شروع برنامه نویسی
با استفاده از
پاسکال شئ گرا

کتاب لازاروس/پاسکال آزاد

نویسنده : معتز عبدالعظیم الطاهر
ویرایش: پت اندرسون–جیسون هاکنی
ترجمه و ویرایش : امیر شهریاری

۱۳۹۳–۲۰۱۴

مقدمه

این کتاب برای کسانی که قصد یادگیری برنامه نویسی به زبان پاسکال شئ گرا را دارند نوشته شده است. این کتاب همچنین به عنوان اولین کتاب برنامه نویسی برای دانشجویان جدید و غیر برنامه نویسان مناسب است. در این کتاب شرح تکنیک‌های برنامه نویسی به طور کلی علاوه بر پاسکال شئ گرا داده شده است.

زبان پاسکال شئ گرا

اولین پیدایش پشتیبانی برنامه نویسی شئ‌گرایی در زبان پاسکال در سال ۱۹۸۳ توسط کمپانی کامپیوتر اپل به وجود آمد. پس از آن بورلند پشتیبانی برنامه نویسی شئ گرا را در توربو پاسکال معروف خود قرار داد.

زبان پاسکال شئ‌گرا به طور کلی هدف دوگانه‌ای را دنبال می‌کند (برنامه‌نویسی ساخت‌یافته و شئ‌گرایی). این زبان می‌تواند برای طیف گسترده‌ای از برنامه‌ها مورد استفاده قرار بگیرد مانند آموزشی، توسعه بازی، برنامه‌های تجاری، برنامه‌های تحت وب، برنامه‌های ارتباطی، توسعه ابزارها و هسته سیستم عامل‌ها.

دلفی

پس از موفقیت توربو پاسکال، شرکت بورلند تصمیم گرفت که آن را در ویندوز ایجاد کند و تکنولوژی جزء محور را در آن معرفی کند. به زودی دلفی بهترین ابزار RAD (توسعه سریع برنامه) در زمان خودش شد.

اولین نسخه دلفی در سال ۱۹۹۵ با یک سری از اجزای و بسته‌های غنی که از ویندوز و برنامه‌های توسعه پایگاه داده پشتیبانی می‌کرد منتشر شد.

پاسکال رایگان

بعد از اینکه شرکت بورلند پشتیبانی از توربو پاسکال را قطع کرد، تیم پاسکال رایگان پروژه متن بازی را شروع کرد تا از ابتدا یک کامپایلر سازگار برای توربو پاسکال بنویسد، و سپس آن را با دلفی سازگار کنند. در این زمان کامپایلر پاسکال رایگان محیط‌ها و سیستم عامل‌های مختلفی را نظیر ویندوز، لینوکس، مکینتاش، ARM و Wince هدف قرار داده بود.

اولین ویرایش کامپایلر پاسکال رایگان در جولای سال ۲۰۰۰ منتشر شد.

لازاروس

پاسکال رایگان یک کامپایلر است، و فاقد یک محیط توسعه یکپارچه (IDE) چیزی شبیه به دلفی در ویندوز است. پروژه لازاروس جهت ارائه‌ی یک محیط توسعه یکپارچه برای پاسکال رایگان شروع شد. لازاروس یک ویرایشگر کد منبع و اشکال‌زدا را ارائه میداد و شامل تعداد زیادی چهارچوب، بسته و کتابخانه اجزا شبیه به محیط دلفی بود.

ویرایش ۱.۰ لازاروس در اگوست ۲۰۱۲ منتشر شد، اما برنامه‌های زیادی وجود دارند که با ویرایش بتا لازاروس توسعه داده شده‌اند. بسیاری از بسته‌ها و اجزا برای لازاروس به طور داوطلبانه نوشته شدند و جامعه آن در حال رشد است.

ویژگی‌های پاسکال شئ‌گرا

پاسکال شئ‌گرا زبانی است که برای شروع خیلی آسان و قابل خواندن است. کامپایلر آن خیلی سریع است، و برنامه‌های تولید شده‌ی آن قابل اطمینان و سریع بوده و می‌توانند با زبان C و C++ برابری کنند. شما می‌توانید برنامه‌های عظیم و بزرگ را با IDE آن (لازاروس و دلفی) بدون پیچیدگی بنویسید.

نویسنده: معتز عبدالعظیم

من از دانشگاه سودان در رشته علم و فناوری در سال ۱۹۹۹ فارغ التحصیل شده‌ام. من فراگیری پاسکال را به عنوان دومین زبان بعد از **BASIC** شروع کردم. از آن زمان به بعد، من به طور مداوم از آن استفاده کردم، و آن را ابزاری قوی و آسان درک کردم، به ویژه بعد از اینکه من **C** و **C++** را مورد مطالعه قرار دادم. سپس به دلفی مهاجرت کردم. از آن پس برای بیشتر برنامه‌هایم از دلفی و لازاروس استفاده کرده‌ام.

من در خارطوم زندگی می‌کنم. کار فعلی من توسعه برنامه است.

اولین ویرایشگر

پت اندرسون از کالج ایالتی واشنگتن غربی در سال ۱۹۶۸ و دانشکده حقوق راتگرز در سال ۱۹۷۵ فارغ التحصیل شده است. او به عنوان دادستان شهر سنوکوالمی واشگتون کار می کند، پت برنامه نویسی را بر روی یک خانه رادیویی **TRS-80** مدل ۳ در سال ۱۹۸۲ با یک مترجم داخلی **BASIC** شروع کرد، اما به زودی توربو پاسکال را پیدا کرد. همه نسخه های توربو پاسکال از ۴ الی ۷، و بسیاری از ویرایش‌های دلفی از ۱ الی ۴ متعلق به او بود. پت در برنامه نویسی خود وقفه‌ای در سال‌های ۱۹۹۸ الی ۲۰۰۹ داد، سپس او با پاسکال رایگان/لازاروس، که دوباره شور و اشتیاقش برای برنامه نویسی بود آمد.

دومین ویرایشگر

جیسون هاکنن از کالج حمل و نقل هوایی دانشگاه میشیگان غربی فارغ التحصیل شده است. او به عنوان یک خلبان حرفه‌ای تمام وقت برای یک شرکت برقی مستقر در میشیگان جنوبی کار می‌کند. جیسون یک برنامه نویس اتفاقی از اولین نمایشش در کومودور ۶۴ در حدود سال ۱۹۸۴ بوده است. به طور خلاصه توربو پاسکال در سال ۱۹۹۰ معرفی شد، علاقه پنهان برنامه نویسی او بتازگی بعد از یافتن لینوکس،لازاروس، و پاسکال رایگان شعله‌ور شده است.

مترجم فارسی

من امیر شهریاری فارغ التحصیل دانشگاه آزاد اسلامی واحد بیرجند در سال ۲۰۰۴ در مقطع کاردانی نرم‌افزار رایانه و جهاد دانشگاهی مشهد در سال ۲۰۱۱ در مقطع مهندسی نرم‌افزار می باشم. پس از زبان **BASIC** در سال ۲۰۰۱ با توبرو پاسکال آشنا شدم و دو سال بعد برنامه نویسی با دلفی ۵ را شروع کردم. در سال‌های ۲۰۰۴ تا ۲۰۰۹ برنامه نویسی با دلفی ۷ را ادامه دادم و کم کم با لینوکس آشنا شدم. در اوایل به دنبال محیطی همچون دلفی در لینوکس بودم که با کایلیکس آشنا شدم ولی خیلی زود آن را کنار گذاشتم. در سال ۲۰۰۹ با لازاروس آشنا شدم و سعی کردم در ویندوز و لینوکس به وسیله آن برنامه نویسی کنم.امروز من از شیفتگان لازاروس هستم و اغلب برنامه‌های خود را با پاسکال آزاد می نویسم.

مجوز

محیط مثال‌های کتاب

ما از لازاروس و پاسکال رایگان برای همه مثال‌های این کتاب استفاده خواهیم کرد. شما می‌توانید لازاروس به همراه کامپایلر پاسکال رایگان را از آدرس زیر دریافت کنید.

http://lazarus.freepascal.org

اگر شما از لینوکس استفاده می‌کنید می‌توانید لازاروس را از مخازن دریافت کنید، در توزیع‌های دبیان بیس از این دستور استفاده کنید :

sudo apt-get install lazarus

و در توزیع‌های بر پایه فدورا از این دستور :

yum install lazarus

لازاروس یک برنامه رایگان و متن باز است که برای تعداد زیادی ساختار فراهم است. برنامه‌هایی که در لازاروس نوشته می‌شوند می‌توانند دوباره برای محیط دیگری ترجمه شوند تا فایل اجرایی برای آن محیط را تولید کنند. برای مثال اگر شما یک برنامه با استفاده از لازاروس در ویندوز می نویسید، می‌توانید برنامه قابل اجرای آن را در لینوکس تولید کنید، کافیست فقط منبع کد آن را در لینوکس کپی کنید و آن را کامپایل نمایید. لازاروس برنامه‌ها را به صورتی که بومی هر سیستم عامل هستند تولید می‌کند، و نیازی به کتابخانه‌های اضافی یا ماشین‌های مجازی ندارد. به همین دلیل، به آسانی گسترش پیدا می‌کند و سریع اجرا می شود.

استفاده در حالت متنی

تمام مثال‌های فصل اول این کتاب برنامه‌های کاربردی تحت کنسول (برنامه‌های تحت متن/ برنامه‌های دستوری) خواهند بود، زیرا آنها به آسانی قابل درک بوده و استاندارد هستند. برنامه‌های رابط کاربر گرافیکی در فصل‌های بعد معرفی خواهند شد.

فهرست مطالب

فصل اول
مبانی زبان

اولین برنامه‌ی ما

بعد از نصب و اجرای لازاروس، می‌توانیم یک برنامه جدید را از منوی اصلی شروع کنیم.

Project/New Project/Program

این کد را در پنجره ویرایشگر منبع مشاهده خواهید کرد:

```
program Project1;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes
  { you can add units after this };

{$IFDEF WINDOWS}{$R project1.rc}{$ENDIF}

begin
end.
```

این برنامه را به وسیله کلیک روی **File/Save** از منوی اصلی می‌توانید ذخیره کنید و سپس نامی برای آن مشخص کنید، مثلاً **first.lpi** .

این خطوط را در میان عبارت‌های ***begin*** و ***end*** می‌توانید بنویسید:

```
  Writeln('This is Free Pascal and Lazarus');
  Writeln('Press enter key to close');
  Readln;
```

متن کامل برنامه به این صورت خواهد بود:

```
program first;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes
  { you can add units after this };

{$IFDEF WINDOWS}{$R first.rc}{$ENDIF}

begin
  Writeln('This is Free Pascal and Lazarus');
  Writeln('Press enter key to close');
  Readln;
end.
```

عبارت *Writeln* متن را بر روی صفحه (پنجره فرمان) نمایش می‌دهد. Readln اجرا را متوقف می‌کند تا اجازه دهد کاربر متن نمایش داده شده را تا زمانی که او کلید Enter را بزند تا برنامه بسته شود و به محیط لازاروس برگردید بخواند.

سپس کلید F۹ را برای اجرای برنامه فشار دهید یا بر روی کلید زیر کلیک کنید:

بعد از اجرای اولین برنامه، این متن خروجی را  دریافت خواهید کرد:

```
This is Free Pascal and Lazarus
Press enter key to close
```

اگر شما از لینوکس استفاده می کنید، یک فایل جدید در پوشه برنامه با نام (first) پیدا خواهید کرد، و در ویندوز این فایل با نام (first.exe) ایجاد خواهد شد. هر دوی این فایل‌ها می‌توانند مستقیماً با دوبار کلیک موس اجرا شوند. فایل اجرایی می‌تواند در رایانه‌های دیگر برای اجرا بدون نیاز به محیط لازاروس کپی شود.

نکته

اگر پنجره برنامه فرمانی ظاهر نمی شود، می‌توانید اشکال زدا را از منوی لازاروس غیر فعال کنید:

Environment/Options/Debugger

در قسمت **Debugger type and path** گزینه **(None)** را انتخاب کنید.

مثال‌های دیگر

در برنامه قبلی این خط را تغییر دهید:

```
  Writeln('This is Free Pascal and Lazarus');
```

به این کد:

```
  Writeln('This is a number: ', 15);
```

سپس F۹ را بزنید و برنامه را اجرا کنید.

شما این نتیجه را مشاهده خواهید کرد:

```
This is a number: 15
```

خط قبلی را به صورت‌های زیر تغییر دهید، و برنامه را هر بار اجرا کنید:

کد:

```
  Writeln('This is a number: ', 3 + 2);
```

خروجی:

```
This is a number: 5
```

کد:

```
  Writeln('5 * 2 = ', 5 * 2);
```

خروجی:

```
5 * 2 = 10
```

کد:

```
  Writeln('This is real number: ', 7.2);
```

خروجی:

```
This is real number: 7.2000000000000E+0000
```

کد:

```
  Writeln('One, Two, Three : ', 1, 2, 3);
```

خروجی:

```
One, Two, Three : 123
```

کد:

```
  Writeln(10, ' * ',  3 , ' = ', 10 * 3);
```

خروجی:

```
10 * 3 = 30
```

می‌توانید هر بار مقادیر مختلفی در دستور Writeln بنویسید و نتایج را مشاهده کنید. این کار به شما در درک بهتر این دستور کمک خواهد کرد.

متغیرها

متغیرها برای نگهداری داده‌ها هستند. برای مثال، زمانی که ما X = ۵ قرار می‌دهیم به این معنی است که X یک متغیر محتوی مقدار ۵ است.

پاسکال شئ‌گرا زبانی به شدت ماشینی است، بدین معنی که ما باید نوع متغیر را قبل از اینکه مقداری در آن قرار دهیم معرفی کنیم، اگر ما تعریف کنیم که X یک integer است، به این معنی است که باید فقط اعداد صحیح را در طول عمر X در برنامه به آن نسبت دهید.

مثال‌هایی از تعریف و استفاده‌ی متغیرها:

```
program FirstVar;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes
  { you can add units after this };

var
  x: Integer;
begin
  x:= 5;
  Writeln(x * 2);
  Writeln('Press enter key to close');
  Readln;
end.
```

ما ۱۰ را در خروجی این برنامه دریافت خواهیم کرد.

توجه داشته باشید که ما از کلمه اختصاصی Var استفاده کرده ایم، بدین معنی که خط بعدی آن تعریف متغیرها خواهد بود:

```
  x: Integer;
```

این خط شامل دو مفهوم است:

اول اینکه نام این متغیر X است و

دوم اینکه نوع آن از نوع اعداد صحیح است، که می‌تواند فقط اعداد صحیح بدون اعشار را نگهداری کند. همچنین می‌تواند به آن اعداد مثبت و منفی را نسبت داد.

و عبارت :

```
  x:= 5;
```

به این معنی است که مقدار ۵ را در متقیر X قرار می دهد.

در مثال بعدی ما متغیر Y را اضافه می کنیم:

```
var
  x, y: Integer;
begin
  x:= 5;
  y:= 10;
  Writeln(x * y);
  Writeln('Press enter key to close');
  Readln;
end.
```

خروجی برنامه قبلی به صورت زیر است:

```
50
Press enter key to close
```

نتیجه دستور (x * y) در برنامه ۵۰ است.

