بسم الله الرحمن الرحيم

إبدأ مع لغة أوبجكت باسكال

Object Pascal

إصدار ذي الحجة 1431 هجرية
الموافق ديسمبر 2010م

تأليف: معتز عبدالعظيم

code.sd

ذي الحجة 1431
3.12.2010

مقدمة الكتاب

بسم الله الرحمن الرحيم، والصلاة والسلام على أشرف الأنبياء والمرسلين، نبينا محمد وعلى آله وصحبه أجمعين.
أما بعد فهذا الكتاب موجه لمن يريد أن يتعلم لغة باسكال الكائنية (Object Pascal) لإستخدمها مع بيئة فري باسكال Free Pascal أو بيئة دلفي Delphi. فهو يصلُح للمبرمج الجديد الذي ليست لديه خبرة في البرمجة ولكن شرط أن يكون لديه خبرة ومعرفة في علوم الحاسوب. كذلك فهو يصلُح لمن كانت لديه خبرة في لغة برمجة أخرى ويريد تعلم هذه اللغة.
وبالنسبة للمبرمج الجديد فهذا الكتاب يساعد أيضاً على فهم وتعلم البرمجة عموماً حيث أنه يساعد المبرمج على فهم كيفية حل المشاكل البرمجية والحيل التي يقوم بها المبرمج في التطبيقات المختلفة.

لغة أوبجكت باسكال

أول ظهور للغة باسكال تدعم البرمجة الكائنية الموجهة (Object Oriented Programming) كان في عام 1983 في شركة أبل للكمبيوتر. بعد ذلك تلتها تيربو باسكال 5.5 الشهيرة التابعة لسلسة تيربو باسكال التي كانت تنتجتها شركة بورلاند، وقد قام المبرمج أندرس هجلسبيرغ بإضافة البرمجة الكائنية لخط تيربو باسكال في عام 1989.
لغة الباسكال الكائنية هي لغة تدعم البرمجة الهيكلية (Structured Programming) كما وتدعم البرمجة الكائنية(Object Oriented Programming).
لغة الباسكال هي لغة متعددة الأغراض تصلح لكافة أنواع التطبيقات والإحتياجات، فبدايةً من تعلُم البرمجة لسهولتها ووضوحها، مروراً بالألعاب ، والبرامج الحسابية، والبرامج التعليمية، وتطبيقات الإنترنت، وبرامج الإتصالات ، ولغات البرمجة، فمثلاً بيئة تطوير دلفي مطورة بالدلفي نفسها، كذلك فري باسكال ولازاراس تم تطويرهما بلغة الباسكال، وإنتهاءً بنظم التشغيل مثل الإصدارات الأولى من نظام تشغيل ماكنتوش كان يستخدم فيها لغة باسكال الكائنية و كذلك نظامي التشغيل StreamOS و Toro OS المستخدم فيه مترجم Free Pascal.

بعد النجاح التي حققته أداة التطوير تيربو باسكال التي كانت تُستخدم لإنتاج برامج تعمل في نظام الدوز، قامت شركة بورلانت بإنتاج أداة التطوير دلفي في عام 1995 لتستهدف نظام التشغيل وندوز 16 بت ثم وندوز 32 بت في إصدارة دلفي 2 ليعمل مع وندوز 95. وقد لاقت دلفي نجاح كبير حيث أنها كانت تنتج برامج أسرع بأضعاف المرات التي كانت تنتجها نظيرتها الفيجوال بيسك، وكان البرنامج الذي ينتج عن الدلفي لايحتاج لمكتبات إضافية أو مايعرف بالـ Run-time libraries حين إنزال البرامج في أجهزة أخرى لاتحتوي على دلفي.

دلفي Delphi

أداة التطوير دلفي التي أنتجتها شركة بورلاند كما سبق ذكره هي عبارة عن أداة تطوير سريعة للبرامج (Rapid Application Development Tool) ، أما اللغة المستخدمة في هذه الأداة فهي لغة باسكال الكائنية.
حدث تطوير كبير للغة باسكال الكائنية من قِبل فريق دلفي حتى أصبحت لغة منافسة لمثيلاتها.
وأصبحت لغة ذات إمكانات عالية ومكتبات غنيّة. المشكلة الرئيسية الموجودة في الدلفي هي أنها مرتبطة بنظام التشغيل وندوز، ولم يحدث تطوير لمترجمها ليعمل على أنظمة أخرى بشكل جدي.
الآن يعمل فريق الدلفي لإعادة صياغة مترجم الدلفي حتى يتمكن من إنتاج برامج وندوز 64 بت وليعمل

كذلك مع منصات أخرى غير الوندوز. و يتوقع إنتاج هذا المترجم الجديد في أواسط عام 2010.
بعد المنافسة بين المصادر المفتوحة والمصادر المغلقة، تأثرت شركة بورلاند كثيراً بهذه المنافسة، خصوصاً القسم المتخصص في إنتاج أدوات البرمجة الذي تم فصله في شركة تسمى CodeGear . بعد ذلك تم بيع هذه الشركة المتخصصة في أدوات البرمجة لشركة Embarcadero في عام 2008.
أنتجت شركة بورلاند في عام 2006 نسخ مجانية في أدوات تطوير دلفي وسي بلدر تسمى تيربو إكسبلورر (Turbo Explorer). طوال هذه الفترة كان الكود المصدري للغة وأدوات التطوير مغلقاً وخاصاً بشركة بورلاند ثم Embarcadero.
بلغ عدد المبرمجين الذين يستخدمون الدلفي أكثر من مليون ونصف مبرمج حسب إحصائية الشركة في عام 2008. وتم تصميم برامج كثيرة بإستخدامها، نذكر منها Skype, Morfik, Age of wonder وغيرها من البرامج المهمة.

فري باسكال Free Pascal

بعد توقف شركة بورلاند عن إنتاج خط تيربو باسكال الذي كان يستخدم نظام الدوز إلى عام 1993، قام فريق فري باسكال بإنتاج نسخة شبيهة بتيربو باسكال ليكون بديل حر مفتوح المصدر. لكن هذه المرة مع إضافة مهمة وهي إستهداف منصات جديدة مثل: لينكس، ماكنتوش، آرم، والآي فون، وغيرها بالإضافة إلى الوندوز 32 بت والوندوز 64 بت.
فريق فري باسكال كانت إحدى أهدافه هي التوافقية مع لغة باسكال الكائنية المستخدمة في الدلفي.
النسخة الأولى من مترجم فري باسكال صدرت في يوليو عام 2000، وآخر نسخة متوفر الآن في عام 2010 هي نسخة رقم 2.4

لازاراس Lazarus

بعد نجاح مترجم فري باسكال وتفوقه على مترجم تيربو باسكال، وإنتاج نسخة تعمل في عدد من المنصات التشغيلية، كانت الحلقة الناقصة هي أداة التطوير المتكاملة. لازاراس هي أداة التطوير المستخدمة مع فري باسكال، أو هي أداة التطوير التي تستخدم فري باسكال كمترجم. وهي عبارة عن مكتبة ضخمة للكائنات class library، وبهذه الطريقة نكود قد حولنا أداة باسكال إلى أداة معتمدة على التطوير بإستخدام المكونات أو الكائنات component driven development مماثلة للدلفي بالإضافة لكونها محرر للكود ومصمم للبرنامج. فهي بذلك تحقق كونها أداة تطوير سريعة RAD Rapid Application Development.
إلى الآن في 2010 لم تنتج النسخة الأولى بعد، لكن توجد نسخة ثابتة يمكن الإعتماد عليها في كتابة برامج. وقد تمت كتابة عدد من البرامج بواسطة هذه النسخ من لازاراس.

