.0+H
  ISTEP =  0
  CALL SMPL (N,H,EK,U)
  F0 = U(N)-1.0
!
! Bisection method for the root
```

```
!
  DO WHILE (ABS(DK).GT.DL)
    EK = (AK+BK)/2.0
    CALL SMPL (N,H,EK,U)
    F1 = U(N)-1.0
    IF ((F0*F1).LT.0) THEN
      BK = EK
      DK = BK-AK
    ELSE
      AK = EK
      DK = BK-AK
      F0 = F1
    END IF
    ISTEP = ISTEP+1
  END DO
!
  WRITE (6,"(I4,3F16.8)") ISTEP,EK,DK,F1
END PROGRAM S_L_LEGENDRE
!
SUBROUTINE SMPL (N,H,EK,U)
!
! The simplest algorithm for the Sturm-Liouville equation.
!
!
  IMPLICIT NONE
  INTEGER, INTENT (IN) :: N
  INTEGER :: I
  REAL, INTENT (IN) :: H,EK
  REAL :: H2,Q,X,P,P1
  REAL, INTENT (OUT), DIMENSION (N) :: U
!
  H2 = 2.0*H*H
  Q = EK*(1.0+EK)
  DO I = 2, N-1
    X  =  (I-1)*H-1.0
    P  =  2.0*(1.0-X*X)
    P1 = -2.0*X*H
    U(I+1) = ((2.0*P-H2*Q)*U(I)+(P1-P)*U(I-1))/(P1+P)
  END DO
END SUBROUTINE SMPL
```

برنامه 50:

```
PROGRAM ONE_D_MOTION2
!
! Simplest predictor-corector algorithm applied to a particle in one
! dimension under an elastic force.
!
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=101,IN=5
  INTEGER :: I
  REAL :: PI,DT
  REAL, DIMENSION (N) :: T,V,X
!
  PI  = 4.0*ATAN(1.0)
  DT  =2.0*PI/100
  X(1)=0.0
  T(1)=0.0
  V(1)=1.0
!
```

```
  DO I = 1, N-1
    T(I+1) = I*DT
!
! Predictor for position and velocity
!
     X(I+1) = X(I)+V(I)*DT
     V(I+1) = V(I)-X(I)*DT
!
! Corrector for position and velocity
!
     X(I+1) = X(I)+(V(I)+V(I+1))*DT/2.0
     V(I+1) = V(I)-(X(I)+X(I+1))*DT/2.0
  END DO
  WRITE(6,"(3F16.8)") (T(I),X(I),V(I),I=1,N,IN)
END PROGRAM ONE_D_MOTION2
```