در مثال بعدی ما یک نوع جدید داده که character نامیده می‌شود را معرفی می‌کنیم:

```
var
  c: Char;
begin
  c:= 'M';
  Writeln('My first letter is: ', c);
  Writeln('Press enter key to close');
  Readln;
end.
```

این نوع می‌تواند فقط یک حرف، یا یک عدد به صورت یک کارکتر حرفی و نه عددی را نگهداری کند.

در مثال بعدی ما نوع عددی Real را معرفی می‌کنیم، که شامل بخش اعشار نیز می شود.

```
var
  x: Single;
begin
  x:= 1.8;
  Writeln('My Car engine capacity is ', x, ' liters');
```

```
  Writeln('Press enter key to close');
  Readln;
end.
```

برای نوشتن برنامه‌هایی با انعطاف پذیری و تعاملی بیشتر، ما باید بتوانیم ورودی را از کاربر دریافت کنیم. برای مثال ما می‌توانیم از کاربر برای وارد کردن یک عدد سؤال کنیم، و سپس این عدد را از کاربر به وسیله تابع/عبارت **Readln** دریافت کنیم.

```
var
  x: Integer;
begin
  Write('Please input any number:');
  Readln(x);
  Writeln('You have entered: ', x);
  Writeln('Press enter key to close');
  Readln;
end.
```

در این مثال، به جایی اینکه در برنامه یک مقدار ثابت به **X** اختصاص داده شود از طریق کیبورد یک مقدار به آن اختصاص داده شده است.

در مثال زیر ما جدولی از ضرایب یک عدد که توسط کاربر وارد می‌شود را نشان می‌دهیم.

```
program MultTable;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes
  { you can add units after this };

var
  x: Integer;
begin
  Write('Please input any number:');
  Readln(x);
  Writeln(x, ' * 1 = ', x * 1);
  Writeln(x, ' * 2 = ', x * 2);
  Writeln(x, ' * 3 = ', x * 3);
  Writeln(x, ' * 4 = ', x * 4);
  Writeln(x, ' * 5 = ', x * 5);
  Writeln(x, ' * 6 = ', x * 6);
  Writeln(x, ' * 7 = ', x * 7);
  Writeln(x, ' * 8 = ', x * 8);
  Writeln(x, ' * 9 = ', x * 9);
  Writeln(x, ' * 10 = ', x * 10);
  Writeln(x, ' * 11 = ', x * 11);
  Writeln(x, ' * 12 = ', x * 12);
  Writeln('Press enter key to close');
  Readln;
end.
```

توجه داشته باشید که در مثال قبلی تمام متنی که میان سینگل کوتیشن (') مشخص شـده‌اند در پنجـره کنسـول نمـایش داده می‌شود ، برای مثال:

```
' * 1 = '
```

متغیرها و عباراتی که بدون علامت سینگل کوتیشن نوشته می‌شوند ارزش آن‌ها ارزیابی شده و نوشته می شود.

تفاوت‌ها را در میان دو عبارت زیر ببینید:

```
  Writeln('5 * 3');
  Writeln(5 * 3);
```

نتیجه اولین عبارت این است:

```
5 * 3
```

نتیجه‌ای که دومین عبارت ارزیابی می‌کند و نمایش می‌دهد:

```
15
```

در مثال بعدی، ما انجام می‌دهیم عملیات ریاضی را بر روی دو عدد (x, y) و نتایج را در سومین متغیر (Res) قرار خواهیم داد.

```
var
  x, y: Integer;
  Res: Single;
begin
  Write('Input a number: ');
  Readln(x);
  Write('Input another number: ');
  Readln(y);
  Res:= x / y;
  Writeln(x, ' / ', y, ' = ', Res);
  Writeln('Press enter key to close');
  Readln;
end.
```

چون عملیات تقسیم است و نتیجه آن ممکن است عددی با قسمت اعشار باشد، بنا به این دلیل ما متغیـر نـتیجه (Res) را از نوع اعداد حقیقی (Single) تعریف می‌کنیم. Single یک عدد حقیقی با ممیز شناور و دقت عادی است .

انواع زیر گروه‌ها

تعداد زیادی انواع زیر گروه برای متغیر ها وجود دارد، برای مثال، زیر گروه‌های عدد صحیح در بازه و تعداد بایت مورد نیاز برای ذخیره سازی ارزش در حافظه متفاوت هستند.

جدول زیر حاوی انواع عدد صحیح، محدوده ارزش و بایت مورد نیاز در حافظه می باشد:

Type	Min value	Max Value	Size in Bytes
Byte	0	255	1
ShortInt	-128	127	1
SmallInt	-32768	32767	2

Word	0	65535	2
Integer	-2147483648	2147483647	4
LongInt	-2147483648	2147483647	4
Cardinal	0	4294967295	4
Int64	-9223372036854780000	9223372036854775807	8

ما می‌توانیم کمترین و بیشترین ارزش و میزان حافظه را برای هر نوع به ترتیب به وسیله استفاده از توابع **Low**, **High** و **SizeOf** به دست آوریم، همانند مثال زیر:

```
program Types;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes;

begin
  Writeln('Byte: Size = ', SizeOf(Byte),
   ', Minimum value = ', Low(Byte), ', Maximum value = ',
   High(Byte));

  Writeln('Integer: Size = ', SizeOf(Integer),
    ', Minimum value = ', Low(Integer), ', Maximum value = ',
    High(Integer));

  Write('Press enter key to close');
  Readln;
end.
```

انواع دستورات شرطی

یکی از ویژگی‌های خیلی مهم دستگاه‌های هوشمند (شبیه کامپیوترها و دستگاه‌های قابل برنامه ریزی) این هست که آن‌ها می‌توانند اعمال متفاوتی در شرایط مختلف انجام دهند. این کار با استفاده از دستورات شرطی انجام داده می شود. برای مثال برخی اتومبیل ها وقتی سرعت به ۴۰ کیلومتر بر ساعت می‌رسد یا از آن بیشتر می شود درها را قفل می کنند. برای این مورد شرط به این صورت خواهد بود :

اگر سرعت =< از ۴۰ است و درها قفل نیستند سپس آن‌ها را قفل کن.

اتومبیل ها ، ماشین‌های لباس شویی و تعداد زیادی از وسایل محتوی مدارات قابل برنامه‌ریزی شبیه به میکرو کنترلر، یا پردازشگر های کوچک شبیه به ARM هستند. چنین مداراتی میتوانند با توجه به معماری خود به وسیله اسمبلی، سی یا پاسکال رایگان برنامه‌ریزی شوند.

دستور شرطی IF

عبارت شرطی IF در زبان پاسکال بسیار آسان و راحت است. در مثال زیر، ما می‌خواهیم تصمیم بگیریم که با توجه به هوای داخل اتاق، آیا خنک کننده هوا روشن یا خاموش شود.

Air-Conditioner program (برنامه خنک کننده هوا)

```
var
  Temp: Single;
begin
  Write('Please enter Temperature of this room :');
  Readln(Temp);
  if Temp > 22 then
    Writeln('Please turn on air-condition')
  else
    Writeln('Please turn off air-condition');

  Write('Press enter key to close');
  Readln;
end.
```

ما در این مثال عبارات if then else را معرفی کرده ایم، و اگر دما از ۲۲ درجه بیشتر شود اولین جمله نمایش داده میشود:

```
Please turn on air-conditioner
```

وگرنه، اگر شرط اتفاق نیفتد(کمتر یا مساوی با ۲۲)، سپس این خط نمایش داده می شود:

```
Please turn off air-conditioner
```

ما همچنین می‌توانیم شرط های چندگانه بنویسیم:

```
var
  Temp: Single;
begin
  Write('Please enter Temperature of this room :');
  Readln(Temp);
  if Temp > 22 then
    Writeln('Please turn on air-conditioner')
  else
  if Temp < 18 then
    Writeln('Please turn off air-conditioner')
  else
  Writeln('Do nothing');
```

شما می‌توانید مثال قبل را با مقادیر درجه حرارت های مختلف برای دیدن نتایج آزمایش کنید.

ما میتوانیم شرایط را پیچیده‌تر کنیم تا مفیدتر باشند:

```
var
  Temp: Single;
  ACIsOn: Byte;
begin
  Write('Please enter Temperature of this room : ');
```

```
  Readln(Temp);
  Write('Is air conditioner on? if it is (On) write 1,',
   ' if it is (Off) write 0 : ');
  Readln(ACIsOn);

  if (ACIsOn = 1) and (Temp > 22) then
    Writeln('Do nothing, we still need cooling')
  else
  if (ACIsOn = 1) and (Temp < 18) then
    Writeln('Please turn off air-conditioner')
  else
  if (ACIsOn = 0) and (Temp < 18) then
    Writeln('Do nothing, it is still cold')
  else
  if (ACIsOn = 0) and (Temp > 22) then
    Writeln('Please turn on air-conditioner')
  else
    Writeln('Please enter a valid values');

  Write('Press enter key to close');
  Readln;
end.
```

در مثال قبل ما از کلمه جدید (and) استفاده می‌کنیم که به این معنی است که اگر شرط اول مقدار درست را برگرداند (ACIsOn = ۱)، و شرط دوم هم مقدار صحیح را نتیجه دهد (Temp > ۲۲) سپس عبارت *Writeln* اجرا می‌شود. اگر یکی از شرایط یا هر دو آن‌ها مقدار اشتباه (False) را نتیجه دهند، سپس ما به قسمت Else خواهیم رفت.

اگر خنک کننده هوا برای مثال به وسیله پورت سریال به یک رایانه متصل شود، ما می‌توانیم آن را به وسیله برنامه، با استفاده از توابع یا ابزارهای پورت سریال روشن و خاموش کنیم. در این حالت ما به اضافه کردن پارامترهای اضافی برای شرایط if نیاز داریم، که می‌خاهیم برای چه مدت خنک کننده هوا فعال بماند. اگر آن بیش از زمان مجاز (برای مثال ۱ ساعت) فعال بود باید آن را بدون در نظر گرفتن درجه حرارت اتاق خاموش کنیم. همچنین ما می‌توانیم میزان از دست دادن خنکی را بسنجیم که اگر آن خیلی کند است (در شب)، ما می‌توانیم آن را برای زمان طولانی‌تری خاموش کنیم.

Weight program (برنامه وزن)

در این مثال، ما از کاربر قدش را به متر ،و وزن را به کیلو میپرسیم تا وارد کند. سپس برنامه وزن مناسب برای آن شخص را با توجه به اطلاعات ورودی محاسبه خواهد کرد و سپس آن را به او در نتایج خواهد گفت.

```
program Weight;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };
```

```
var
  Height: Double;
  Weight: Double;
  IdealWeight: Double;
begin
  Write('What is your height in meters (e.g. 1.8 meter) : ');
  Readln(Height);
  Write('What is your weight in kilos : ');
  Readln(Weight);
  if Height >= 1.4 then
    IdealWeight:= (Height - 1) * 100
  else
    IdealWeight:= Height * 20;

  if (Height < 0.4) or (Height > 2.5) or (Weight < 3) or
  (Weight > 200) then
  begin
    Writeln('Invalid values');
    Writeln('Please enter proper values');
  end
  else
  if IdealWeight = Weight then
    Writeln('Your weight is suitable')
  else
  if IdealWeight > Weight then
    Writeln('You are under weight, you need more ',
     Format('%.2f', [IdealWeight - Weight]), ' Kilos')
  else
    Writeln('You are over weight, you need to lose ',
     Format('%.2f', [Weight - IdealWeight]), ' Kilos');

  Write('Press enter key to close');
  Readln;
end.
```

در این مثال، ما از این کلمات جدید استفاده کرده ایم:

۱- Double : که هست شبیه به نوع Single . هر دوی آن‌ها اعداد حقیقی هستند، اما نوع Double از دقت ممیز شناور دوبرابر بر خوردار است، و نیاز دارد به ۸ بایت در حافظه، درحالی که نوع single فقط به ۴ بایت نیاز دارد.

۲- دومین چیز جدید کلمه کلیدی OR است و ما از آن در چک کردن یکی از شرایط if که صحیح است یا نه استفاده می‌کنیم. اگر یکی از شرایط درست باشد، عبارت اجرا خواهد شد. برای مثال اگر اولین شرط مقدار صحیح برگردلند (Height < ۰.۴)، در نتیجه عبارت Writeln فراخوانی خواهد شد

```
Writeln('Invalid values');
```

اگر اولین شرط مقدار غلط را نتیجه دهد سپس دومین آن‌ها الی آخر بررسی خواهد شد. اگر تمام شرایط مقدار اشتباه را برگردانند، به قسمت else خواهیم رفت.

۳- ما از کلمه begin و end در دستور if استفاده کرده‌ایم، زیرا عبارت if یک دستور را می‌تواند اجرا کند. Begin و end عبارات چندگانه‌ای که درنظر گرفته به عنوان یک بلوک (عبارت) پوشش می‌دهد، سپس عبارات چندگانه به وسیله شرط if

می‌توانند اجرا شوند . این دو عبارت را ببینید :

```
    Writeln('Invalid values');
    Writeln('Please enter proper values');
```

آن‌ها می‌توانند با استفاده از begin و end در یک عبارت پوشش داده شوند:

```
  if (Height < 0.4) or (Height > 2.5) or (Weight < 3) or
    (Weight > 200) then
    begin
      Writeln('Invalid values');
      Writeln('Please enter proper values');
    end
```

۴- ما از روال format استفاده کرده‌ایم، که ارزش‌ها را در یک قالب خاص نمایش می‌دهد. در این روش ما نیاز داریم تا فقط ۲ عدد بعد از نقطه اعشار نمایش دهیم. ما باید واحد SysUtils را به قسمت Uses به منظور استفاده از این تابع اضافه کنیم.

```
What is your height in meters (e.g. 1.8 meter) : 1.8
What is your weight in kilos : 60.2
You are under weight, you need more 19.80 Kilos
```

نکته:

این مثال ممکن است صددرصد دقیق نباشد. شما می‌توانید جزئیات محاسبه وزن را در وب جستجو کنید. ما قصد داریم فقط تشریح کنیم که چگونه برنامه نویس می‌تواند این مشکلات را حل کند و تجزیه و تحلیل خوبی از موضوع برای تولید برنامه‌های قابل اعتماد انجام دهد.