في هذا الكتاب سوف نستخدم لازاراس وفري باسكال في كتابة وشرح البرامج، إلا أن نفس البرامج يمكن تطبيقها بإستخدام الدلفي مع بعض التعدليات.

ميزات لغة باسكال

تتميز لغة باسكال الهدفية بسهولة تعلمها، وإمكاناتها العالية، وسرعة مترجماتها والبرامج التي تنتج عنها. لذلك فهي تعطي المبرمج فرصة إنتاج برامج ذات كفاءة وإعتمادية عاليتين في وقت وجيز، بإستخدام

بيئة تطوير متكاملة وواضحة دون الدخول في تعقيدات اللغات وأدوات التطوير الصعبة. وهذا يحقق الإنتاجية العالية.

المؤلف: معتز عبدالعظيم الطاهر

تخرجت في جامعة السودان للعلوم والتكنلوجيا عام 1999م الموافق 1420 هجرية. وقد بدأت بدراسة لغة باسكال في العام الأول في الجامعة عام 1995 كلغة ثانية بعد تعلم البيسك الذي بدأت تعلمه في المرحلة المتوسطة. ومنذ بداياتي مع لغة باسكال مازلت أستخدمها ووجدها أسهل وأفضل لغة للإستخدام خصوصاً بعد أن درست السي ++. وقد بدأت بإستخدام الدلفي في عام 1997م إلى الآن (عام 2010) حيث استخدمها بالإضافة إلى لازاراس وفري باسكال في كافة البرامج التي تلبي كافة الأغراض. وقد إستخدمتها في أكثر من 95% من الوقت الذي قضيته في البرمجة متمثلة في عشرات البرامج المختلفة التي قمت بكتابتها والتي تعمل الآن. وماتبقى يتمثل في الجافا سكريبت . والآن أعمل كمطور برامج Software Developer.

ترخيص الكتاب:

ترخيص هذا الكتاب هو ترخيص **"وقفي"** حيث أن المقصود به فائدة المسلمين وزيادة قوتهم وعلمهم. ويمكن لأي شخص طباعته ونسخه وتوزيعه بدون أي شروط.

بيئة التعليم المستخدمة مع هذا الكتاب:

سوف نستخدم في هذا الكتاب إن شاء الله بيئة لازاراس وفري باسكال كما سبق ذكره، ويمكن الحصول عليه من هذا الموقع: lazarus.freepascal.org.
أو يمكن الحصول عليه من داخل نظام لينكس عن طريق برامج إضافة التطبيقات، حيث نجده في قسم أدوات التطوير، أو عن طريق إستخدام yum install lazarus في فيدورا أو مايشابهها من توزيعات لينكس، أو بإستخدام apt-get install lazarus في توزيعة Ubuntu أو مايشابهها.

واللازاراس هو عبارة عن برنامج حر ومفتوح المصدر كما ذكرنا، ويوجد في أكثر من منصة نظام تشغيل. الكود والبرامج المكتوبة به يمكن نقلها لإعادة ترجمتها وربطها (Compilation and linking) في أي منصة يريدها المبرمج، إلا أن البرنامج الناتج (الملف الثنائي executables) لايمكن نقله، فهو مربتط فقط بالمنصة التي قام المبرمج بترجمة وربط برنامجه فيها حيث أن اللازاراس وفري باسكال ينتج عنه ملفات ثنائية تنفيذية تعمل مباشرة على نواة نظام التشغيل ومكتباتها ولاتحتاج لوسيط مثل برامج الجافا والدوت نت، لذلك فهو يتفوق على هذه اللغات بالسرعة في التنفيذ وعدم إحتياج مكتبات إضافية في الأجهزة التي سوف يتم تثبيت برامج لازاراس بها.

إستخدام البيئة النصية

الفصول الأولى من هذا الكتاب تستخدم إمكانات الإدخال والإخراج البسيطة والأساسية التي تسمى بالطرفية console أو البيئة النصية Text mode، وذلك لبساطتها وتوفرها في كل أنظمة التشغيل وسهولة فهمها.
ربما يمل الطالب منها ويتمنى أن يقوم بعمل برامج تستخدم الشاشة الرسومية بما تحتويه من أزرار ونوافذ وقوائم ومربعات نصية وغيرها، إلا أننا أحببنا أن لاينشغل الطالب بشيء ويركز على المفاهيم الأساسية التي سوف يستخدمها في البرامج النصية البسيطة وسوف يستخدمها في البرامج الرسومية المعقدة إن شاء الله. بهذه الطريقة يكون الفهم سهلاً وسريعاً لأساسيات الباسكال والبرمجة.
وفي الفصول المتقدمة سوف نستخدم إن شاء الله البيئة الرسومية ذات الواجهة المحببة للمستخدم العادي.

الأمثلة

معظم الأمثلة التي قمنا بكتباتبها سوف نجدها في الموقع http://code.sd في الصفحة التي تحتوي على هذا الكتاب. وعلى الدارس محاولة تصميم الأمثلة خطوة بخطوة، حتى تترسخ له البرمجة وإستخدام هذه الأدوات. فإذا تعذر له ذلك أو حدثت مشكلة يمكن مقارنة البرنامج الذي قام بتصميمه مع المثال الموجود في الصفحة.

المحتويات

مقدمة الكتاب.....2
لغة أوبجكت باسكال 2
دلفي *Delphi*.....2
فري باسكال *Free Pascal*.....3
لازاراس *Lazarus*.....3
ميزات لغة باسكال.....3
المؤلف: معتز عبدالعظيم الطاهر.....4
ترخيص الكتاب:.....4
بيئة التعليم المستخدمة مع هذا الكتاب:.....4
إستخدام البيئة النصية 5
الأمثلة.....5

الفصل الأول

أساسيات اللغة

البرنامج الأول.....11
تجارب أخرى:.....13
المتغيرات *Variables*.....15
الأنواع الفرعية.....19
التفرعات المشروطة *Conditional Branching*.....20
عبارة الشرط *If condition*.....20
برنامج مكيف الهواء:.....20
برنامج الأوزان.....22
عبارة الشرط *Case .. of*.....24
برنامج المطعم.....24
برنامج المطعم بإستخدام عبارة *if*.....25
برنامج درجة الطالب.....26

برنامج لوحة المفاتيح.....26
الحلقات *loops*.....28
حلقة *for*.....28
جدول الضرب بإستخدام *for loop*.....29
برنامج المضروب *Factorial*.....29
حلقة *Repeat Until*.....31
برنامج المطعم بإستخدام *Repeat Until* 31
حلقة *while do*.....33
برنامج المضروب بإستخدام حلقة *while do*.....33
المقاطع *strings*.....35
الدالة *Copy*.....38
الإجراء *Insert*.....39
الإجراء *Delete*.....39
الدالة *Trim*.....40
الدالة *StringReplace*.....40
المصفوفات *arrays*.....42
السجلات *Records*.....45
الملفات *files*.....47
الملفات النصية *text files*.....48
برنامج قراءة ملف نصي.....48
برنامج إنشاء وكتابة ملف نصي.....50
الإضافة إلى ملف نصي.....53
برنامج الإضافة إلى ملف نصي:.....53
ملفات الوصول العشوائي *Random access files*.....54
الملفات ذات النوع *typed file*.....54
برنامج كتابة درجات الطلاب.....54
برنامج قراءة ملف الدرجات.....55
برنامج إضافة درجات الطلاب.....56
برنامج إنشاء وإضافة درجات الطلاب.....57
برنامج سجل السيارات.....58
نسخ الملفات *Files copy*.....60
برنامج نسخ الملفات عن طريق البايت.....60
الملفات غير محددة النوع *untyped files*.....62
برنامج نسخ الملفات بإستخدام الملفات غير محددة النوع.....62

برنامج عرض محتويات ملف بالبايت.....64
التاريخ والوقت *Date and Time*.....66
مقارنة التواريخ والأوقات.....68
مسجل الأخبار.....69
الثوابت *constants*.....71
برنامج إستهلاك الوقود.....71
الأنواع المعدودة *Ordinal types*.....73
المجموعات *sets*.....75
معالجة الإعتراضات *Exception handling*.....77
عبارة *try except*.....77
عبارة *try finally*.....78
رفع الإستثناءات *raise an exception*.....79

الفصل الثاني

البرمجة الهيكلية Structured Programming

مقدمة.....82
الإجراءات *procedures*.....82
المدخلات *Parameters*.....83
برنامج المطعم بإستخدام الإجراءات.....83
الدوال *functions*.....85
برنامج المطعم بإستخدام الدوال وحلقة *repeat*.....86
المتغيرات المحلية *Local Variables*.....87
برنامج قاعدة بيانات الأخبار.....88
الدالة كمُدخل.....91
مخرجات الإجراءات والدوال.....92
النداء بإستخدام المرجع *calling by reference*.....92
الوحدات *units*.....94
الوحدات وبنية لازاراس وفري باسكال.....96
الوحدات التي يكتبها المبرمج.....96
وحدة التاريخ الهجري.....97
تحميل الإجراءات والدوال *Procedure and function Overloading*.....100
القيم الإفتراضية للمدخلات *default parameters*.....101
ترتيب البيانات *sorting*.....102

خوارزمية ترتيب الفُقّاعة *bubble sort*.....102
برنامج ترتيب درجات الطلاب :.....104
خوارزمية الترتيب الإختياري *Selection Sort* 106
خوارزمية الترتيب *Shell*.....107
ترتيب المقاطع.....109
برنامج ترتيب الطلاب بالأسماء.....109
مقارنة خوارزميات الترتيب.....110

الفصل الثالث

الواجهة الرسومية Graphical User Interface

مقدمة.....115
دعم اللغة العربية.....115
البرنامج ذو الواجهة الرسومية الأول.....116
البرنامج الثاني: برنامج إدخال الإسم.....121
برنامج الـ *ListBox*.....123
برنامج محرر النصور *Text Editor*.....124
برنامج الأخبار.....125
برنامج الفورم الثاني.....127
برنامج المحول الهجري.....127

الفصل الرابع

البرمجة الكائنية المنحى

Object Oriented Programming

مقدمة.....132
المثال الأول، برنامج التاريخ والوقت.....132
برنامج الأخبار بطريقة كائنية.....137
برنامج الصفوف.....143
الملف الكائني *Object Oriented File*.....148
برنامج نسخ الملفات بواسطة *TFileStream*.....148
الوراثة *Inheritance*.....149

الفصل الأول

أساسيات اللغة

البرنامج الأول

بعد تثبيت اللازاراس وتشغيله، نقوم بإنشاء برنامج جديد بإستخدام الأمر التالي من القائمة الرئيسية:

Project/New Project/Program

لنحصل على الكود التالي في الـ Source Editor:

```
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.
```

قم بحفظ البرنامج عن طريق File/Save من القائمة الرئيسية. وأحفظه في مجلد معروف بالنسبة لك، ثم سمه first.lpi

بين عبارتي begin end. قم بكتابة الأوامر التالية:

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

ليصبح البرنامج الكامل كالآتي:

```
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.
```

عبارة Writeln تقوم بكتابة نص في الشاشة. وعبارة Readln تقوم بتعليق البرنامج حتى يقوم المتسخدم بقراءة الأسطر السابقة وإنتظار المستخدم حتى يقوم بالضغط على مفتاح إدخال. بعد ذلك يتنهي البرنامج ويتم إغلاقه.

بعد ذلك نقوم **بتنفيذ** البرنامج عن طريق المفتاح F9 أو بالضغط على الزر:

لتحصل على المخرجات التالية:

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

بعد تشغيل البرنامج والضغط على مفتاح الإدخال، ينتهي البرنامج ونرجع إلى الحالة الأولى، وهي حالة كتابة الكود. إذا كنا نستخدم نظام لينكس نجد على القرص ملف بإسم first، وإذا كنا نستخدم وندوز نجد ملف بإسم first.exe وهي الملفات الثنائية التنفيذية التي يمكن توزيعها في أي أجهزة أخرى لاتحتوي بالضرورة على لازاراس أو فري باسكال. هذه البرامج تكون عبارة عن برامج تطبيقية طبيعية (Native Applications).

ملاحظة

إذا لم تظهر مخرجات البرنامج كالسابق (الشاشة السوداء) قم بتعطيل الـ Debugger كالآتي:
قم بإختيار Environment/Options/Debugger من القائمة الرئيسية في لازاراس.
في مربع Debugger type and path قم بإختيار (None)

تجارب أخرى:

في نفس البرنامج السابق قم بتغيير هذا السطر:

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

بالسطر التالي:

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

قم بتشغيل البرنامج وشاهد المخرجات التالية:

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

قم بإستبدال السطر السابق بالأسطر التالية ونفذ البرنامج في كل مرة:

الكود:

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

المخرجات:

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

الكود:

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

المخرجات:

```
5 * 2 = 10
```

الكود:

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

المخرجات:

```
7.2000000000000E+0000
```

الكود:

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

المخرجات:

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

الكود:

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

المخرجات:

```
10 * 3 = 30
```

يمكن كتابة أي قيم بأشكال مختلفة في عبارة Writeln لنرى ماهي النتائج.

المتغيرات Variables

المتغيرات هي عبارة حاويات للبيانات. فمثلاً في الطريقة الرياضية عندما نقول أن س = 5 فهذا يعني أن س هي متغير وهي في هذه اللحظة تحمل القيمة 5. كذلك يمكن إدخالها في عبارات رياضية، حيث أن قيمة س مضروبة في 2 ينتج عنها 10:

س = 5
س * 2 ينتج عنها 10

تتميز لغة باسكال بأنها تلتزم بنوع المتغير Strong Typed language، فحسب البيانات التي سوف نضعها لابد من تحديد نوع المتغير. وهذا المتغير سوف يحمل نوع واحد فقط من البيانات طوال تشغيل البرنامج، فمثلاً إذا قمنا بتعريف متغير من النوع الصحيح، فإنه يمكننا فقط إسناد ارقام صحيحة له، ولايمكننا إسناد عدد كسري مثلاً.
كذلك يجب التعريف عن المتغير قبل إستخدامه كما في المثال التالي:

```
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.
```

فعند تنفيذه نحصل على القيمة **10**.
في المثال أعلاه قمنا بكتابة الكلمة المحجوزة Var والتي تفيد بأن الأسطر القادمة عبارة عن تعريف للمتغيرات. وعبارة:

```
  x: Integer;
```

تفيد شيئين: أولهما أن إسم المتغير الذي سوف نستخدمه هو x وأن نوعه Integer وهو نوع العدد الصحيح الذي يقبل فقط أرقام صحيحة لاتحتوي على كسور ويمكن أن تكون موجبة أو سالبة.