عبارت شرطی Case ...of

روش دیگری برای شاخه شرطی وجود دارد، که عبارت Case...of است. این شاخه با توجه به ارزش ترتیبی مورد آن را اجرا می کند.برنامه رستوران استفاده از عبارت case of را نشان خواهد داد:

Restaurant program(برنامه رستوران)

```
var
  Meal: Byte;
begin

  Writeln('Welcome to Pascal Restaurant. Please select your order');
  Writeln('1 - Chicken      (10$)');
  Writeln('2 - Fish         (7$)');
  Writeln('3 - Meat         (8$)');
  Writeln('4 – Salad        (2$)');
  Writeln('5 - Orange Juice (1$)');
  Writeln('6 - Milk         (1$)');
  Writeln;
  Write('Please enter your selection: ');
  Readln(Meal);

  case Meal of
```

```
    1: Writeln('You have ordered Chicken,',
        ' this will take 15 minutes');
    2: Writeln('You have ordered Fish, this will take 12 minutes');
    3: Writeln('You have ordered meat, this will take 18 minutes');
    4: Writeln('You have ordered Salad, this will take 5 minutes');
    5: Writeln('You have ordered Orange juice,',
         ' this will take 2 minutes');
    6: Writeln('You have ordered Milk, this will take 1 minute');
  else
    Writeln('Wrong entry');
  end;
  Write('Press enter key to close');
  Readln;
end.
```

اگر ما برنامه را با استفاده از عبارت شرطی if بنویسیم، پیچیده‌تر، و شامل تکرار خواهد بود:

Restaurant program با استفاده از شرط IF

```
var
  Meal: Byte;
begin
  Writeln('Welcome to Pascal restaurant, please select your meal');
  Writeln('1 - Chicken      (10$)');
  Writeln('2 - Fish         (7$)');
  Writeln('3 - Meat         (8$)');
  Writeln('4 - Salad        (2$)');
  Writeln('5 - Orange Juice (1$)');
  Writeln('6 - Milk         (1$)');
  Writeln;
  Write('Please enter your selection: ');
  Readln(Meal);

  if Meal = 1 then
    Writeln('You have ordered Chicken, this will take 15 minutes')
  else
  if Meal = 2 then
    Writeln('You have ordered Fish, this will take 12 minutes')
  else
  if Meal = 3 then
     Writeln('You have ordered meat, this will take 18 minutes')
  else
  if Meal = 4 then
    Writeln('You have ordered Salad, this will take 5 minutes')
  else
  if Meal = 5 then
    Writeln('You have ordered Orange juice,' ,
    ' this will take 2 minutes')
  else
  if Meal = 6 then
    Writeln('You have ordered Milk, this will take 1 minute')
  else
    Writeln('Wrong entry');
```

```
  Write('Press enter key to close');
  Readln;
end.
```

در مثال بعدی، برنامه ارزیابی می‌کند نمره‌های دانش آموزان را و تبدیل می‌کند آن‌ها را به درجه‌هـــای : A و B و C و D و E و F :

Students' Grades program

```
var
  Mark: Integer;
begin
  Write('Please enter student mark: ');
  Readln(Mark);
  Writeln;

  case Mark of
    0 .. 39 : Writeln('Student grade is: F');
    40 .. 49: Writeln('Student grade is: E');
    50 .. 59: Writeln('Student grade is: D');
    60 .. 69: Writeln('Student grade is: C');
    70 .. 84: Writeln('Student grade is: B');
    85 .. 100: Writeln('Student grade is: A');
  else
    Writeln('Wrong mark');
  end;

  Write('Press enter key to close');
  Readln;
end.
```

در مثال قبلی ما از یک بازه استفاده کردیم، شبیه (۰..۳۹)،بدین معنی که شرط در صورتی که مقدار مشخص شده در این بازه قرار داشته باشد مقدار درست را برمیگرداند.

نکته:

عبارت Case فقط با انواع ترتیبی شبیه به اعداد صحیح،کارکتر کار می‌کند،و نمی‌تواند با دیگر انواع داده همچـون رشـته‌ها و اعداد حقیقی کار کند.

Keyboard program

در این مثال، ما یک کارکتر از کیبورد دریافت می‌کنیم و برنامه شماره سطر کلید وارد شده در کیبورد را به ما خواهد گفت:

```
var
  Key: Char;
begin
  Write('Please enter any English letter: ');
  Readln(Key);
  Writeln;
```

```
  case Key of
    'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p':
      Writeln('This is in the second row in keyboard');

    'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l':
      Writeln('This is in the third row in keyboard');

    'z', 'x', 'c', 'v', 'b', 'n', 'm':
      Writeln('This is in the fourth row in keyboard');
  else
    Writeln('Unknown letter');
  end;


  Write('Press enter key to close');
  Readln;
end.
```

توجه داشته باشید که ما یک تکنیک جدید در شرط Case استفاده کرده ایم، که آن مجموعه‌ای از مقادیر است.

```
  'z', 'x', 'c', 'v', 'b', 'n', 'm':
```

که به این معنی است که اگر کلید در یکی از این مقادیر وجود داشت شاخه ای از عبارات Case اجرا می شود:

z, x, c, v, b, n or m

ما می‌توانیم همچنین حالت محدوده و مجموعه مقادیر را شبیه به این ترکیب کنیم:

```
    'a' .. 'd', 'x', 'y', 'z':
```

که به این معنی است که شرط اجرا می‌شود اگر مقدار بین a و d ، یا برابر با x و y یا z باشد.

حلقه‌ها

حلقه‌ها برای اجرای بخش های خاصی از کد استفاده می‌شود (عبارات) برای یک تعداد خاصی از دفعات، یا تا زمانی که یک شرط برقرار شود.

حلقه For

شما یک عبارت را برای یک تعداد خاص از سیکل می‌توانید اجرا کنید با استفاده از شمارنده‌ای شبیه به این مثال:

```
var
  i: Integer;
  Count: Integer;
begin
  Write('How many times? ');
  Readln(Count);
  for i:= 1 to Count do
    Writeln('Hello there');

  Write('Press enter key to close');
  Readln;
end.
```

ما باید از انواع ترتیبی شبیه به اعداد صحیح(integer)،بایت و کارکتر در متغیر حلقه for استفاده کنیم. ما این متغیر را یک متغیر حلقه یا شمارنده حلقه می نامیم. ارزش شمارنده حلقه با هر عددی می‌تولند مقدار دهی اولیه شود، وما همچنین می‌توانیم آخرین مقدار شمارنده حلقه را مشخص کنیم. برای مثال، اگر ما قصد درایم شمارش کنیم از ۵ تا ۱۰، پس باید اینچنین عمل کنیم:

```
  for i:= 5 to 10 do
```

ما می‌توانیم نمایش دهیم شمارنده حلقه را در هر سیکل از حلقه، همانند تغییراتی که در مثال زیر داده شده:

```
var
  i: Integer;
  Count: Integer;
begin
  Write('How many times? ');
  Readln(Count);
  for i:= 1 to Count do
  begin
    Writeln('Cycle number: ', i);
    Writeln('Hello there');
  end;

  Write('Press enter key to close');
  Readln;
end.
```

توجه داشته باشید که در این زمان ما نیاز داریم که دو عبارت را تکرار کنیم، و به همین دلیل ما از کلمات کلیدی ***begin*** و ***end*** استفاده می‌کنیم تا از آن‌ها یک عبارت بسازیم.

جدول ضرب با استفاده از حلقه for

این ویرایش حلقه **for** از جدول ضرب ساده‌تر و خیلی کوتاه‌تر است:

```
program MultTableWithForLoop;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes
  { you can add units after this };

var
  x, i: Integer;
begin
  Write('Please input any number: ');
  Readln(x);
  for i:= 1 to 12 do
    Writeln(x, ' * ', i, ' = ', x * i);

  Writeln('Press enter key to close');
  Readln;
end.
```

به جای نوشتن عبارت **Writeln** با ۱۲ بار، ما آن را درون یک حلقه که ۱۲ اجرا می‌شود بار نوشتیم.

ما می‌توانیم بسازیم عبارت تکراری حلقه **for** را در یک مسیر به عقب با استفاده از کلمه کلیدی **downto** در عوض کلمه کلیدی **to** با استفاده از این ترکیب:

```
for i:= 12 downto 1 do
```

Factorial program

فاکتوریل در علوم ریاضی ضرب یک عدد در اعداد قبل خود تا عدد ۱ هست.برای مثال:

3! = 3 * 2 * 1 = 6

```
var
  Fac, Num, i: Integer;
begin
  Write('Please input any number: ');
  Readln(Num);
  Fac:= 1;
  for i:= Num downto 1 do
    Fac:= Fac * i;
  Writeln('Factorial of ', Num ,' is ', Fac);

  Writeln('Press enter key to close');
  Readln;
end.
```

حلقه تکرار **Repeat Until**

برخلاف حلقه **for** که برای یک تعداد خاص سیکل‌ها را تکرار می‌کند، حلقه **Repeat** شمارنده‌ای ندارد. این حلقه تا زمانی که شرایط خاصی رخ می‌دهد (نتایج شرط درست)، آن را به عبارت بعدی حلقه می برد.

مثال:

```
var
  Num : Integer;
begin
  repeat
    Write('Please input a number: ');
    Readln(Num);
  until Num <= 0;
  Writeln('Finished, please press enter key to close');
  Readln;
end.
```

در مثال قبلی، برنامه وارد حلقه می‌شود، سپس از کاربر می‌خواهد تا عددی وارد کند. اگر عدد کمتر یا مساوی با صفر بود از حلقه خارج خواهد شد. اگر عدد ورودی بزرگ‌تر از صفر بود حلقه ادامه خواهد داشت.

Restaurant program با استفاده از حلقه LOOP

```
var
  Selection: Char;
  Price: Integer;
  Total: Integer;
```

```
begin
  Total:= 0;
  repeat
    Writeln('Welcome to Pascal Restaurant. Please select your order');
    Writeln('1 - Chicken      (10 Geneh)');
    Writeln('2 - Fish         (7 Geneh)');
    Writeln('3 - Meat         (8 Geneh)');
    Writeln('4 – Salad        (2 Geneh)');
    Writeln('5 - Orange Juice (1 Geneh)');
    Writeln('6 - Milk         (1 Geneh)');
    Writeln('X - nothing');
    Writeln;
    Write('Please enter your selection: ');
    Readln(Selection);

    case Selection of
      '1': begin
             Writeln('You have ordered Chicken, this will take 15 minutes');
             Price:= 10;
           end;
      '2': begin
             Writeln('You have ordered Fish, this will take 12 minutes');
             Price:= 7;
           end;
      '3': begin
             Writeln('You have ordered meat, this will take 18 minutes');
             Price:= 8;
           end;
      '4': begin
             Writeln('You have ordered Salad, this will take 5 minutes');
             Price:= 2;
           end;
      '5': begin
             Writeln('You have ordered Orange juice, this will take 2 minutes');
             Price:= 1;
           end;
      '6': begin
             Writeln('You have ordered Milk, this will take 1 minute');
             Price:= 1;
           end;
      else
      begin
        Writeln('Wrong entry');
        Price:= 0;
      end;
    end;

    Total:= Total + Price;

  until (Selection = 'x') or (Selection = 'X');
  Writeln('Total price            = ', Total);
  Write('Press enter key to close');
  Readln;
end.
```

در مثال قبلی، ما از این دو تکنیک استفاده کرده‌ایم:

۱. اضافه کردن ***begin*** و ***end*** در زیر شاخه‌های **case** تا بتوانیم عبارات چندگانه را در یک عبارت پوشش دهیم

۲. ما متغیر **total** را به صفر مقدار دهی اولیه کرده‌ایم (گذاشتن یک مقدار شروع در یک متغیر)، تا مجموع قیمت سفارش‌ها را ذخیره کنیم. سپس ما قیمت انتخاب شده را در هر بار اجرای حلقه با متغیر **total** جمع می‌کنیم:

```
Total:= Total + Price;
```

۳. ما دو گزینه برای اتمام خرید گذاشتیم، هر یک از این‌ها: حرف بزرگ **X**، یا کوچک **x** . هردو کارکتر در ارائه در حافظه

کامپیوتر (ذخیره سازی) متفاوت هستند.

نکته:

ما باید این خط را جابه جا کنیم:

```
  until (Selection = 'x') or (Selection = 'X');
```

با این کد کوتاه شده:

```
  until UpCase(Selection) = 'X';
```

این کار متغیر **Selection** را اگر یک حرف کوچک باشد به حروف بزرگ تغییر خواهد داد، و شرط در هر دو حالت (**X** یا **x**) مقدار **True** (درست) بر خواهد گرداند.

حلقه While

حلقه **while** بسیار شبیه به حلقه **repeat** است، اما با آن از این جنبه‌ها تفاوت دارد:

۱. در **while** اول و قبل از ورود به حلقه شرط چک می‌شود، اما در **repeat** اول وارد حلقه می‌شود و سپس شرط را چک می‌کند. این بدین معنی است که **repeat** همیشه عبارات را حداقل یک بار اجرا می‌کند، اما حلقه **while** در شروع اگر شرط مقدار **false** (غلط) برگرداند می‌تواند از ورود به اولین سیکل حلقه جلوگیری کند.

۲. در حلقه **while** اگر چندین عبارت که لازم است اجرا شوند در حلقه وجود داشته باشد به **begin** و **end** نیاز دارد، اما **repeat** نیازی به **begin** و **end** ندارد، بلوک آن (دستورات تکراری) از کلمه کلیدی **repeat** شروع می‌شوند و با کلمه کلیدی **until** پایان می‌یابد.

مثال:

```
var
  Num: Integer;
begin
  Write('Input a number: ');
  Readln(Num);
  while Num > 0 do
  begin
    Write('From inside loop: Input a number : ');
    Readln(Num);
  end;
  Write('Press enter key to close');
  Readln;
end.
```

برنامه Factorial با استفاده از حلقه While

```
var
  Fac, Num, i: Integer;
begin
  Write('Please input any number: ');
  Readln(Num);
  Fac:= 1;
  i:= Num;
  while i > 1 do
```

```
  begin
    Fac:= Fac * i;
    i:= i - 1;
  end;
  Writeln('Factorial of ', Num ,' is ', Fac);

  Writeln('Press enter key to close');
  Readln;
end.
```

حلقه **while** شمارنده ندارد، و به همین علت ما می‌خواهیم از متغیر **i** استفاده کنیم تا شبیه به شمارنده حلقه عمل کند. مقدار شمارنده حلقه توسط عددی مقدار دهی اولیه می‌شود که ما قصد داریم فاکتوریل آن را محاسبه کنیم، سپس ما یک واحد از آن را به صورت دستی در هر سیکل حلقه کم می‌کنیم. وقتی **i** به ۱ میرسد، حلقه قطع خواهد شد.

رشته‌ها

نوع رشته ای برای تعریف متغیرهایی که می‌توانند یک زنجیره از کارکترها را نگهداری کنند استفاده می‌شود. آن می‌تواند برای ذخیره متن، یک نام، یا یک ترکیبی از کارکترها و اعداد شبیه به یک شماره پلاک ماشین مورد استفاده قرار گیرد.

در این مثال ما خواهیم دید چگونه می‌توانیم متغیر رشته‌ای را برای پذیرش یک نام کاربری استفاده کنیم.

```
var
  Name: string;
begin
  Write('Please enter your name : ');
  Readln(Name);
  Writeln('Hello ', Name);

  Writeln('Press enter key to close');
  Readln;
end.
```

در مثال بعدی، ما از رشته‌ها را برای ذخیره اطلاعات در مورد یک شخص استفاده خواهیم کرد:

```
var
  Name: string;
  Address: string;
  ID: string;
  DOB: string;
begin
  Write('Please enter your name : ');
  Readln(Name);
  Write('Please enter your address : ');
  Readln(Address);
  Write('Please enter your ID number : ');
  Readln(ID);
  Write('Please enter your date of birth : ');
  Readln(DOB);
  Writeln;
  Writeln('Card:');
  Writeln('-----------------------------------------');
  Writeln('| Name    : ', Name);
  Writeln('| Address : ', Address);
  Writeln('| ID      : ', ID);
  Writeln('| DOB     : ', DOB);
  Writeln('-----------------------------------------');
```

```
  Writeln('Press enter key to close');
  Readln;
end.
```

رشته‌ها می‌توانند برای تولید رشته‌های بزرگ‌تر به هم متصل شوند. برای مثال ما می‌توانیم الحاق کنیم ***FirstName*** و ***SecondName*** و ***FamilyName*** را در رشته‌ی دیگری با نام ***FullName***، شبیه به مثال بعدی :

```
var
  YourName: string;
  Father: string;
  GrandFather: string;
  FullName: string;
begin
  Write('Please enter your first name : ');
  Readln(YourName);
  Write('Please enter your father name : ');
  Readln(Father);
  Write('Please enter your grand father name : ');
  Readln(GrandFather);
  FullName:= YourName + ' ' + Father + ' ' + GrandFather;
  Writeln('Your full name is: ', FullName);

  Writeln('Press enter key to close');
  Readln;
end.
```

توجه کنید که در این مثال ما یک فضای خالی میان نام‌ها (شبیه ***YourName*** + ' ' + ***Father***) برای ساختن یک تفکیک کننده میان نام‌ها اضافه کردیم. **Space** (فضای خالی) نیز یک کارکتر است.