أما عبارة

```
  x:= 5;
```

فهي تعني وضع القيمة 5 في المتغير الصحيح x

يمكن إضافة متغير آخر للبرنامج نسميه y مثلاً كما في المثال التالي:

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

نجد أن مخرجات البرنامج السابق هي:

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

في المثال التالي نقوم بإختبار نوع جديد من المتغيرات، وهو متغير يحتوي على حرف character

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

أما المثال التالي فهو لنوع الأعداد الحقيقية التي يمكن أن تحتوي على كسور:

```
var
  x: Single;
begin
  x:= 1.8;
  Writeln('My Car engine capacity is ', x, ' litters');
  Writeln('Press enter key to close');
  Readln;
end.
```

لنتمكن من كتابة برامج أكثر تفاعلاً لابد من ذكر طريقة إدخال قيمة المتغيرات من المستخدم بدلاً من كتابتها في البرنامج. والطريقة البسيطة هي إستخدام عبارة Readln التي تُمكّن المستخدم من إدخال مدخلات حسب نوع المتغيرات كما في المثال التالي:

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

في هذه الحالة أصبح تخصيص القيمة للمتغير x هو عن طريق لوحة المفاتيح.

البرنامج التالي يقوم بحساب جدول الضرب لرقم يقوم بإدخاله المستخدم:

```
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.
```

الملاحظة المهمة في المثال السابق أن أي عبارة تتم كتابتها بين علامتي تنصيص أحادية تكتب كما هي مثلاً:

```
  ' * 1 = '
```

أما مايُكتب بدون تنصيص فإن قيمته هي التي تظهر.
يمكن تجربة العبارتين التاليتين حتى تكون الفرق أوضح بإستخدام علامة التنصيص وعدم إستخدامها:

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

فنتيجة العبارة الأولي يكون:

```
5 * 3
```

أما ناتج العبارة الثانية فيكون حاصل العملية الحسابية:

```
15
```

في المثال التالي سوف نقوم بإجراء عملية حسابية ووضع الناتج في متغير ثالث ثم إظهار قيمة هذا المتغير:

```
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.
```

فبما أن العملية الحسابية هي قسمة وربما ينتج عنها عدد كسري (حقيقي) لذلك يرفض مترجم الباسكال وضع النتيجة في متغير صحيح، لذلك لابد أن يكون هذا المتغير Res عدد حقيقي.
ونوع المتغير Single يُستخدم لتعريف متغيرات كسرية ذات دقة عشرية أُحادية.

الأنواع الفرعية

توجد أنواع كثيرة للمتغيرات، فمثلاً الأعداد الصحيحة توجد منها Byte, SmallInt, Integer, LongInt, Word. وتختلف عن بعضها في مدى الأرقام وهي أصغر وأكبر عدد يمكن إسناده لها. كذلك تختلف في إحتاجها لعدد خانات الذاكرة (البايت).

النوع	الحجم بالبايت	أصغر قيمة	أكبر قيمة
Byte	1	0	255
ShortInt	1	128-	127
SmallInt	2	32768-	32767
Word	2	0	65535
Integer	4	2147483648-	2147483647
LongInt	4	2147483648-	2147483647
Cardinal	4	0	4294967295
Int64	8	9223372036854780000-	9223372036854775807

يمكن معرفة مدى هذه الأنواع وعدد خانات الذاكرة التي تحتاجها بإستخدام الدوال: Low, High, SizeOf. كما في المثال التالي:

```
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.
```

التفرعات المشروطة Conditional Branching

من أهم مايميز برامج الحاسوب أو أي جهاز إلكتروني آخر، هو إمكانية تنفيذ إجراء معين عند حدوث شرط معين. مثلاً نجد أن في بعض السيارات يتم تأمين الأبواب عند بلوغها سرعة معينة وهي غير مؤمنة. ففي هذه الحالة الشرط مركب هو وصول سرعة معينة في وجود أبواب غير مؤمنة، أما التفرع أو الإجراء فهو عملية تأمين الأبواب. أما في حالة عدم توفر الشرط وهو (بلوغ سرعة معينة مع وجود حالة الأبواب غير مؤمنة) فإن الإجراء (تأمين الأبواب) لا يتم تنفيذه.

معظم السيارات والمصانع والغسالات الآلية وماشابهها من الأجهزة تعمل بمعالج صغير أو مايسمى micro controller وهي دائرة مدمجة(IC) يمكن برمجتها بواسطة الأسمبلي أو لغة سي. كذلك فإن بعض أنواع هذه المعالجات التي تعمل في مثل هذه الأنظمة المدمجة embedded systems مثل المعالج آرم ARM يمكن برمجتها بواسطة فري باسكال، وهي توجد في الموبايلات وبعض أجهزة كمبيوترات الإنترنت أو الألعاب وبعض الأجهزة الأخرى.

عبارة الشرط If condition

عبارة if الشرطية في لغة باسكال هي عبارة سهلة وواضحة. في المثال التالي سوف نستقبل من المستخدم درجة الحرارة ثم نحكم هل نقوم بتشغيل جهاز التكيف أم لا:

برنامج مكيف الهواء:

```
var
  Temp: Single;
begin
  Write('Please enter Temperature of this room :');
  Readln(Temp);
  if Temp > 28 then
    Writeln('Please turn on air-condition')
  else
    Writeln('Please turn off air-condition');

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

نجد أن العبارات الجديدة هي: if then else وهي تعني: إذا كانت درجة الحرارة أكبر من 28 قم بكتابة السطر الأول (Please turn on air-condition) وإذا لم يتحقق الشرط (رقم أصغر أو يساوي 28) في هذه الحالة قم بكتابة السطر الثاني (Please turn off air-condition).

يمكن كتابة شروط متتالية كما في المثال التالي:

```
var
  Temp: Single;
begin
```

```
  Write('Please enter Temperature of this room :');
  Readln(Temp);
  if Temp > 28 then
    Writeln('Please turn on air-condition')
  else
  if Temp < 25 then
    Writeln('Please turn off air-condition')
  else
    Writeln('Do nothing');
```

قم بتنفيذ البرنامج أعلاه عدة مرات مُدخِلاً قيم تتراوح بين 20 و 30 وشاهد النتائج.

يمكن تعقيد البرنامج السابق ليصبح أكثر واقعية كما في المثال أدناه:

```
var
  Temp: Single;
  ACIsOn: Byte;
begin
  Write('Please enter Temperature of this room : ');
  Readln(Temp);
  Write('Is air condition on? if it is (On) write 1, if it is (Off) write 0 : ');
  Readln(ACIsOn);

  if (ACIsOn = 1) and (Temp > 28) then
    Writeln('Do no thing, we still need cooling')
  else
  if (ACIsOn = 1) and (Temp < 25) then
    Writeln('Please turn off air-condition')
  else
  if (ACIsOn = 0) and (Temp < 25) then
    Writeln('Do nothing, it is still cold')
  else
  if (ACIsOn = 0) and (Temp > 28) then
    Writeln('Please turn on air-condition')
  else
    Writeln('Please enter a valid values');

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

نجد في المثال السابق وجود عبارة **and** وهي تعني إذا تحقق الشرطين الأول والثاني قم بتفيذ العبارة التالية.