ما می‌توانیم انجام دهیم عملیات های زیادی بر روی رشته‌ها، شبیه به جستجو برای کلمه‌ای در یک رشته، کپی کردن یک رشته در یک متغیر رشته‌ای دیگر، یا تبدیل کرکترهای متن به حروف بزرگ یا کوچک شبیه به مثال زیر:

این خط ارزش رشته‌ای حروف را در ***FullName*** به حروف بزرگ تبدیل می‌کند:

```
  FullName:= UpperCase(FullName);
```

و این یکی به حروف کوچک تبدیل می‌کند :

```
  FullName:= LowerCase(FullName);
```

در مثال بعدی، ما می‌خواهیم با استفاده از تابع **Pos** در یک نام کاربری حرف **a** را جستجو کنیم. تابع **Pos** اولین محل وجود(شاخص) کارکتر در رشته را برمی‌گرداند، یا صفر را برمی‌گرداند اگر که حرف یا زیر رشته جستجو شده در متن وجود نداشته باشد:

```
var
  YourName: string;
begin
  Write('Please enter your name : ');
  Readln(YourName);
  If Pos('a', YourName) > 0 then
    Writeln('Your name contains a')
  else
    Writeln('Your name does not contain a letter');

  Writeln('Press enter key to close');
  Readln;
end.
```

اگر نام حاوی حرف **A** بزرگ باشد، پس تابع **Pos** نمیتواند به آن اشاره کند، چون **A** متفاوت است از **a** که قبلاً ذکر شد.

برای حل این مشکل می‌توان تبدیل کرد همه حروف نام کاربر را به حروف کوچک و پس از آن ما می‌توانیم جستجو را انجام دهیم:

```
  If Pos('a', LowerCase(YourName)) > 0 the
```

در ویرایش کد بعدی، ما می‌خواهیم موقعیت حرف **a** را در یک نام کاربری نمایش دهیم:

```
var
  YourName: string;
begin
  Write('Please enter your name : ');
  Readln(YourName);
  If Pos('a', LowerCase(YourName)) > 0 then
  begin
    Writeln('Your name contains a');
    Writeln('a position in your name is: ',
      Pos('a', LowerCase(YourName)));
  end
  else
    Writeln('Your name does not contain a letter');

  Write('Press enter key to close');
  Readln;
end.
```

شما باید توجه کنید که اگر نام حاوی بیش از یک حرف **a** باشد، تابع **Pos** اولین محل وجود حرف **a** را در نام کاربری بر خواهد گرداند.

شما می‌توانید تعداد کارکترهای یک رشته (طول رشته) را با استفاده از تابع **Length** دریافت کنید.

```
Writeln('Your name length is ', Length(YourName), ' letters');
```

و شما می‌توانید اولین حرف/کارکتر از یک رشته را با استفاده از عدد شاخص آن بگیرید:

```
Writeln('Your first letter is ', YourName[1]);
```

و دومین کارکتر :

```
Writeln('Your second letter is ', YourName[2]);
```

و آخرین کارکتر :

```
Writeln('Your last letter is ', YourName[Length(YourName)]);
```

شما همچنین می‌توانید یک متغیر رشته ای را کارکتر به کارکتر با استفاده از حلقه **for** نمایش دهید:

```
  for i:= 1 to Length(YourName) do
    Writeln(YourName[i]);
```

تابع Copy

ما می‌توانیم بخشی از یک رشته را با استفاده از تابع Copy کپی کنیم. برای مثال، اگر ما بخواهیم کلمه 'world' را از رشته 'hello world' استخراج کنیم،می‌توانیم آن را با اطلاع از موقعیت آن بخش انجام دهیم، همانطور که ما در مثال زیر انجام داده ایم:

```
var
  Line: string;
  Part: string;
begin
  Line:= 'Hello world';

  Part:= Copy(Line, 7, 5);

  Writeln(Part);

  Writeln('Press enter key to close');
  Readln;
end.
```

توجه داشته باشید که ما از تابع کپی با این ترکیب استفاده کرده‌ایم:

```
  Part:= Copy(Line, 7, 5);
```

در اینجا عبارت بالا را توضیح می دهیم:

- Part:= این متغیر رشته ای هست که نتایج تابع را در آن گذاشته ایم.
- Line این رشته‌ی اصلی است که حاوی جمله 'Hello world' است.
- ۷ این اشاره گر رشته یا شماره زیر رشته ای است که ما نیاز داریم آن را استخراج کنیم، در این مورد آن کارکتر W است.
- ۵ این طول بخش استخراج شده است. در این مورد طول کلمه 'world' را بیان می‌کند.

در مثال بعدی، ما از کاربر سؤال خواهیم کرد تا نام یک ماه را وارد کند، شبیه به February ، سپس برنامه ویرایش کوتاهی از آن را شبیه به Feb تایپ می‌کند:

```
var
  Month: string;
  ShortName: string;
begin
  Write('Please input full month name e.g. January : ');
  Readln(Month);

  ShortName:= Copy(Month, 1, 3);

  Writeln(Month, ' is abbreviated as : ', ShortName);

  Writeln('Press enter key to close');
  Readln;
end.
```

روال Insert

روال insert یک زیر رشته را در یک رشته وارد می کند. برخلاف عملگر الحاق (+) که دو زیر رشته را با هم پیوند می‌زند، insert یک زیر رشته را در وسط رشته‌ی دیگر اضافه می‌کند.

برای مثال، ما میتوانیم کلمه Pascal را داخل رشته 'Hello world' وارد کنیم، و نتیجه عبارت 'Hello Pascal world' است شبیه به مثال زیر:

```
var
  Line: string;
begin
  Line:= 'Hello world';

  Insert('Pascal ', Line, 7);

  Writeln(Line);

  Writeln('Press enter key to close');
  Readln;
end.
```

پارامترهای روال Insert اینچنین اند:

- pascal این زیر رشته‌ای است که ما قصد داریم در رشته مورد نظر وارد کنیم.
- Line این رشته ی نهایی است که شامل نتایج عملیات خواهد بود.
- ۷ این موقعیت شروع وارد کردن در رشته نهایی است. در این مورد این موقعیت بعد از هفت کـارکتر خواهـد بـود، که اون اولین فضای خالی از 'Hello world' است.

روال Delete

این تابع برای حذف یک کارکتر یا زیر رشته از یک رشته استفاده می‌شود. ما نیاز داریم تا شروع موقعیت و طول زیـر رشـته ای که باید حذف شود بدانیم.

برای مثال، اگر ما نیاز داشته باشیم تا حروف ll را از رشـته 'Hello world' بـرای سـاختن 'Heo world' حـذف کنیـم، میتوانیم شبیه به این عمل کنیم:

```
var
  Line: string;
begin
  Line:= 'Hello world';
  Delete(Line, 3, 2);
  Writeln(Line);
  Writeln('Press enter key to close');
  Readln;
end.
```

تابع Trim

این تابع برای حذف فضای خالی از شروع و پایان رشته‌ها استفاده می‌شود. اگر ما رشته‌ای حاوی متن ' Hello ' داشته باشیم بعد از استفاده از این تابع آن رشته 'Hello' خواهد شد.

ما نمی‌توانیم فضای خالی را در یک پنجره ترمینال نمایش دهیم مگر اینکه کارکترها را میان آن‌ها بگذاریم.
ببینید مثال بعدی را:

```
program TrimStr;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };

var
  Line: string;
begin
  Line:= ' Hello ';

  Writeln('<', Line, '>');

  Line:= Trim(Line);

  Writeln('<', Line, '>');

  Writeln('Press enter key to close');
  Readln;
end.
```

در مثال فوق، ما از واحد SysUtils استفاده می کنیم، که حاوی تابع Trim است.

دو تابع دیگر وجود دارد که فضای خالی را فقط از یک سمت رشته، قبل/بعد حذف می‌کنند. این توابع TrimRight و TrimLeft هستند.

تابع StringReplace

تابع StringReplace کارکترها یا زیر رشته‌ها را با کارکترها یا زیر رشته‌های دیگر در رشته‌ی مطلوب جایگزین می‌کند.

```
program StrReplace;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };
```

```
var
  Line: string;
  Line2: string;
begin
  Line:= 'This is a test for string replacement';
  Line2:= StringReplace(Line, ' ', '-', [rfReplaceAll]);
  Writeln(Line);
  Writeln(Line2);
  Write('Press enter key to close');
  Readln;
end.
```

پارامترهای تابع StringReplace اینچنین‌اند:

۱. Line : این رشته اصلی است که ما نیاز داریم تا ویرایش شود.

۲. ' ' : این زیر رشته ای است که ما می‌خواهیم آن را جایگزین کنیم. در این مثال این رشته فضای خالی است.

۳. '-' :این دیگر زیر رشته ای است که ما قصد داریم به جای قبلی در رشته اصلی جایگزین کنیم.

۴. [rfReplaceAll] : این نوع جایگزینی است. در این مورد ما می‌خواهیم همه فضاهای خالی زیر رشته جایگزین شوند.

ما می‌توانیم فقط از یک متغیر رشته‌ای استفاده کنیم و متغیر Line۲ را حذف کنیم همانند مثال اصلاح شده که نمایش داده می شود، اما مقدار متن اصلی را از دست خواهیم داد.

```
var
  Line: string;
begin
  Line:= 'This is a test for string replacement';
  Writeln(Line);
  Line:= StringReplace(Line, ' ', '-', [rfReplaceAll]);
  Writeln(Line);
  Write('Press enter key to close');
  Readln;
end.
```

آرایه‌ها

یک آرایه یک سلسله از متغیرهای هم‌نوع است. اگر ما بخواهیم یک آرایه از ۱۰ متغیر عدد صحیح (Integer) تعریف کنیم، می‌توانیم اینچنین عمل کنیم:

```
  Numbers: array [1 .. 10] of Integer;
```

ما می‌توانیم با استفاده از شاخص به هریک از متغیرهای آرایه دسترسی داشته باشیم. برای مثال، برای اینکه اولین متغیر در آرایه را مقدار دهی کنیم می‌توانیم اینگونه آن را بنویسیم:

```
  Numbers[1]:= 30;
```

برای قرار دادن مقدار در دومین متغیر، از شاخص ۲ استفاده کنید:

```
  Numbers[2]:= 315;
```

در مثال بعدی، ما از کاربر می‌خواهیم ۱۰ نمره دانش آموز را وارد کند و آن‌ها را در یک آرایه قرار می‌دهیم. سرانجام ما از

طریق آن‌ها نتایج Pass/fail را استخراج خواهیم کرد:

```
var
  Marks: array [1 .. 10] of Integer;
  i: Integer;
begin
  for i:= 1 to 10 do
  begin
    Write('Input student number ', i, ' mark: ');
    Readln(Marks[i]);
  end;

  for i:= 1 to 10 do
  begin
    Write('Student number ', i, ' mark is : ', Marks[i]);
    if Marks[i] >= 40 then
      Writeln(' Pass')
    else
      Writeln(' Fail');
  end;

  Writeln('Press enter key to close');
  Readln;
end.
```

ما می‌توانیم کد قبلی را ویرایش کنیم تا بیشترین و کمترین نمره دانش آموز را بگیریم:

```
var
  Marks: array [1 .. 10] of Integer;
  i: Integer;
  Max, Min: Integer;
begin

  for i:= 1 to 10 do
  begin
    Write('Input student number ', i, ' mark: ');
    Readln(Marks[i]);
  end;

  Max:= Marks[1];
  Min:= Marks[1];

  for i:= 1 to 10 do
  begin
    // Check if current Mark is maximum mark or not
    if Marks[i] > Max then
      Max:= Marks[i];

    // Check if current value is minimum mark or not
    if Marks[i] < Min then
      Min:= Marks[i];

    Write('Student number ', i, ' mark is : ', Marks[i]);
    if Marks[i] >= 40 then
      Writeln(' Pass')
    else
      Writeln(' Fail');
  end;

  Writeln('Max mark is: ', Max);
  Writeln('Min mark is: ', Min);
  Writeln('Press enter key to close');
  Readln;
end.
```

توجه داشته باشید که ما اولین نمره (Marks[۱]) را به عنوان بیشترین و کمترین نمره در نظر می‌گیریم، و سپس آن را با دیگر نمره‌ها مقایسه می‌کنیم.

```
Max:= Marks[1];
Min:= Marks[1];
```

در داخل حلقه ما Max و Min را با هر نمره مقایسه می‌کنیم. اگر ما یک عدد که از Max بزرگ‌تر است پیدا کردیم، ارزش آن را با Max جایگزین می‌کنیم.اگر هم عددی کوچکتر از Min یافتیم مقدار آن را در Min قرار می دهیم.

در مثال قبلی ما توضیحات معرفی داشتیم:

```
   // Check if current Mark is maximum mark or not
```

ما خط را با کارکترهای // شروع کرده‌ایم، که بدین معنیست که این خط یک توضیح است و تأثیر نخواهد گذاشت بر روی کد و کامپایل نخواهد شد. این تکنیک استفاده شده است تا برای برنامه نویس‌های دیگر یا برای خود برنامه نویس بخشی از کد را توصیف کند.

// برای توضیحات کوتاه استفاده می‌شود. اگر نیاز دارید تا توضیحات چند خطی بنویسید می‌توانید آن‌ها را به وسیله {} یا (**) محصور کنید.

مثال :

```
for i:= 1 to 10 do
begin
  { Check if current Mark is maximum mark or not
    check if Mark is greater than Max then put
    it in Max }
  if Marks[i] > Max then
    Max:= Marks[i];

  (* Check if current value is minimum mark or not
     if Min is less than Mark then put Mark value in Min
  *)
  if Marks[i] < Min then
    Min:= Marks[i];

  Write('Student number ', i, ' mark is : ', Marks[i]);
  if Marks[i] >= 40 then
    Writeln(' Pass')
  else
    Writeln(' Fail');
end;
```

ما می‌توانیم همچنین بخشی از کد را موقتاً به وسیله توضیحی کردن آن غیر فعال کنیم:

```
 Writeln('Max mark is: ', Max);
// Writeln('Min mark is: ', Min);
 Writeln('Press enter key to close');
 Readln;
```

در قسمت بالا، ما روال را برای نوشتن کمترین نمره دانش آموز غیر فعال کرده‌ایم.