وهذا هو تفسير البرنامج السابق:

–إذا كان المكيف يعمل ودرجة الحرارة أكبر من 28 قم بكتابة: Do no thing, we still need cooling، أما إذا لم ينطبق هذا الشرط (else) فقم بالذهاب إلى الإجراء التالي:

–إذا كان المكيف يعمل ودرجة الحرارة أقل من 25 قم بكتابة: Please turn off air-condition. أما إذا لم ينطبق الشرط فقم بالإجراء التالي:

–إذا كان المكيف مغلق ودرجة الحرارة أقل من 25 قم بكتابة: Do nothing, it is still cold. أما إذا لم

ينطبق الشرط فقم بالإجراء التالي:

–إذا كان المكيف مغلق ودرجة الحرارة أكبر من 28 قم بكتابة: Please turn on air-condition.

- أما إذا لم ينطبق هذا الشرط فهوي يعني أن المستخدم قام بإدخال قيمة غير الـ 0 أو 1 في المتغير **ACIsOn**، وفي هذه الحالة نقوم بتنبيهه بإدخال قيم صحيحة: Please enter a valid values.

إذا إفترضنا أن جهاز الحاسوب أو أي جهاز آخر يمكن تشغيل الباسكال به موصل بالمكيف وأن هناك إجراء لفتح المكيف وآخر لغلقه في هذه الحال يمكننا إستبدال إجراء Writeln بإجراءات حقيقية تقوم بتنفيذ الفتح والإغلاق. في هذه الحالة يجب زيادة الشروط لتصبح أكثر تعقيداً تحسباً للمدة الزمنية التي عمل بها المكيف، كذلك يمكن الأخذ في الإعتبار أن الوقت هو في الصباح أم المساء.

برنامج الأوزان

في المثال التالي يقوم المستخدم بإدخال طوله بالأمتار ووزنه بالكيلوجرام. يقوم البرنامج بحساب الوزن المثالي بناءً على طول المستخدم. ويقارن الوزن المثالي بالوزن الحالي للمستخدم، ويطبع النتائج حسباً لفرق الوزن:

```
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 wight 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.
```

في المثال السابق نجد عدة أشياء جديدة:

النوع **Double**: وهو مشابه للنوع Single فكلاهما عدد حقيقي، أو مايسمى بالرقم ذو الفاصلة العائمة (floating point number). والنوع Double هو رقم حقيقي ذو دقة مضاعفة، وهو يحتل مساحة ضعف النوع single حيث أن الأخير يحتل مساحة 4 بايت من الذاكرة أما الـ Double فهو يحتاج إلى 8 بايت من الذاكرة لتخزينه.

الشي الثاني هو إستخدامنا للكلمة **Or**، وهي بخلاف And ومن معناها تفيد تحقيق أحد الشروط. مثلاً إذا تحقق الشرط الأول (Height < 0.4) فإن البرنامج يقوم بتنفيذ إجراء الشرط، ولو لم يتحقق سوف يتم إختبار الشرط الثاني، فإذ تحقق تم تنفيذ إجراء الشرط، وإذا لم يتحقق الشرط الثاني تم إختبار الشرط الثالث وهكذا..

الشيء الثالث هو إستخدامنا لعبارتي **begin** و **end** بعد عبارة if وذلك لأن إجراء الشرط لابد أن يكون عبارة واحدة، وفي هذه الحالة إحتجنا لأن نقوم بتنفيذ عبارتين وهما:

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

لذلك إستخدمنا begin end لتحويل العبارتين إلى كتلة واحدة أو عبارة واحدة تصلح لأن تكون إجراء الشرط:

```
  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
```

الشيء الرابع هو إستخدام الدالة **Format**. ولا يمكن إستخدام هذه الدالة إلا بعد إضافة الوحدة **SysUtils** في عبارة Uses، والوحدة SysUtils هي عبارة عن مكتبة تحتوي على الإجراء Format، وسوف نقوم بتفصيل الوحدات أو المكتبات والإجراءات في فصل البرمجة الهيكلية إن شاء الله. هذه الدالة تقوم بإظهار المتغيرات بشكل معين، مثلاً في المثال السابق تقوم بإظهار العدد الحقيقي في شكل رقم يحتوي على فاصلة عشرية ذات خانتين كما يظهر في التنفيذ. مثلاً

```
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
```

ملحوظة: حساب الوزن المثالي في البرنامج السابق ربما تكون غير دقيقة. ولمزيد من الدقة يمكن البحث عن الموضوع في الإنترنت. والمقصود بهذا البرنامج كيفية وضع حلول برمجية لمثل هذه الأمثلة.

فلابد للمبرمج أن يعرف كيف يضع حل لمسألة معينة، لأن المعرفة بالبرمجة وأدواتها لاتكفي لأن يصبح المبرمج قادر على تحليل وتصميم برامج يعتمد عليها، لذلك لابد من الدخول في تفاصيل الموضوع الذي يتم تحليله قبل وضع البرنامج المناسب.

عبارة الشرط Case .. of

توجد طريقة أخرى للتفرع المشروط وهو إستخدام عبارة **Case .. of**.
مثلاً نريد إستقبال طلب للزبون في مطعم، والوجبات مرقمة في قائمة رئيسية، ونريد من الزبون إختيار رقم الطلب كما في المثال التالي:

برنامج المطعم

```
var
  Meal: Byte;
begin

  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;
  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.
```

إذا أردنا عمل نفس البرنامج بإستخدام عبارة If فإن البرنامج سوف يكون أكثر تعقيداً ويحتوي على تكرار.

برنامج المطعم بإستخدام عبارة if

```
var
  Meal: Byte;
begin
  Writeln('Welcome to Pascal restaurant, please select your meal');
  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;
  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.
```

يمكن كذلك إستخدام مدى للأرقام بعبارة case. في المثال التالي يقوم البرنامج بتقييم درجة الطالب:

برنامج درجة الطالب

```
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.
```

نلاحظ في البرنامج السابق أننا إستخدما مدى مثل 85 .. 100 أي أن الشرط هو أن تكون الدرجة بين هاتين القيمتين.
عبارة case لاتعمل إلا مع المتغيرات المعدودة مثل الأعداد الصحيحة أو الحروف، لكنها لاتعمل مع بعض الأنواع الأخرى مثل الأعداد الحقيقية والمقاطع.

برنامج لوحة المفاتيح

في البرنامج التالي نقوم بإستخدام نوع المتغيرات الحرفية char وذلك لإستقبال حرف من لوحة المفاتيح ومعرفة موقعه:

```
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.
```

نلاحظ في المثال السابق أنا لم نستخدم المدى لكن إستخدمنا تعدد الخيارات، مثلاً إذا إخترنا العبارة الأخيرة:

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

فهي تعني إذا كانت قيمة **key** هي z أو x أو c أو v أو b أو n أو m قم بتنفيذ إجراء الشرط.

يمكن كذلك المزج بين صيغة المدى وتعدد الخيارات مثل:

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

وهي تعني إختبار الحرف إذا كان في المدى من الحرف a إلى الحرف d أو كان هو الحرف x أو y إو z قم بتنفيذ الشرط.

الحلقات loops

الحلقات هي من المواضيع المهمة والعملية في البرمجة، فهي تعني الإستمرار في تنفيذ جزء معين من العبارات بوجود شرط معين. وعندما ينتهي أو ينتفي هذا الشرط تتوقف الحلقة عن الدوران.