نکته:

ما می‌توانیم یک آرایه شبیه به زبان C که یک شاخص پایه صفر داشته باشد معرفی کنیم :

```
  Marks: array [0 .. 9] of Integer;
```

این هم می‌تواند ۱۰ عنصر نگهداری کند، اما مورد اول با استفاده از شاخص ۰ قابل دسترس است:

```
  Numbers[0]:= 30;
```

دومین عنصر :

```
  Numbers[1]:= 315;
```

آخرین عنصر :

```
  Numbers[9]:= 10;
```

یا

```
  Numbers[High(Numbers)]:= 10;
```

رکوردها

مادامی که آرایه‌ها تعداد زیادی متغیرهای هم‌نوع نگهداری می‌کنند، رکوردها متغیرهایی از انواع مختلف را نگهداری می‌کنند، و این متغیرها فیلدها نامیده می‌شوند.

این گروه از متغیرها/فیلدها می‌توانند به عنوان یک واحد منفرد یا متغیر به هم مربوط باشند. ما می‌توانیم از رکوردها برای ذخیره کردن اطلاعاتی که متعلق است به اشیاء مشابه استفاده کنیم، برای مثال، اطلاعات اتومبیل:

۱. نوع اتومبیل : متغیر رشته‌ای

۲. اندازه موتور : عدد حقیقی

۳. سال تولید : عدد صحیح

ما می‌توانیم این انواع مختلف را گرد آوری کنیم در یک رکورد تا یک اتومبیل را همانند مثال بعدی نمایش دهیم:

```
program Cars;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes
  { you can add units after this };

type
  TCar = record
    ModelName: string;
    Engine: Single;
    ModelYear: Integer;
  end;
```

```
var
  Car: TCar;
begin
  Write('Input car Model Name: ');
  Readln(Car.ModelName);
  Write('Input car Engine size: ');
  Readln(Car.Engine);
  Write('Input car Model year: ');
  Readln(Car.ModelYear);

  Writeln;
  Writeln('Car information: ');
  Writeln('Model Name  : ', Car.ModelName);
  Writeln('Engine size : ', Car.Engine);
  Writeln('Model Year  : ', Car.ModelYear);

  Write('Press enter key to close..');
  Readln;
end.
```

در این مثال، ما یک نوع جدید (رکورد) با استفاده از کلمه کلیدی 'type' تعریف کرده‌ایم:

```
type
  TCar = record
    ModelName: string;
    Engine: Single;
    ModelYear: Integer;
  end;
```

ما حرف (T) را به Car اضافه کرده‌ایم تا اشاره کنیم به اینکه این یک نوع است و یک متغیر نیست. نام‌های متغیر می‌توانند شبیه باشند به : Car,Hour,UserName اما نام‌های انواع باید باشند شبیه به : Tcar,THouse,TuserName . این یک استاندارد در زبان پاسکال است.

وقتی ما نیاز داریم این نوع جدید را استفاده کنیم، در این هنگام ما باید یک متغیر از آن نوع تعریف کنیم، برای مثال :

```
var
  Car: TCar;
```

اگر ما نیاز داشتیم در یکی از این متغیرها/فیلدها مقداری ذخیره کنیم، باید مانند این به آن دسترسی پیدا کنیم:

```
  Car.ModelName
```

رکوردها در فصل فایل‌ها با دسترسی تصادفی در این کتاب استفاده خواهند شد.

فایل‌ها

فایل‌ها مهمترین اصل از سیستم عامل و برنامه‌ها هستند. اجزا سیستم عامل به عنوان فایل‌ها نشان داده شده‌اند، همچنین اطلاعات و داده‌ها نیز شبیه به تصاویر،کتاب‌ها، برنامه‌ها، و فایل‌های متنی ساده اینچنین اند.

سیستم عامل‌ها بر مدیریت فایل‌ها همانند: خواندن، نوشتن، ویرایش کردن و حذف فایل‌ها نظارت می‌کنند.

فایل‌ها به بسیاری از انواع با توجه به دیدگاه‌های زیادی تقسیم می شوند. ما می‌توانیم فایل‌ها را در دو نوع گروه بندی کنیم :

فایل‌های اجرایی، و فایل‌های داده. برای مثال برنامه‌های کاربردی لازاروس که به صورت باینری کامپایل شده‌اند فایل‌های اجرایی هستند، با این حال متن برنامه پاسکال (pas.) فایل‌های داده هستند. همچنین کتاب‌های PDF ، تصاویر JPEG فایل‌های داده هستند.

ما می‌توانیم فایل‌های داده‌ای را با توجه به نمایش محتویاتشان در دو نوع تقسیم کنیم:

۱. فایل‌های متنی : که فایل‌های متنی ساده‌ای هستند که می‌توانند با استفاده از هر ابزار ساده‌ای داخل خط فرمان سیستم عامل همانند Cat درفرامین لینوکس و Copy con ، Type در فرمان‌های ویندوز نوشته، یا خوانده شوند.

۲. فایل‌های داده باینری : این‌ها پیچیده‌تر هستند و به برنامه‌های خاصی برای باز کردن آن‌ها نیاز است. برای مثال، فایل‌های تصویری با استفاده از ابزارهای ساده خط فرمان باز نمی شوند، در عوض آن‌ها باید با استفاده از برنامه‌هایی شبیه به ، GIMP MS Paint ، Kolour Paint و غیره باز شوند. اگر ما فایل‌های تصویر، یا صدا را باز کنیم، کارکترهای غیر قابل تشخیص که به معنی هیچ چیزی برای کاربر نیستند دریافت خواهیم کرد. مثالی از فایل‌های داده باینری، فایل‌های پایگاه داده هستند، که باید با استفاده از برنامه مناسب باز شوند.

روش دیگری برای دسته بندی فایل‌ها وجود دارد که با توجه به انواع دسترسی است:

۱. فایل‌ها با دسترسی ترتیبی : فایل متنی یک مثال از یک فایل ترتیبی است، که رکوردی با اندازه ثابت ندارد. هر خط دارای طول خود است، بنابرین برای مثال ما موقعیت (در کارکترها) برای شروع خط سوم را نمی‌دانیم. به همین علت، ما می‌توانیم فایل را فقط برای خواندن، یا فقط برای نوشتن، باز کنیم و همچنین می‌توانیم متن را فقط به انتهای فایل اضافه کنیم. اگر ما نیاز داشتیم تا متن را در میان یک فایل وارد کنیم، بایدمحتویات فایل را در حافظه بخوانیم، تغییرات را انجام دهیم، داخل فایل را در دیسک پاک کنیم، و سپس آن را در فایل با متن ویرایش شده باز نویسی کنیم.

۲. فایل‌ها با دسترسی تصادفی : این نوع از فایل رکورد با اندازه ثابت دارد. یک رکورد کوچکترین واحدی است که می‌توانیم در یک لحظه بخوانیم و بنویسیم، و آن باید string ، Integer ، byte ، یا یک رکورد تعریف شده کاربر باشد. ما می‌توانیم در همان زمان بخوانیم و بنویسیم، برای مثال، می‌توانیم رکورد شماره ۳ را بخوانیم و آن را در رکورد شماره ۱۰ کپی کنیم. در این مورد، ما دقیقاً موقعیت هر رکورد را در فایل می‌توانیم بفهمیم. اصلاح رکوردها بسیار ساده است. در فایل‌های با دسترسی تصادفی، ما می‌توانیم هر رکورد را بدون تأثیر بر بقیه فایل جایگزین/بازنویسی کنیم.

فایل‌های متنی

فایل‌های متنی ساده‌ترین فایل‌ها هستند، اما باید در آن‌ها فقط در یک جهت (فقط رو به جلو) بنویسیم. ما نمیتوانیم به عقب برگردیم تا زمانی که در یک فایل متنی می نویسیم. همچنین باید نوع عمل را قبل از باز کردن فایل تعریف کنیم: خواندن، نوشتن، یا افضودن (نوشتن در انتهای فایل).

در این مثال، ما محتوای یک فایل انتخاب شده به وسیله کاربر را نمایش خواهیم داد. برای مثال، یک کاربر ممکن است یک فایل متنی شبیه به نام C:\test\first.pas در ویندوز یا home/user/first.pas/ در لینوکس داشته باشد:

برنامه خواندن فایل متنی

```
program ReadFile;

{$mode objfpc}{$H+}
uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, sysUtils
  { you can add units after this };

var
  FileName: string;
  F: TextFile;
  Line: string;
begin
  Write('Input a text file name: ');
  Readln(FileName);
  if FileExists(FileName) then
  begin
    // Link file variable (F) with physical file (FileName)
    AssignFile(F, FileName);

    Reset(F); // Put file mode for read, file should exist

    // while file has more lines that does not read yet do the loop
    while not Eof(F) do
    begin
      Readln(F, Line); // Read a line from text file
      Writeln(Line);   // Display this line in user screen
    end;
    CloseFile(F); // Release F and FileName connection
  end
  else // else if FileExists..
    Writeln('File does not exist');
  Write('Press enter key to close..');
  Readln;
end.
```

در لینوکس ما می‌توانیم این نامهای فایل را وارد کنیم:

```
/etc/resolv.conf
```

یا

```
/proc/meminfo
/proc/cpuinfo
```

ما از نوع ها، توابع و روال های جدید برای دستکاری کردن فایل‌های متنی استفاده می‌کنیم که هستند :

۱.

```
  F: TextFile;
```

TextFile یک نوع است که برای تعریف یک متغیر فایل متنی استفاده شده است. این متغیر می‌تواند برای استفاده بعدی به یک فایل متنی ارتباط داده شده باشد.

۲.

```
  if FileExists(FileName) then
```

FileExists یک تابع است که در واحد SysUtils آن را شامل می‌شود. این تابع موجودیت یک فایل را بررسی می‌کند. اگر

فایل در وسیله ذخیره سازی وجود داشت این تابع مقدار True را برمی‌گرداند.

۳.

```
  AssignFile(F, FileName);
```

بعد از اینکه موجودیت فایل رابررسی کردیم،می‌توانیم از روال AssignFile برای ارتباط بین متغیر فایل (F) با فایل فیزیکی استفاده کنیم. هر استفاده دیگری از متغیر F فایل فیزیکی را نمایش خواهد داد.

۴.

```
  Reset(F); // Put file mode for read, file should exist
```

روال Reset فایل‌های متنی را فقط برای خواندن باز می‌کند، و اشاره‌گر خواندن را در اولین کاراکتر از فایل قرار می‌دهد.

اگر اجازه خواندن توسط کاربر فعلی برای آن فایل وجود نداشته باشد، یک پیغام خطا ظاهر می‌شود، مانند دسترسی ممنوع است.

۵.

```
  Readln(F, Line); // Read a line from text file
```

روال Readln برای خواندن یک خط کامل از فایل و قرار دادن آن در متغیر Line استفاده شده است. قبل از خواندن یک خط از فایل متنی، ما باید مطمئن شویم که فایل به پایان نرسیده است. ما می‌توانیم با تابع بعدی، EOF این کار را انجام دهیم.

۶.

```
  while not Eof(F) do
```

تابع EOF اگر فایل به پایان رسیده باشد مقدار True بر می‌گرداند. این تابع به این دلیل استفاده می‌شود تا نشان دهد که عمل خواندن دیگر نمی‌تواند مورد استفاده قرار گیرد و ما تمام محتویات فایل را خوانده‌ایم.

۷.

```
  CloseFile(F); // Release F and FileName connection
```

پس از پایان خواندن یا نوشتن در یک فایل، ما باید برای آزاد کردن فایل آن را ببندیم، زیرا روال Rest فایل را در سیستم عامل رزرو می‌کند و از نوشتن و حذف توسط برنامه دیگری تا زمانی که فایل باز است جلوگیری می‌کند.

ما باید از CloseFile تنها زمانی که روال Reset در باز کردن فایل موفق بوده است استفاده کنیم. اگر Reset موفق نباشد (به عنوان مثال، اگر فایل وجود ندشته باشد، و یا توسط برنامه دیگری شروع به استفاده شده است) در این صورت نباید فایل را ببندیم.

در مثال بعدی ما یک فایل متنی جدید ایجاد خواهیم کرد و متن در آن قرار می‌دهیم :

ایجاد و نوشتن در فایل متنی

```
var
  FileName: string;
  F: TextFile;
  Line: string;
  ReadyToCreate: Boolean;
  Ans: Char;
  i: Integer;
```

```
begin
  Write('Input a new file name: ');
  Readln(FileName);

  // Check if file exists, warn user if it is already exist
  if FileExists(FileName) then
  begin

    Write('File already exist, did you want to overwrite it? (y/n)');
    Readln(Ans);
    if upcase(Ans) = 'Y' then
      ReadyToCreate:= True
    else
      ReadyToCreate:= False;
  end
  else // File does not exist
    ReadyToCreate:= True;

  if ReadyToCreate then
  begin
    // Link file variable (F) with physical file (FileName)
    AssignFile(F, FileName);

    Rewrite(F); // Create new file for writing

    Writeln('Please input file contents line by line, '
     , 'when you finish write % then press enter');
    i:= 1;
    repeat
      Write('Line # ', i, ':');
      Inc(i);
      Readln(Line);
      if Line <> '%' then
        Writeln(F, Line); // Write line into text file
    until Line = '%';

    CloseFile(F); // Release F and FileName connection, flush buffer
  end
  else // file already exist and user does not want to overwrite it
    Writeln('Doing nothing');
  Write('Press enter key to close..');
  Readln;
end.
```

در این مثال، ما بسیاری از چیزهای مهم را استفاده کرده‌ایم :

۱. نوع Boolean :

```
  ReadyToCreate: Boolean;
```

این نوع می‌تواند تنها یکی از دو مقدار را نگه دارد: درست یا نادرست. این ارزش‌ها را می‌توان به طور مستقیم در شرط if، حلقه while، یا حلقه loop استفاده نمود.

در مثال قبل، ما شرط if را مانند این استفاده کردیم:

```
  if Marks[i] > Max then
```

که در نهایت به درست یا غلط تبدیل می شود.

۲. تابع UpCase :

```
  if upcase(Ans) = 'Y' then
```

این دستورات زمانی که فایل وجود دارد اجرا شده است. این برنامه به کاربر در مورد رونویسی در یک فایل موجود هشدار می‌دهد. اگر او بخواهد ادامه دهد، پس باید حرف کوچک y یا بزرگ Y را وارد کند. تابع Upcase اگر آن حرف کوچک باشد آن را به حرف بزرگ تبدیل خواهد کرد.

۳. روال Rewite :

```
  Rewrite(F); // Create new file for writing
```

روال بازنویسی برای ایجاد یک فایل جدید خالی مورد استفاده قرار گرفته است. اگر فایل از قبل وجود داشت، آن را پاک و رونویسی می‌شود. همچنین فایل را تنها در مورد فایل‌های متنی برای نوشتن باز می کند.

۴. روال Writeln :

```
  Writeln(F, Line); // Write line into text file
```

این روال برای نوشتن رشته یا متغیر در فایل متنی استفاده شده است، و به آنها کارکتر پایان خط را اضافه می‌کند، که ترکیبی از بازگشت به ابتدای خط/تعویض سطر (CR / LF) است، که به ترتیب کارکترهای شماره ۱۳ و ۱۰ آن را بیان می‌کنند.

این کارکتر در یک پنجره کنسول نشان داده نمی شود، اما آن مکان نما را به یک خط جدید در صفحه نمایش حرکت می‌دهد.

۵. روال Inc :

```
  Inc(i);
```

این روال یک متغیر عدد صحیح را یکی افزایش می‌دهد. این معادل این دستور است:

```
  i:= i + ۱;
```

۶. روال CloseFile :

```
   CloseFile(F); // Release F and FileName connection, flush buffer
```

همانطور که قبلا ذکر شد، روال CloseFile یک فایل را در سیستم عامل آزاد می‌کند. علاوه بر این، آن در هنگام نوشتن در یک فایل متنی یک کار اضافی دارد، که آن دور ریختن بافر نوشتن است.