حلقة for

يمكن تكرار عبارة معينة بعدد معين بإستخدام for كما في المثال التالي:

```
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.
```

نستخدم في حلقات **for** متغير صحيح يسمى متغير الحلقة، وهو في هذا المثال المتغير **i**، وقيمته تبدأ في الدورة الأولى بالقيمة الإبتدائية التي حددها المبرمج، في هذه الحالة هو الرقم **1** ثم يزيد هذا المتغير في كل دورة حتى يتنهي في الدورة الأخيرة بالقيمة الأخيرة، وهي التي يحددها المستخدم بإدخالة لقيمة **Count**

يمكن إظهار قيمة متغير الحلقة كما في التعديل التالي للمثال السابق:

```
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.
```

نلاحظ هذه المرة أننا قمنا تنفيذ عبارتين، لذلك إحتجنا لأن نوحدهما في شكل عبارة واحدة بإستخدام begin end.

جدول الضرب بإستخدام for loop

لو قارنا بين برنامج جدول الضرب السابق والتالي الذي سوف نستخدم فيه عبارة for سوف نجد أن الأخير ليس فيه تكرار كالأول:

```
program MultTableWithForLoop;

{$mode objfpc}{$H+}

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

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

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.
```

نجد أننا بدلاً عن كتابة إجراء إظهار حاصل الضرب 12 مرة فقد تمت كتابته مرة واحدة فقط وتولت حلقة for تكرار هذا الإجراء 12 مرة.

يمكن جعل الحلقة تدور بالعكس، من القيمة الكبرى إلى القيمة الصغرى وذلك بإستخدام downto وذلك بتغيير سطر واحد في المثال السابق لجدول الضرب:

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

برنامج المضروب Factorial

المضروب هو مجموع حاصل ضرب الرقم مع الرقم الذي يسبقه إلى الرقم واحد:
مضروب 3 يساوي 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.
```

حلقة Repeat Until

بخلاف حلقة for loop التي تتميز بأنها تدور بعدد مين فإن Repeat loop تدور إلى أن يتحقق شرط معين، فمادام هذا الشرط غير مُحقق فهي تعيد الدوران، فإذا تحقق الشرط فسوف يخرج مؤشر التنفيذ من هذه الحلقة ويتم تنفيذ مابعدها. لذلك نجد أن هذه الحلقة غير محددة بعدد معين من الدورات، فعددها يعتمد على أن الشرط تحقق بعد كم من الدورات.

```
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.
```

في البرنامج السابق يقوم البرنامج بالدخول للحلقة أولاً ثم سؤال المستخدم أن يدخل عدداً، ثم يقوم في نهاية الحلقة بفحص قيمة هذا العدد فإذا ساوي الصفر أو عدد أقل منه فإن الشرط يكون قد تحقق وتنتهي الحلقة، أما إذا أدخلنا فيها أرقام أكبر من الصفر فإن الحلقة تستمر في الدوران.

برنامج المطعم بإستخدام Repeat Until

```
var
  Selection: Char;
  Price: Integer;
  Total: Integer;
begin
  Total:= 0;
  repeat
    Writeln('Welcome to Pascal Resturant. 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 Checken, ',
               '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.
```

في البرنامج السابق هناك عدة أشياء جديدة وهي:

1.إضافة begin end مع case وذلك لأن تنفيذ الشرط لابد أن يكون عبارة واحدة، وكما سبق ذكره فإن begin end تقوم بتحويل عدة عبارات إلى عبارة واحدة. فعبارة Writeln('You have ordred.. تمت إضافة عبارة جديدة معها وهي price:= 2 وهي لمعرفة قيمة الطلب وإضافته لاحقاً لمعرفة الحساب الكلي.

2.تم إستخدام المتغير selection من النوع الحرفي char وهو يتميز بإمكانيته في إستقبال أي رمز من لوحة المفاتيح، فمثلاً يمكن أن يستقبل رقم (خانة واحدة فقط) أو حرف. وقد إستخدمنا الحرف x للدلالة على نهاية الطلبات.

3.قمنا بوضع صفر في المتغير Total والذي سوف نقوم بتجميع قيمة الطلبات فيه ثم نعرضه في النهاية. وقد إستخدمنا هذه العبارة للتجميع:

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

4.وضعنا خيارين لإدخال الحرف x فإذا قام المستخدم بإدخال x كبيرة X أو صغيرة x فإن الحلقة

تتوقف. علماً بأن قيمة الحرف x تختلف عن قيمته بالصيغة الكبيرة X.

حلقة while do

تتشابه حلقة while مع حلقة repeat بإرتباط الإستمرار في الدوران مع شرط معين، إلا أن while تختلف إختلافين:
1.يتم فحص الشرط أولاً ثم الدخول إلى الحلقة
2.حلقة repeat يتم تنفيذها مرة واحدة على الأقل، وبعد الدورة الأولى يتم فحص الشرط، أما while فهي تمنع البرنامج من دخول الحلقة إذا كان الشرط غير صحيح من البداية.
3.إذا كانت الحلقة تحتوي على أكثر من عبارة فلابد من إستخدام begin end. أما حلقة repeat فإنها لاتحتاج، لأن الحلقة حدودها تبدأ من كلمة repeat وتنتهي بـ until.

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.
```

يمكننا إعادة كتابة برنامج المضروب بإستخدام while do

برنامج المضروب بإستخدام حلقة while do

```
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.
```

بما أن حلقة while ليس لديها متغير حلقة مثل for loop فقد قمنا بإستخدام المتغير i ليكون متغير الحلقة. لذلك قمنا بإنقاص قيمة المتغير i حتى يصلى إلى واحد وهو شرط عدم الدخول مرة أخرى في الحلقة.

المقاطع strings

المقاطع هي عبارة عن نص (text) أو مجموعة من الحروف والأرقام والرموز، حيث أنه يمكن للمتغير المقطعي أن يستقبل إسم مستخدم مثلاً أو رقم جواز أو رقم لوحة سيارة تحتوي على حروف وأرقام.

المثال التالي يوضح كيفية إستقبال وطباعة قيمة متغير مقطعي:

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

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

والمثال التالي نستخدم عدة متغيرات لتخزين معلومات شخص معين:

```
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.
```

يمكن تجميع المقاطع لتكوين مقطع أكبر، مثلاً إذا افترضنا أن لدينا ثلاث متغيرات مقطعية، أحدها يحمل إسم الشخص والثاني يحمل إسم والده والثالث إسم الجد، فيمكن تجميعها لتصبح مقطعاً واحداً:

```
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.
```

نلاحظ في المثال السابق أننا قمنا بإضافة مسافة ' ' بين كل إسم والآخر حتى لاتلتصق الأسماء وتكون غير مقروءة. وهذه المسافة أيضاً عبارة عن رمز يمثل وحدة من وحدات المقطع.

يمكن إجراء عدة عمليات على المقاطع، مثل البحث فيها عن مقطع معين أو نسخها لمتغير مقطعي آخر، أو تحويلها إلى حروف إنجليزية كبيرة أو صغيرة. وكمثال يمكننا تجربة الدوال الخاصة بالحروف الكبيرة والصغيرة في اللغة الإنجليزية:

يمكننا إضافة سطر للمثال السابق قبل طباعة الإسم كاملاً. والإضافة هي:

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

أو التحويل للحروف الصغيرة:

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

في المثال التالي نريد البحث عن الحرف a في إسم المستخدم، وذلك بإستخدام الدالة Pos والتي تقوم بإرجاع رقم الحرف في المقطع، مثلاً الحرف رقم 1 أو 2، وهكذا، وإذا لم يكن موجود فإن هذه الدالة ترجع 0 وهو يعني عدم وجود هذا الحرف في المقطع المعني:

```
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.
```

نلاحظ بعد تشغيل البرنامج السابق أن الإسم إذا إحتوى على A كبيرة فإنها لاتساوي الـ a الصغيرة ويعتبر البرنامج أنها غير موجودة، ولجعل البرنامج يقوم بتجاهل حالة الحرف يمكننا إضافة الدالة LowerCase أثناء البحث:

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

كذلك يمكن معرفة موضع الحرف في الإسم وذلك بتعديل البرنامج إلى:

```
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.
```

نلاحظ أن الإسم إذا أحتوى على أكثر من حرف a فإن الدالة Pos تقوم بإرجاع رقم أول ظهور لهذا الحرف فقط.