بافر کردن فایل‌های متنی یک ویژگی است که توزیع فایل‌های متنی را سریعتر می سازد. به جای نوشتن یک خط یا یک کاراکتر به طور مستقیم بر روی دیسک و یا هر رسانه ذخیره سازی دیگر (که در مقایسه با نوشتن در حافظه خیلی کند است)، برنامه کاربردی این نوشته را در یک بافر حافظه می نویسد. وقتی که بافر به اندازه کامل خود می‌رسد (پر می شود)، آن به رسانه های ذخیره سازی دائمی مانند هارد دیسک سر ریز خواهد شد (مجبور است که نوشته شود). این عمل باعث می شود نوشتن سریع‌تر شود، اما به آن خطر از دست دادن برخی از داده‌ها اضافه خواهد شد (در بافر) اگربرق به طور ناگهانی از دست داده شود. برای به حداقل رساندن از دست دادن داده ها، ما باید فایل را بلافاصله پس از اتمام نوشتن در آن ببندیم، و یا روال Flush را فراخوانی کنیم تا به وضوح بافر را تخلیه کنیم.

الحاق به یک فایل متنی

در این مثال، ما قصد داریم یک فایل متنی موجود را برای نوشتن در انتهای آن باز کنیم، با استفاده از روال Append بدون اینکه محتوای اصلی آن حذف شود.

برنامه افزودن به فایل متنی

```
var
  FileName: string;
  F: TextFile;
  Line: string;
  i: Integer;
begin
  Write('Input an existed file name: ');
  Readln(FileName);
  if FileExists(FileName) then
  begin
    // Link file variable (F) with physical file (FileName)
    AssignFile(F, FileName);

    Append(F); // Open file for appending

    Writeln('Please input file contents line by line',
      'when you finish write % then press enter');
    i:= 1;
    repeat
      Write('Line # ', i, ' append :');
      Inc(i);
      Readln(Line);
      if Line <> '%' then
        Writeln(F, Line); // Write line into text file
    until Line = '%';
    CloseFile(F); // Release F and FileName connection, flush buffer
  end
  else
    Writeln('File does not exist');
  Write('Press enter key to close..');
  Readln;
end.
```

بعد از اینکه این برنامه را اجرا و یک فایل متنی را وارد کنیم ، ما می‌توانیم آن را با دستور cat/type مشاهده کنیم، یا به وسیله دابل کلیک کردن بر روی آن در پوشه‌ای که در آن هست برای اینکه داده‌های اضافه شده را ببینیم.

فایل‌ها با دسترسی تصادفی

همانطور که قبلاً ذکر شد،دومین نوع فایل بر طبق یک دید نوع دسترسی، دسترسی تصادفی، یا دسترسی مستقیم است. این نوع از فایل دارد یک رکورد با اندازه ثابت، به طوری که ما می‌توانیم به هر رکوردی برای خواندن یا نوشتن در هر لحظه پرش کنیم.

دو نوع از فایل‌های با دسترسی تصادفی وجود دارد: فایل‌های نوع گونه (Typed) و فایل‌های بدون نوع (Untyped).

فایل‌های نوع گونه – Typed files

فایل‌های نوع گونه برای فایل‌هایی که محتوی انواع داده هم نوع که رکوردهایی هم اندازه دارند استفاده می‌شود، برای مثال، اگر یک فایل شامل رکوردهای از نوع Byte باشد، به این معنیست که اندازه رکورد=۱ بایت است. اگر فایل شامل اعداد حقیقی (Single) باشد،به این معنیست که همه رکوردها دارای اندازه ۴ بایت هستند و غیره.

در مثال بعدی، ما چگونگی استفاده فایل از Byte را نشان خواهیم داد:

برنامه نمره

```
var
  F: file of Byte;
  Mark: Byte;
begin
  AssignFile(F, 'marks.dat');
  Rewrite(F); // Create file
  Writeln('Please input students marks, write 0 to exit');

  repeat
    Write('Input a mark: ');
    Readln(Mark);
    if Mark <> 0 then  // Don't write 0 value
      Write(F, Mark);
  until Mark = 0;
  CloseFile(F);

  Write('Press enter key to close..');
  Readln;
end.
```

در این مثال، ما از این نحو را برای تعریف یک فایل نوع گونه استفاده کرده‌ایم:

```
    F: file of Byte;
```

که به معنی: فایل شامل رکوردهایی با نوع داده بایت، که ارزشی از ۰ تا ۲۵۵ را می‌تواند نگهداری کند.

و ما فایل را ایجاد کرده‌ایم و آن را برای نوشتن با استفاده از روال Rewrite باز می‌کنیم:

```
  Rewrite(F); // Create file
```

همچنین ما از تابع Write به جای Writeln برای نوشتن رکوردها در فایل نوع گونه استفاده کرده‌ایم:

```
      Write(F, Mark);
```

برای فایل‌های نوع گونه Writeln مناسب نیست، زیرا آن کارکتر CR/LF را بر روی هر خط بعد از نوشتن متن اضافه می‌کند، اما Write رکورد را همانند آن بدون هیچ ضمیمه‌ای ذخیره می‌کند. در این مورد، اگر ما ۱۰ رکورد را وارد کرده‌ایم (از Byte)، اندازه فایل بر روی دیسک ۱۰ بایت خواهد بود.

در مثال بعدی، ما تا چگونگی نمایش دادن محتویات فایل قبلی را نشان خواهیم داد:

خواندن نمره‌های دانش آموز

```
program ReadMarks;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
```

```
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };

var
  F: file of Byte;
  Mark: Byte;
begin
  AssignFile(F, 'marks.dat');
  if FileExists('marks.dat') then
  begin
    Reset(F); // Open file
    while not Eof(F) do
    begin
      Read(F, Mark);
      Writeln('Mark: ', Mark);
    end;
    CloseFile(F);
  end
  else
    Writeln('File (marks.dat) not found');

  Write('Press enter key to close..');
  Readln;
end.
```

در مثال بعدی، ما چگونگی افضودن رکورد جدید را بدون حذف اطلاعات موجود را نشان می‌دهیم:

برنامه اضافه کردن نمره‌های دانش آموز

```
program AppendMarks;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };

var
  F: file of Byte;
  Mark: Byte;
begin
  AssignFile(F, 'marks.dat');
  if FileExists('marks.dat') then
  begin
    FileMode:= 2; // Open file for read/write
    Reset(F); // open file
    Seek(F, FileSize(F)); // Go to beyond last record
    Writeln('Please input students marks, write 0 to exit');

    repeat
      Write('Input a mark: ');
      Readln(Mark);
      if Mark <> 0 then  // Don't write 0 value in disk
        Write(F, Mark);
    until Mark = 0;
    CloseFile(F);
  end
  else
    Writeln('File marks.dat not found');
```

```
  Write('Press enter key to close..');
  Readln;
end.
```

بعد از اجرای این برنامه و وارد کردن رکوردهای جدید، ما می‌توانیم آن را دوباره برای دیدن اطلاعات اضافه شده اجرا کنیم.

توجه داشته باشید که ما از Reset جهت باز کردن فایل برای نوشتن به جـای روال Rewrite اسـتفاده کرده‌ایـم. Rewrite همه داده‌ها را در فایل‌های موجود پاک می‌کند، و اگر فایل وجود نداشته باشد یک فایل خالی می‌سازد، درحـالی کـه Reset فقط می‌تواند یک فایل موجود را باز کند بدون اینکه محتویات آن را پاک کند.

همچنین ما ۲ را به متغیر FileMode اختصاص داده‌ایم برای اینکه نشان دهیم نیاز داریم فایل را برای خواندن/نوشـتن بـاز کنیم. ۰ در FileMode به معنی فقط خواندنی، ۱ به معنی فقط نوشتنی، ۲ (پیش فرض) به معنی خواندن و نوشتن است.

```
    FileMode:= 2; // Open file for read/write
    Reset(F); // open file
```

Reset اشاره‌گر خواندن/نوشتن را در اولین رکورد قرار می‌دهد، و به همین دلیل اگر ما بلافاصله شروع به نوشـتن رکوردهـا کنیم، رکوردهای قدیمی را بازنویسی خواهیم کرد، پس از زوال Seek برای حرکت اشاره‌گر خواندن/نوشتن به انتهـای فایـل استفاده می‌کنیم. Seek می‌تواند فقط با فایل‌های دسترسی تصادفی استفاده شود.

اگرما سعی کنیم تا به یک موقعیت رکورد که وجود ندارد با روال Seek دسترسی پیدا کنیم (برای مثال، رکورد شماره ۱۰۰، درحالی که ما فقط ۵۰ رکورد داریم)، یک خطا دریافت می کنیم.

ما همچنین از تابع FileSize استفاده کرده‌ایم،که تعداد رکورد فعلی را در فایل بـر می‌گردانـد.آن در ترکیـبی بـا روال Seek برای پرش به انتهای فایل استفاده شده است:

```
        Seek(F, FileSize(F)); // Go to after last record
```

توجه داشته باشید این مثال اگر فایل نمره دانش آموز در حال حاضر وجود داشته باشد می‌تواند استفاده شود؛ وگرنه، ما بایـد برنامه اول را (مرتب کردن نمره‌های دانش آموز) اجرا کنیم زیرا در آن از Rewrite استفاده شده که می‌تواند فایل‌های جدید را ایجاد کند.

ما می‌توانیم هر دو روش را (Reset و Rewrite) ترکیب کنیم با توجه به موجودیت فایل، همانطور که مـا در مثـال بعـدی انجام می دهیم:

برنامه ایجاد و اضافه کردن نمره‌های دانش آموز

```
program ReadWriteMarks;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };

var
  F: file of Byte;
  Mark: Byte;
begin
```

```
  AssignFile(F, 'marks.dat');
  if FileExists('marks.dat') then
  begin
    FileMode:= 2; // Open file for read/write
    Reset(F); // open file
    Writeln('File already exist, opened for append');
    // Display file records
    while not Eof(F) do
    begin
      Read(F, Mark);
      Writeln('Mark: ', Mark);
    end
  end
  else // File not found, create it
  begin
    Rewrite(F);
    Writeln('File does not exist,, not it is created');
  end;

  Writeln('Please input students marks, write 0 to exit');
  Writeln('File pointer position at record # ', FilePos(f));
  repeat
    Write('Input a mark: ');
    Readln(Mark);
    if Mark <> 0 then  // Don't write 0 value
      Write(F, Mark);
  until Mark = 0;
  CloseFile(F);

  Write('Press enter key to close..');
  Readln;
end.
```

توجه داشته باشید در این مثال، ما از روال Seek استفاده نکرده‌ایم، در عوض ما تمام محتوای فایل را از اول خوانده‌ایم. این عمل (خواندن تمام فایل) اشاره‌گر را به انتهای فایل انتقال می‌دهد.

در مثال بعدی، ما یک فایل از رکوردها را برای ذخیره کردن اطلاعات ماشین‌ها استفاده خواهیم کرد.

برنامه پایگاه داده ماشین‌ها

```
program CarRecords;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };

type
  TCar = record
    ModelName: string[20];
    Engine: Single;
    ModelYear: Integer;
  end;

var
  F: file of TCar;
```

```
  Car: TCar;
begin
  AssignFile(F, 'cars.dat');
  if FileExists('cars.dat') then
  begin
    FileMode:= 2; // Open file for read/write
    Reset(F); // open file
    Writeln('File already exist, opened for append');
    // Display file records
    while not Eof(F) do
    begin
      Read(F, Car);
      Writeln;
      Writeln('Car # ', FilePos(F), ' -------------------------');
      Writeln('Model : ', Car.ModelName);
      Writeln('Year  : ', Car.ModelYear);
      Writeln('Engine: ', Car.Engine);
    end
  end
  else // File not found, create it
  begin
    Rewrite(F);
    Writeln('File does not exist, created');
  end;

  Writeln('Please input car informaion, ',
        'write x in model name to exit');
  Writeln('File pointer position at record # ', FilePos(f));

  repeat
    Writeln('-------------------------');
    Write('Input car Model Name : ');
    Readln(car.ModelName);
    if Car.ModelName <> 'x' then
    begin
      Write('Input car Model Year : ');
      Readln(car.ModelYear);
      Write('Input car Engine size: ');
      Readln(car.Engine);
      Write(F, Car);
    end;
  until Car.ModelName = 'x';
  CloseFile(F);

  Write('Press enter key to close..');
  Readln;
end.
```

در مثال قبلی، ما نوع TCar را برای اطلاعات ماشین تعریف کردیم. اولین فیلد (Modelname) یک متغیر رشته ای هست، اما ما طول آن را به حداکثر ۲۰ کارکتر محدود کرده ایم:

```
  ModelName: string[20];
```

ما باید متغیرهای رشته‌ای را قبل از اینکه بخواهیم از آن‌ها در فایل‌ها استفاده کنیم تعریف کنیم، زیرا پیش‌فرض متغیرهای رشته‌ای ANSI متفاوت و نوع ذخیره سازی آن‌ها در حافظه نامحدود است، اما برای فایل‌های نوع گونه، هر نوع داده‌ای باید تعریف شده باشد.

کپی کردن فایل

همه انواع فایل‌ها، همچون فایل‌های متنی و فایل‌های باینری بر اساس واحد بایت هستند، که کوچکترین نمایش از داده در حافظه رایانه و بر روی دیسک است. هر فایلی باید محتوی یک بایت،دو بایت و ... یا بدون هیچ بایتی در تمام خود باشد. هر بایت می‌تواند یک عدد صحیح یا یک کد کارکتر از ۰ تا ۲۵۵ را نگهداری کند. ما می‌توانیم همه نوع‌های فایل را با استفاده از تعریف File of Byte یا File of Chare باز کنیم.

ما می‌توانیم هر فایلی را در دیگری با استفاده از فایل‌های File of Byte کپی کنیم، و نتیجه یک فایل جدید خواهد بود که محتوای آن برابر با فایل منبع است.

کپی کردن فایل‌ها با استفاده از file of byte

```
program FilesCopy;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };


var
  SourceName, DestName: string;
  SourceF, DestF: file of Byte;
  Block: Byte;
begin
  Writeln('Files copy');
  Write('Input source file name: ');
  Readln(SourceName);

  Write('Input destination file name: ');
  Readln(DestName);
  if FileExists(SourceName) then
  begin
    AssignFile(SourceF, SourceName);
    AssignFile(DestF, DestName);

    FileMode:= 0;   // open for read only
    Reset(SourceF); // open source file
    Rewrite(DestF); // Create destination file

    // Start copy
    Writeln('Copying..');
    while not Eof(SourceF) do
    begin
      Read(SourceF, Block); // Read Byte from source file
      Write(DestF, Block);  // Write this byte into new
                                          // destination file
    end;
    CloseFile(SourceF);
    CloseFile(DestF);

  end
  else // Source File not found
    Writeln('Source File does not exist');
```

```
  Write('Copy file is finished, press enter key to close..');
  Readln;
end.
```

بعد از اجرا کردن مثال قبل، ما باید یک فایل منبع موجود و یک فایل مقصد جدید را وارد کنیم. در لینوکس می‌توانیم نام‌های فایل را شبیه به این وارد کنیم:

```
Input source file name: /home/motaz/quran/mishari/32.mp3
Input destination file name: /home/motaz/Alsajda.mp3
```

در ویندوز می‌توانیم هرچیزی شبیه به این وارد کنیم:

```
Input source file name: c:\photos\mypphoto.jpg
Input destination file name: c:\temp\copy.jpg
```

اگر فایل منبع در همان پوشه برنامه FileCopy موجود باشد، ما می‌توانیم فقط نام فایل را وارد کنیم مانند این:

```
Input source file name: test.pas
Input destination file name: testcopy.pas
```

اگر ما از این روش برای کپی فایل‌های بزرگ استفاده کنیم، زمان خیلی زیادی در مقایسه با روال کپی سیستم عامل خواهد گرفت. این بدین معناست که سیستم عامل از یک تکنیک متفاوت برای کپی فایل‌ها استفاده می‌کند. اگر ما بخواهیم که یک فایل ۱ مگابایتی را کپی کنیم، به این معنی است که حلقه While در حدود ۱ میلیون بار تکرار خواهد شد، یعنی یک میلیون عملیات خواندن و یک میلیون عملیات نوشتن. اگر ما اعلان file of Byte را با file of Word جایگزین کنیم، به این معنی است که اینکار در حدود ۵۰۰،۰۰۰ سیکل برای خواندن و نوشتن خواهد گرفت، اما این کار فقط برای فایل‌هایی که اندازه آن‌ها زوج است، نه فرد کار خواهد کرد. اینکار موفق خواهد شد اگر یک فایل محتوی ۱،۴۲۰ بایت باشد، اما با یک فایل ۱،۴۲۳ بایتی بی نتیجه خواهد بود.

برای کپی یک فایل از هر نوعی از یک روش سریع‌تر استفاده می‌کنیم، ما باید استفاده کنیم از فایل‌های بدون نوع.

فایل‌های بدون نوع – Untyped files

فایل‌های بدون نوع فایل‌هایی با دسترسی تصادفی هستند، که رکورد با طول ثابت دارند، اما مرتبط به هر نوع داده ای نیستند. در عوض، آن‌ها داده‌ها (رکوردها) را به عنوان یک آرایه از بایت یا کارکتر ارتباط داده‌اند.

برنامه کپی فایل با استفاده از فایل‌های بدون نوع

```
program FilesCopy2;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };
```

```
var
  SourceName, DestName: string;
  SourceF, DestF: file;
  Block: array [0 .. 1023] of Byte;
  NumRead: Integer;
begin
  Writeln('Files copy');
  Write('Input source file name: ');
  Readln(SourceName);

  Write('Input destination file name: ');
  Readln(DestName);

  if FileExists(SourceName) then
  begin
    AssignFile(SourceF, SourceName);
    AssignFile(DestF, DestName);

    FileMode:= 0;   // open for read only
    Reset(SourceF, 1); // open source file
    Rewrite(DestF, 1); // Create destination file

    // Start copy
    Writeln('Copying..');
    while not Eof(SourceF) do
    begin
      // Read Byte from source file
      BlockRead(SourceF, Block, SizeOf(Block), NumRead);
      // Write this byte into new destination file
      BlockWrite(DestF, Block, NumRead);
    end;
    CloseFile(SourceF);
    CloseFile(DestF);

  end
  else // Source File not found
    Writeln('Source File does not exist');

  Write('Copy file is finished, press enter key to close..');
  Readln;
end.
```

چیزهای جدیدی در مثال قبلی هستند:

۱. تعریف نوع فایل‌ها :

```
  SourceF, DestF: file;
```

۲. خواندن/نوشتن متغیر (بافر)

```
  Block: array [0 .. 1023] of Byte;
```

ما از یک آرایه از بایت ها (۱ کیلو بایت، و می‌توان آن را ویرایش کرد) برای خواندن و کپی بلاک‌های فایل‌ها استفاده کرده‌ایم.

۳. باز کردن یک فایل بدون نوع به پارامتر اضافی نیاز دارد:

```
  Reset(SourceF, 1); // open source file
  Rewrite(DestF, 1); // Create destination file
```

پارامتر اضافی اندازه رکورد است، که کمترین عنصری که می‌تواند خواند/نوشت در یک لحظه است. از آنجایی که ما نیاز داریم تا هر نوع فایل را کپی کنیم، پس به این معنی است که آن باید یک بایت باشد، زیرا می‌تواند با هر اندازه فایلی مورد استفاده قرار گیرد.

۴. روال خواندن:

```
BlockRead(SourceF, Block, SizeOf(Block), NumRead);
```

روال BlockRead با فایل‌های بدون نوع استفاده شده است. آن یک دسته از داده‌ها را در یک لحظه می‌خواند.

پارامتر های روال BlockRead هستند :

- SourceF : این متغیر فایل منبع است که ما می‌خواهیم آن را کپی کنیم

- Block : این متغیر یا آرایه ای است که داده‌های فعلی که در حال خواندن و نوشتن است را در آن نگه می داریم.

- SizeOf(Block : این عدد مطلوب رکوردهایی است که ما نیاز داریم در یک لحظه بخوانیم. برای مثال، اگر ما عدد ۱۰۰ را وارد کنیم به این معنی است که ما می‌خواهیم ۱۰۰ رکورد (بایت در این مورد) بخوانیم. اگر ما از تابع SizeOf استفاده کنیم، به این معنی است که ما می‌خواهیم رکوردهای معمولی به تعداد نگه دارنده های آن‌ها (آرایه ای از بایت) بخوانیم.

- NumRead : ما گفته‌ایم که تابع BlockRead یک تعداد خاص از رکوردها (۱۰۲۴) می خولند، و بعضی اوقات آن در خواندن همه این میزان، و بعضی اوقات فقط بخشی از آن موفق است. برای مثال فرض کنید که ما نیاز داریم تا یک فایل شامل فقط ۱۰۰ بایت را بخوانیم، این یعنی BlockRead می‌تواند فقط ۱۰۰ بایت بخواند. همچنین در بلوک انتهایی فایل خواندن رکوردهای کمتری از درخواست اتفاق می افتد. برای مثال اگر فایل شامل ۱۰۳۴ بایت باشد، به این معنی است که در حلقه اول، ما ۱۰۲۴ بایت دریافت می‌کنیم، اما در حلقه بعدی ما فقط ۱۰ بایت باقی‌مانده را دریافت خواهیم کرد، و تابع Eof مقدار صحیح را بر خواهد گرداند. ما ارزش NumRead را برای استفاده در روال BlockWrite نیاز داریم.

۵. نوشتن در فایل‌های بدون نوع:

```
BlockWrite(DestF, Block, NumRead);
```

این روال نوشتن است و مشابه روال BlockRead می‌باشد، اما مقداری متفاوت است: ۱. به جای استفاده از روال SizeOf ما از NumRead استفاده کرده‌ایم زیرا NumRead شامل اندازه واقعی بلوک خواندن است. ۲. چهارمین پارامتر NumWritten (که در این مثال استفاده نشده است) مهم نیست زیرا ما همیشه رکوردهای نوشته شده مورد نظرمان را بدست می آوریم، مگر دیسک پر باشد.

بعد از اینکه ما این برنامه را اجرا کنیم، سرعت کپی شدن فایل‌های بزرگ را ملاحظه می‌کنید. اگر فایل شامل ۱ مگابایت باشد، این بدین معنی است که ما فقط در حدود یک هزار سیکل خواندن/نوشتن برای کپی تمام فایل نیاز داریم.

در مثال بعدی، ما از فایل‌های بدون نوع برای خواندن و نمایش هر فایلی که در حافظه یا دیسک ذخیره شده است استفاده خواهیم کرد. همانطور که می دانیم، فایل‌های لیستی از بایت‌ها هستند.

برنامه نمایش محتوای فایل

```
program ReadContents;

{$mode objfpc}{$H+}
```

```
uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };

var
  FileName: string;
  F: file;
  Block: array [0 .. 1023] of Byte;
  i, NumRead: Integer;
begin
  Write('Input source file name: ');
  Readln(FileName);

  if FileExists(FileName) then
  begin
    AssignFile(F, FileName);

    FileMode:= 0;   // open for read only
    Reset(F, 1);

    while not Eof(F) do
    begin
      BlockRead(F, Block, SizeOf(Block), NumRead);
      // display contents in screen
      for i:= 0 to NumRead - 1 do
        Writeln(Block[i], ':', Chr(Block[i]));
    end;
    CloseFile(F);

  end
  else // File does not exist
    Writeln('Source File does not exist');

  Write('press enter key to close..');
  Readln;
end.
```

بعد از اجرای این مثال و وارد کردن نام یک فایل متنی به عنوان مثال، ما برای اولین بار ارزش‌های ۱۳/۱۰ - CR/LF را خواهیم دید زیرا ما کد (اسکی)هر کارکتر را نمایش می‌دهیم. در لینوکس ما فقط تعویض خط (LF) را خواهیم یافت که مقدار ۱۰ را در ده دهی دارد. این یک حائل خط در فایل متنی است.

ما می‌توانیم انواع دیگر فایل‌ها را شبیه تصاویر، فایل‌های اجرایی، برای دیدن اینکه آن‌ها از داخل چگونه اند نمایش بدهیم.

ما از تابع Chr در این مثال برای گرفتن مقدار عددی کارکترها استفاده کرده‌ایم، برای مثال حرف a در حافظه در یک بایت با مقدار ۹۷ ذخیره شده است، و حرف A بزرگ با مقدار ۶۵.

تاریخ و زمان

تاریخ و زمان دو موضوع خیلی مهم در برنامه نویسی هستند. آن‌ها برای برنامه‌هایی که تراکنش یا هر عملیات اطلاعاتی را ذخیره می کنند مهم هستند، همانند خرید، پرداخت صورتحساب‌ها و غیره، آن‌ها نیازدارند تا تاریخ و زمان معامله را ذخیره

کنند. بعد آن‌ها می‌توانند مشخص کنند که تراکنش‌ها و عملیات برای مثال، در ماه گذشته یا ماه فعلی اتفاق افتاده‌اند.

لاگ برنامه‌های کاربردی عملی است که به ثبت تاریخ/زمان متکی است. ما بایستی بدانیم کی برخی از برنامه‌های کاربردی شروع،متوقف، ایجاد یک خطا یا مختل می شوند.

TdateTime یک نوع در پاسکال شئ‌گرا است که می‌توانیم از آن برای ذخیره اطلاعات تاریخ/زمان استفاده کنیم. این نوع یک عدد با دقت اعشار است که ۸ بایت از حافظه را آشغال می کند. بخش کوچکی از این نوع زمان را نشان می دهد، و بخش بیشتر آن عددی از روزها را که از سال گذشته است نشان می‌دهد Dec/۳۰/۱۸۹۹.

در مثال بعدی، ما چگونگی نمایش تارخ/زمان فعلی را با استفاده از تابع Now نشان می‌دهیم.

```
program DateTime;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes , SysUtils
  { you can add units after this };

begin
  Writeln('Current date and time: ', DateTimeToStr(Now));
  Write('Press enter key to close');
  Readln;
end.
```

ما از تابع DateTimeToStr که در واحد SysUtils موجود است برای تبدیل یک نوع TdateTime که به وسیله تابع Now نتیجه شده است به یک رشته نمایشی قابل خواندن تاریخ/زمان استفاده کرده‌ایم.

اگر ما از این تبدیل استفاده نمی کردیم، ما یک تاریخ/زمان مشابه یک عدد حقیقی برای نمایش دریافت می کردیم.

```
  Writeln('Current date and time: ', Now);
```

همچنین ۲ تابع تبدیل تاریخ/زمان وجود دارد که فقط بخشی از تاریخ و زمان را نمایش می‌دهد:

```
  Writeln('Current date is ', DateToStr(Now));
  Writeln('Current time is ', TimeToStr(Now));
```

تابع Date فقط بخش تاریخ امروز را نتیجه می‌دهد، و تابع Time فقط زمان فعلی را نتیجه می‌دهد:

```
  Writeln('Current date is ', DateToStr(Date));
  Writeln('Current time is ', TimeToStr(Time));
```

این دو تابع صفرهایی را در بخش دیگر قرار می دهند، برای مثال، تابع Date روز فعلی را برمی‌گرداند و صفر را در قسمت زمان قرار می دهد(صفر در زمان به معنی ۱۲:۰۰ قبل از ظهر است). تابع Time زمان فعلی سیستم را برمی‌گرداند، و صفر را در قسمت تاریخ قرار می‌دهد (صفر در تاریخ به معنی ۳۰/۱۲/۱۸۹۹ است)

ما می‌توانیم این را با استفاده از تابع DateTimeToStr چک کنیم:

```
  Writeln('Current date is ', DateTimeToStr(Date));
  Writeln('Current time is ', DateTimeToStr(Time));
```

تابع DateTimeToStr تاریخ/زمان را مطابق تنظیمات تاریخ/زمان رایانه نشان می‌دهد. نتیجه آن ممکن است بین دو سیستم رایانه‌ای متفاوت باشد، اما تابع FormatDateTime تاریخ و زمان را در قالب نوشته شده توسط برنامه‌نویس صرفنظر از تنظیمات رایانه نشان می‌دهد:

```
  Writeln('Current date is ',
     FormatDateTime('yyyy-mm-dd hh:nn:ss', Date));
  Writeln('Current time is ',
    FormatDateTime('yyyy-mm-dd hh:nn:ss', Time));
```

در مثال بعدی، ما تاریخ/زمان را به عددی شبیه به اعداد حقیقی ربط می دهیم، و جمع و تفریق مقادیر آن را انجام خواهیم داد:

```
begin
  Writeln('Current date and time is ',
    FormatDateTime('yyyy-mm-dd hh:nn:ss', Now));
  Writeln('Yesterday time is ',
    FormatDateTime('yyyy-mm-dd hh:nn:ss', Now - 1));
  Writeln('Tomorrow time is ',
    FormatDateTime('yyyy-mm-dd hh:nn:ss', Now + 1));
  Writeln('Today + 12 hours is ',
    FormatDateTime('yyyy-mm-dd hh:nn:ss', Now + 1/2));
  Writeln('Today + 6 hours is ',
    FormatDateTime('yyyy-mm-dd hh:nn:ss', Now + 1/4));
  Write('Press enter key to close');
  Readln;
end.
```

اگر ما یک را جمع یا کسر کنیم از یک تاریخ، آن جمع/تفریق با یک روز کامل خواهد شد. اگر ما جمع کنیم، برای مثال ۱/۲ یا ۰.۵ را با آن نصف یک روز (۱۲ ساعت) به آن اضافه خواهد شد.

در مثال بعدی ما یک مقدار از نوع تاریخ را با استفاده از سال،ماه و روز به‌خصوص تولید خواهیم کرد:

```
var
  ADate: TDateTime;
begin
  ADate:= EncodeDate(1975, 11, 16);
  Writeln('My date of birth is: ', FormatDateTime('yyyy-mm-dd', ADate));
  Write('Press enter key to close');
  Readln;
end.
```

تابع EncodeDate در ورودی سال،ماه و روز رادریافت می‌کند، و مقدار روز/ماه/سال را در یک متغیر از نوع TdateTime برمی‌گرداند.

تابع EncodeTime ساعت،دقیقه،ثانیه و میلی ثانیه را دریافت می‌کند و مقدار زمان را شبیه به مقدار TdateTime برمی‌گرداند:

```
var
  ATime: TDateTime;
begin
  ATime:= EncodeTime(19, 22, 50, 0);
  Writeln('Almughrib prayer time is: ', FormatDateTime('hh:nn:ss', ATime));
  Write('Press enter key to close');
```

```
  Readln;
end.
```

مقایسه تاریخ/زمان

شما می‌توانید دو متغیر تاریخ/زمان را همانند مقایسه اعداد حقیقی مقایسه کنید. برای مثال در اعداد حقیقی ۹.۳ بزرگ‌تر است از ۵.۱ به همین ترتیب برای مقادیر TdateTime اتفاق می‌افتد. اکنون ۱+، که نشان دهنده‌ی فردا است، بزرگتر است از امروز (اکنون) ، و اکنون + ۱/۲۴ که به معنی یک ساعت بعد از الان بزرگ‌تر است از اکنون – ۲/۲۴ که به معنی دو ساعت قبل از حال است.

در مثال بعدی، ما تاریخ ۱ Jan سال ۲۰۱۲ را در یک متغیر قرار خواهیم داد و آن را با تاریخ فعلی مقایسه می‌کنیم و چک می‌کنیم که این تاریخ گذشته است و یا هنوز نیامده است.

```
var
  Year2012: TDateTime;
begin
  Year2012:= EncodeDate(2012, 1, 1);

  if Now < Year2012 then
    Writeln('Year 2012 is not coming yet')
  else
    Writeln('Year 2012 is already passed');
  Write('Press enter key to close');
  Readln;
end.
```

ما می‌توانیم جمع کنیم توابع جدید را به این مثال، تا روزهای باقی‌مانده یا گذشته از آن تاریخ را نمایش دهیم:

```
var
  Year2012: TDateTime;
  Diff: Double;
begin
  Year2012:= EncodeDate(2012, 1, 1);
  Diff:= Abs(Now - Year2012);

  if Now < Year2012 then
    Writeln('Year 2012 is not coming yet, there are ',
      Format('%0.2f', [Diff]), ' days Remaining ')
  else
    Writeln('First day of January 2012 is passed by ',
      Format('%0.2f', [Diff]), ' Days');
  Write('Press enter key to close');
  Readln;
end.
```

Diff یک متغیر عدد حقیقی است که از تفاوت بین تاریخ فعلی و تاریخ ۲۰۱۲ نگهداری خواهد کرد. ما همچنین از تابع Abs استفاده کرده‌ایم، که قدرمطلق یک عدد را (عددی بدون علامت منفی) برمی‌گرداند.

برنامه ضبط اخبار

در این مثال، ما از فایل‌های متنی برای ذخیره عناوین خبرها استفاده خواهیم کرد، و در مجموع، ما همچنین تاریخ و زمان را

ذخیره خواهیم کرد.

بعد از بستن و بازکردن مجدد برنامه، عناوین خبرهای اخیر با تاریخ/زمان آن‌ها نمایش داده خواهد شد.

```
program news;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };

var
  Title: string;
  F: TextFile;
begin
  AssignFile(F, 'news.txt');
  if FileExists('news.txt') then
  begin       // Display old news
    Reset(F);
    while not Eof(F) do
    begin
      Readln(F, Title);
      Writeln(Title);
    end;
    CloseFile(F); // reading is finished from old news
    Append(F);   // open file again for appending
  end
  else
    Rewrite(F);

  Write('Input current hour news title: ');
  Readln(Title);
  Writeln(F, DateTimeToStr(Now), ', ', Title);
  CloseFile(F);

  Write('Press enter to close');
  Readln;
end.
```

ثابت ها

ثوابت شبیه به متغیرها هستند. آن‌ها نامی دارند و ارزش‌ها را می‌توانند نگهدارند، اما تفاوت آن‌ها با متغیرها در ویرایش ارزش است. ارزش یک ثابت نمی‌تواند ویرایش شود تا زمانی که برنامه درحال اجرا است، و مقدار آنها باید قبل از کامپایل برنامه مشخص باشد.

ما ثوابت را در یک روش متفاوت قبلا، بدون نامگذاری آنها، فقط به عنوان یک عدد صحیح یا رشته ای در این مثال استفاده کرده‌ایم:

```
  Writeln(5);
  Writeln('Hello');
```

مقدار ۵ هیچ‌گاه بعد از اجرای برنامه تغییر نخواهد کرد. 'Hello' همچنین یک ثابت رشته ای هست.

ما می‌توانیم ثوابت را با استفاده از کلمه کلیدی Const بعد از قسمت Uses برنامه همانند مثال زیر تعریف کنیم:

برنامه مصرف سوخت

```
program FuelConsumption;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };


const GallonPrice = 6.5;

var
  Payment: Integer;
  Consumption: Integer;
  Kilos: Single;
begin
  Write('How much did you pay for your car''s fuel: ');
  Readln(Payment);
  Write('What is the consumption of your car? (Kilos per Gallon): ');
  Readln(Consumption);

  Kilos:= (Payment / GallonPrice) * Consumption;

  Writeln('This fuel will keep your car running for : ',
    Format('%0.1f', [Kilos]), ' Kilometers');
  Write('Press enter');
  Readln;
end.
```

در مثال قبلی ، ما کیلومتری که آن اتومبیل با سوخت فعلی می‌تواند طی کند را محاسبه خواهیم کرد با توجه به این نکات:

۱. مصرف سوخت ماشین: ما از متغیر Consumption برای نگهداری کیلومتر در هر گالن برای اتومبیل فعلی استفاده خواهیم کرد

۲. مخزن: ما از متغیر Payment برای ذخیره اینکه چه مقدار پول برای سوخت فعلی پرداخت می شود استفاده خواهیم کرد

۳. ارزش گالن: ما از ثابت GallonPrice برای ذخیره ارزش هر گالن سوخت برای گشور فعلی استفاده کرده‌ایم. این مقدار نمی‌تواند به وسیله کاربر وارد شود، در عوض، باید به وسیله برنامه نویس تعریف شود.

ثوابت وقتی ارزش یکسانی چندین بار در یک برنامه استفاده می‌کنیم توصیه می‌شوند. اگر ما نیاز داریم که تغییر دهیم این مقدار را، می‌توانیم آن را یک بار در هدر کد انجام دهیم.

انواع Ordinal

نوع های Ordinal مقدارهایی عددی هستند که از علامت های حرفی به جای اعداد استفاده می‌کنند. برای مثال، اگر ما بخواهیم متغیرهای زبان (عربی/انگلیسی/فرانسه) تعریف کنیم می‌توانیم از ارزش ۱ برای عربی، ۲ برای انگلیسی، و ۳برای فرانسه استفاده کنیم. دیگر برنامه نویس‌ها ارزش‌های ۱ و ۲ و ۳ را نمی فهمند مگر اینکه آن‌ها توضیحی برای این مقادیر پیدا کنند. آن بیشتر قابل خواندن خواهد بود اگر انجام دهیم آن را با نوع های Ordinal همانند مثال زیر :

```
program OrdinalTypes;

{$mode objfpc}{$H+}


uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes
  { you can add units after this };

type
  TLanguageType = (ltArabic, ltEnglish);

var
  Lang: TLanguageType;
  AName: string;
  Selection: Byte;
begin
  Write('Please select Language: 1 (Arabic), 2 (English)');
  Readln(Selection);

  if Selection = 1 then
    Lang:= ltArabic
  else
  if selection = 2 then
    Lang:= ltEnglish
  else
    Writeln('Wrong entry');

  if Lang = ltArabic then
    Write('ما هو اسمك: ')
  else
    if Lang = ltEnglish then
      Write('What is your name: ');

  Readln(AName);

  if Lang = ltArabic then
  begin
    Writeln('مرحباً بك ', AName);
    Write('الرجاء الضغط على مفتاح إدخال لإغلاق البرنامج');
  end
  else
  if Lang = ltEnglish then
  begin
    Writeln('Hello ', AName);
    Write('Please press enter key to close');
  end;
  Readln;

end.
```

انواع عدد صحیح، کارکتر و بولیان از انواع Ordinal هستند، در صورتی که اعداد حقیقی و رشته‌ها نیستند.

مجموعه‌ها

نوع های مجموعه می‌توانند چندین خاصیت یا مشخصات را در یک متغیر نگهداری کنند. مجموعه‌ها فقط باارزش‌های ordinal قابل استفاده هستند.

برای مثال، اگر ما بخواهیم سیستم عاملی تعریف کنیم که برنامه‌ها را پشتیبانی کند، ما می‌توانیم آن را به این صورت انجام دهیم:

۱. تعریف نوع ordinal که نشان دهنده سیستم عامل است : *TApplicationEnv*

```
  TApplicationEnv = (aeLinux, aeMac, aeWindows);
```

۲. تعریف برنامه‌ای به عنوان مجموعه‌ای از TapplicationEnv : برای مثال:

```
  FireFox: set of TApplicationEnv;
```

۳. قرار دادن مقادیر سیستم عامل در مجموعه متغیرهای برنامه:

```
  FireFox:= [aeLinux, aeWindows];
```

```
program Sets;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes
  { you can add units after this };

type
  TApplicationEnv = (aeLinux, aeMac, aeWindows);

var
  FireFox: set of TApplicationEnv;
  SuperTux: set of TApplicationEnv;
  Delphi: set of TApplicationEnv;
  Lazarus: set of TApplicationEnv;
begin
  FireFox:= [aeLinux, aeWindows];
  SuperTux:= [aeLinux];
  Delphi:= [aeWindows];
  Lazarus:= [aeLinux, aeMac, aeWindows];

  if aeLinux in Lazarus then
    Writeln('There is a version for Lazarus under Linux')
  else
    Writeln('There is no version of Lazarus under linux');

  if aeLinux in SuperTux then
    Writeln('There is a version for SuperTux under Linux')
  else
```

```
    Writeln('There is no version of SuperTux under linux');

  if aeMac in SuperTux then
    Writeln('There is a version for SuperTux under Mac')
  else
    Writeln('There is no version of SuperTux under Mac');

  Readln;
end.
```

همچنین ما می‌توانیم از مجموعه قواعد برای نوع های ordinal استفاده کنیم، همانند اعداد صحیح:

```
  if Month in [1, 3, 5, 7, 8, 10, 12] then
    Writeln('This month contains 31 days');
```

یا کارکتر :

```
if Char in ['a', 'A'] then
    Writeln('This letter is A');
```

بکارگرفتن استثناها

۲ نوع از خطاها وجود دارد: خطاهای کامپایلی شبیه به استفاده یک متغیر بدون اینکه آن را در قسمت Var تعریف کنید، یا نوشتن عبارتی با قواعد اشتباه. این نوع از خطاها از کامپایل برنامه جلوگیری می‌کنند، و کامپایلر یک پیام مخصوص نشان می‌دهد و به خطی که شامل خطا است اشاره می‌کند.

دومین نوع خطاهای هنگام اجرا هستند. این نوع از خطا ها زمانی که برنامه در حال اجرا است رخ می دهند، برای مثال، تقسیم بر صفر، در خطی شبیه به این:

```
x:= y / z;
```

این از نظر قواعد درست است، اما در زمان اجرا کاربر می‌تواند ۰ تا Z را در این متغیر وارد کند، و سپس برنامه کرش خواهد کرد و پیام خطا تقسیم بر صفر را نشان خواهد داد.

همچنین تلاش برای باز کردن فایلی که وجود ندارد خطای هنگام اجرا تولید خواهد کرد (File not found) ، یا سعی برای ایجاد یک فایل در یک پوشه فقط خواندنی.

کامپایلر نمی‌تواند اینگونه خطاها را که فقط بعد از اجرای برنامه اتفاق می‌افتند بفهمد.

برای ساختن یک برنامه قابل اطمینان تا در زمان اجرا با پیام‌های خطا کرش نکند باید از استثناها استفاده کنیم.

روش‌های مختلفی از استفاده استثناها در پاسکال شیءگرا وجود دارد.

عبارت Try...except

قواعد :

```
try
  // Start of protected code
  CallProc1;
  CallProc2;
  // End of protected code

except
on e: exception do // Exception handling
begin
  Writeln('Error: ' + e.message);
end;
end;
```

مثال تقسیم بر صفر :

```
program ExceptionHandling;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, sysutils
  { you can add units after this };


var
  x, y: Integer;
  Res: Double;
begin
  try
    Write('Input x: ');
    Readln(x);
    Write('Input y: ');
    Readln(y);
    Res:= x / y;
    Writeln('x / y = ', Res);

  except
  on e: exception do
  begin
    Writeln('An error occurred: ', e.message);
  end;
  end;
  Write('Press enter key to close');
  Readln;
end.
```

دو بخش در عبارت try وجود دارد. اول اینکه بلوکی که ما می‌خواهیم از آن محافظت کنیم، که میان try ... ecxept قرار می‌گیرد. بخش دیگر بین except ... end وجود دارد.

اگر چیزی در بخش اول (try..except) اشتباه باشد، برنامه به بخش بعدی (except..end) خواهد رفت و برنامه کـــرش نخواهد کرد، اما ترجیحا پیام خطا مناسب را نمایش خواهد داد و اجرا را ادامه می‌دهد.

این خطی است که اگر مقدار y برابر صفر باشد می‌تواند خطا ایجاد کند:

```
    Res:= x / y;
```

اگر خطایی رخ ندهد، قسمت except...end اجرا نخواهد شد.

عبارت Try ...finally

قواعد :

```
try
  // Start of protected code
  CallProc1;
  CallProc2;
  // End of protected code

finally
  Writeln('This line will be printed in screen for sure');
end;
```

برنامه تقسیم با استفاده از روش try finally :

```
program ExceptionHandling;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes
  { you can add units after this };

var
  x, y: Integer;
  Res: Double;
begin
  try
    Write('Input x: ');
    Readln(x);
    Write('Input y: ');
    Readln(y);
    Res:= x / y;
    Writeln('x / y = ', Res);

  finally
    Write('Press enter key to close');
    Readln;
  end;
end.
```

این بار قسمت finally..end در همه حالات، صرف نظر از اینکه چه خطایی رخ داده است اجرا خواهد شد.

ایجاد استثناء

بعضی مواقع برای جلوگیری از بعضی خطاهای منطقی ما باید یک استثناء را تولید/ایجاد کنیم. برای مثال اگر کاربر مقدار ۱۳

را برای متغیر ماه وارد کند، ما می‌توانیم استثنایی ایجاد کنیم تا به او بگوید که او محدوده ماه‌ها را رعایت نکرده است.

مثال:

```
program RaiseExcept;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils
  { you can add units after this };


var
  x: Integer;
begin
  Write('Please input a number from 1 to 10: ');
  Readln(X);
  try

    if (x < 1) or (x > 10) then  // rais exception
      raise exception.Create('X is out of range');
    Writeln(' X * 10 = ', x * 10);

  except
  on e: exception do  // Catch my exception
  begin
    Writeln('Error: ' + e.Message);
  end;
  end;
  Write('Press enter to close');
  Readln;

end.
```

اگر کاربر مقداری خارج از بازه ۱ تا ۱۰ وارد کند، یک استثناء تولید خواهد شد (X is out of range) ، و اگر استفاده از استثناء در این بخش وجود نداشته باشد، برنامه کرش خواهد کرد.چون ما try..except را دور کد برنامه به همراه کلمه کلیدی rise نوشته ایم، برنامه کرش نخواهد کرد، اما در عوض یک پیام خطا نمایش خواهد داد.