يمكن معرفة طول المقطع (عدد حروفه) بإستخدام الدالة length كما في المثال التالي:

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

كذلك يمكن معرفة الحرف الأول بإستخدام:

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

والحرف الثاني:

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

والحرف الأخير:

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

ويمكن كتابة كافة الحروف مفرقة ، كل حرف في سطر كالآتي:

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

الدالة Copy

يمكن نسخ جزء معين من المقطع مثلاً إذا كان المقطع يحتوى على الكلمة 'hello world' فيمكننا إستخلاص كلمة 'world' إذا عرفنا موضعها بالضبط في المقطع الأصلي وذلك كما في المثال التالي:

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

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

  Writeln(Part);

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

نلاحظ أننا إستخدمنا الدالة Copy بالصيغة التالية

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

ولشرحها من الشمال إلى اليمين:

Part:=	وهي تعني أن نضع المخرجات النهائية للدالة Copy في المتغير المقطعي part
Line	وهو المقطع المصدر الذي نريد النسخ منه
7	وهو رقم الحرف الذي نريد النسخ إبتداءً منه، وهو في هذا المثال يمثل الحرف w
5	وهو طول المقطع الذي نريد نسخه، وهي في هذه الحالة يمثل طول المقطع الجزئي world

في المثال التالي نطلب من المستخدم إدخال إسم شهر مثل February ويقوم البرنامج بإظهار إختصار الشهر في هذه الصيغة : Feb وهي تمثل الحروف الثلاث الأولى من إسم الشهر:

```
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.
```

الإجراء Insert

تقوم الدالة insert بإضافة مقطع داخل مقطع آخر، بخلاف إستخدام علامة الجمع + التي تقوم بإلصاق مقطعين في النهاية، فإن الدالة insert يمكنها إدخال مقطع أو حرف داخل مقطع آخر.

في المثال التالي نقوم بإدخال كلمة Pascal داخل عبارة hello world لتصبح Hello Pascal World

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

  Insert('Pascal ', Line, 7);

  Writeln(Line);

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

ومدخلات الإجراء هي كالتالي:

'Pascal' وهو المقطع الذي نريد إدخاله على العبارة
Line وهو المقطع الأصلي الذي نريد تغييره وإدخال مقطع آخر في وسطه
7 المكان بالأحرف الذي نريد الإضافة فيه. وهذا المكان أو الفهرس هو موضع في المقطع line

الإجراء Delete

يستخدم الإجراء delete لحذف حرف أو مقطع من مقطع آخر، مثلاً في المثال التالي نقوم بحذف حرفي الـ L في عبارة Hello World لتصبح heo World. وللقيام بهذه العملية يجب معرفة موقع الحرف الذي نريد الحذف من عنده وطول الجزء المحذوف. وهي في هذه الحالة 3 و 2 على التوالي.

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

الدالة Trim

تقوم الدالة Trim بحذف المسافات فقط في بداية ونهاية مقطع معين. مثلاً إذا كان لدينا هذا المقطع ' hello ' فإنه يصبح 'hello' بعد إستخدام هذه الدالة. لكن لايمكن إظهار المسافة في أمثلتنا الحالية إلا إذا قمنا بكتابة حرف أو رمز قبل وبعد الكلمة المطبوعة:

```
program MultTableWithForLoop;

{$mode objfpc}{$H+}

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

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

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

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

  Line:= Trim(Line);

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

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

نلاحظ أننا قمنا بإضافة الوحدة أو المكتبة SysUtils وهي التي توجد فيها هذه الدوال.

توجد دوال أخرى هي TrimLeft و TrimRight وهي تقوم بحذف المسافات من جهة واحدة يمكن تجربتها في المثال السابق.

الدالة StringReplace

تقوم الدالة StringReplace بتبديل مقاطع أو حروف معينة من مقطع مصدر، ثم وضع النتيجة في مقطع جديد، كما في المثال التالي:

```
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.
```

مدخلات الدالة هي كالآتي:

1. Line: وهو المقطع المصدر الذي نريد إصدار نسخة معدلة منه
2. ' ': وهو المقطع الذي نريد البحث عنه وإبداله، وفي هذه الحالة يمثل حرف واحد وهو المسافة
3. '-': المقطع البديل، وهو المقطع الذي سوف يتم تبديله
4. [rfReplaceAll]: وهي طريقة الإبدال، في هذه الحالة سوف يتم إبدال كلي لكل المسافات.

يمكن إستخدام مقطع واحد Line والإستغناء عن المتغير Line2 مع الحصول على نفس النتيجة كالآتي:

```
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.
```

المصفوفات arrays

المصفوفة هي عبارة عن صف أو مجموعة من متغيرات ذات نوع واحد، مثلاً إذا قلنا أن لدينا مصفوفة من الأعداد الصحيحة تحتوي على 10 عناصر فإن تعريفها يكون كالتالي

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

ويمكننا وضع قيمة في المتغير الأول في المصفوفة كالتالي:

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

ويمكننا وضع قيمة في المتغير الثاني في المصفوفة كالتالي:

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

في المثال التالي نقوم بتخزين درجات 10 طلاب ثم عرضها:

```
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.
```

نجد أننا إستخدمنا حلقة for لإدخال وطباعة الدرجات. كذلك قمنا بمقارنة نتيجة كل طالب داخل الحلقة لنعرف الناجح وغير الناجح.

يمكن تعديل البرنامج السابق لنعرف أكبر وأصغر درجة في الدرجات كالتالي:

```
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
    // Get 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.
```

نلاحظ أننا إفترضنا أن الرقم الأول هو أكبر درجة لذلك قمنا بوضعه في المتغير Max كذلك إعتبرنا أنه أصغر درجة فوضعناه في المتغير Min إلى أن يثبت العكس في كلا الحالتين.

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

وفي داخل الحلقة عند طباعة الأرقام العشرة، قمنا بمقارنة كل رقم مع القيم max و min فإذا وجدنا رقم أكبر من max قمنا بإستبدال قيمة max بقيمة الدرجة الحالية ، ونفعل نفس الشيء مع القيمة min.

نلاحظ في المثال السابق أننا إستخدمنا تعليق مثل:

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

وقد بدأنا السطر بالعلامة // وهي تعني أن باقي السطر عبارة عن تعليق لاتتم ترجمته، إنما يستفيد منه المبرمج كشرح حتى يصبح البرنامج مقروء له ولمن يريد الإطلاع على البرنامج.
هذه الطريقة تصلح للتعليق القصير، أما إذا كان التعليق أكثر من سطر يمكس إستخدام الأقواس المعكوفة {} أو الأقواس والنجمة (* *)

مثلاً:

```
for i:= 1 to 10 do
begin
  { Get 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;
```

كذلك يمكن الإستفادة من هذه الخاصية بتعطيل جزء من الكود مؤقتاً كالتالي:

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

في هذا المثال قمنا بتعطيل إجراء طباعة أصغر درجة.

السجلات Records

كما لاحظنا أن المصفوفات تحتوي على مجموعة متغيرات من نوع واحد، فإن السجلات تجمع بين مجموعة من أنواع مختلفة تسمى حقول Fields، ولكنها تمثل كيان واحد. مثلاً إذا افترضنا أننا نريد تسجيل معلومات سيارة، فنجد أن هذه المعلومات هي:

1.نوع السيارة: متغير مقطعي
2.سعة المحرك: حدد حقيقي (كسري)
3.سنة التصنيع: متغير صحيح

فلايمكن التعبير عن هذه الأنواع المختلفة كوحدة واحدة إلا بإستخدام السجل كما في المثال التالي:

```
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.
```

في المثال السابق نجد أننا قمنا بتعريف نوع جديد بإستخدام الكلمة المفتاحية type :

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

وسمينا هذا النوع الجديد TCar والحرف T هو من كلمة Type حتى نفرق بينه وبين المتغيرات.
وهذا النوع الجديد الذي يمثل سجل Record يحتوي على ثلاث أنواع كما يظهر في المثال.

وعندما نريد استخدام هذا النوع الجديد لابد من تعريف متغير يمثل هذا النوع، حيث لايمكننا استخدام النوع TCar مباشرةً كما لايمكننا إستخدام النوع Integer مباشرة إلا بعد تعريف متغير مثلاً I أو Num.
لذلك قمنا بتعريف المتغير Car من النوع TCar في بند المتغيرات:

```
var
  Car: TCar;
```

وعندما نريد إدخال قيم للمتغيرات أو طباعتها نستخدم هذه الطريقة للوصول لمتغير ما في سجل:

```
  Car.ModelName
```

في درس الملفات ذات الوصول العشوائي إن شاء الله سوف نستفيد فائدة مباشرة من السجلات التي تمثل ركن أساسي في قواعد البيانات.

الملفات files

الملفات هي من أهم العناصر في نظام التشغيل والبرامج عموماً، فإن بعض إجزاء نظام التشغيل نفسها هي عبارة عن ملفات. والمعلومات والبيانات هي عبارة عن ملفات، مثل الصور والكتب والبرامج والنصوص البسيطة هي عبارة عن ملفات. ويجب على نظام التشغيل توفير إمكانية لإنشاء الملفات، ولقراءتها وكتابتها وتحريرها وحذفها.

تنقسم الملفات إلى عدة أنواع بناءً على عدة أوجه نظر. فيمكن تقسيم الملفات إلى ملفات تنفيذية وملفات بيانات، حيث أن الملفات التنفيذية هي التي تمثل البرامج وأجزائه التنفيذية مثل الملف الثنائي الذي يحتوي على كود يفهمه نظام التشغيل ومثال لها الملفات التنفيذية التي تصدر عن مترجم الفري باسكال أو الـ gcc والتي يمكن نقلها في عدد من الأجهزة التي تحتوي على نفس نظام التشغيل ثم تشغيلها بالنقر عليها أو بكتابة إسمها في شاشة الطرفية console. وكمثال لها برنامج الآلة الحاسبة، محرر النصوص، برامج الألعاب، إلخ.
أما النوع الثاني فهي ملفات البيانات التي لاتحتوى على كود ولاتمتلك إمكانية التشغيل، إنما تحتوي على بيانات بسيطة أو معقدة يمكن قراءتها مثل النصوص والملفات المصدرية للغات البرمجة مثل first.lpr ، أو الصور وملفات الصوت التي هي عبارة عن بيانات يمكن عرضها بإستخدام برامج معينة مثل برامج تحرير الصور وبرامج تعدد الوسائط multimedia. وكمثال لهذه الملفات ملفات الصوت .mp3 وملفات الكتب .pdf

يمكننا كذلك تقسيم الملفات من حيث نوع **البيانات** إلى قسمين:
1.ملفات نصية بسيطة يمكن إنشائها وقراءتها عن طريق أدوات بسيطة في نظام التشغيل مثل cat التي تعرض محتويات ملف نصي في نظام لينكس، و type التي تطبع محتويات ملف نصي في وندوز، و vi التي تقوم بتحرير وإنشاء الملفات النصية في نظام لينكس.

2.ملفات بيانات ثنائية وهي يمكن أن تحتوى على رموز غير مقروءة وهذه الملفات لاتصلح لأن يقوم المستخدم بتحريرها أو قراءتها مباشرة، بل يجب عليه إستخدام برامج محددة لهذه العمليات. مثل ملفات الصور إذا حاول المستخدم قراءتها بإستخدام cat مثلاً فإن يحصل على رموز لاتُفهم، لذلك يجب فتحها بإستخدام برامج عرض أو تحرير مثل المتصفح أو برنامج Gimp. كذلك يمكن أن يكون مثال لهذه الأنواع ملفات قواعد البيانات البسيطة مثل dBase, paradox فهي عبارة عن ملفات ثنائية تُخزن فيها سجلات تحتوى على معلومات منظمة بطريقة معينة.

النوع الآخر من التقسيم للملفات هو **طريقة الوصول** والكتابة، حيث يوجد نوعين من الملفات:

1.ملفات ذات وصول تسلسلي sequential access files: ومثال لها الملفات النصية، وتتميز بأن طول السجل أو السطر فيها غير ثابت، لذلك لامكن معرفة موقع سطر معين في الملف. ويجب فتح الملف للقراءة فقط أو الكتابة فقط، ولا يمكن الجمع بين الوضعين (الكتابة والقراءة) ولايمكن تعديلها بسهولة إلا بقراءتها كاملة في مخزن مؤقت ثم تعديل أسطر معينة فيها ثم مسح الملف وكتابته من جديد. كذلك فإن القراءة والكتابة تتم بتسلسل وهو من بداية الملف إلى نهايته، حيث لايمكن الرجوع سطر إلى الوراء.

2.ملفات ذات وصول عشوائي random access files: وهي ملفات ذات طول سجل ثابت، حين أن السجل يمثل أصغر وحدة يتعامل معها البرنامج في القراءة، الكتابة والتعديل. ويمكن الجمع بين الكتابة والقراءة في نفس اللحظة، مثلاً يمكن قراءة السجل الخامس، ثم نسخ محتوياته في السجل الأخير. ويمكن الوصول مباشرة إلى أي سجل دون التقيد بمكان القراءة أو الكتابة الحالي. فمثلاً إلى كان طول السجل هو 5 حروف أو بايت، فإن السجل الخامس يبدأ في الموقع رقم 50 في هذا الملف.

الملفات النصية text files

الملفات النصية كما سبق ذكره هي ملفات بسيطة يمكن قراءتها بسهولة. ومن ناحية الوصول هي ملفات تسلسلية sequential files حيث يجب القراءة فقط في إتجاه واحد وهو من البداية للنهاية ويجب الكتابة فيها بنفس الطريقة.
لعمل برنامج بسيط يقوم بقراءة محتويات ملف، علينا أولاً إنشاء ملفات نصية أو البحث عنها ثم وضعها في الدليل أو المجلد الذي يوجد فيه البرنامج ثم تنفيذ البرنامج التالي:

برنامج قراءة ملف نصي

```
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.
```

بعد تنفيذ البرنامج السابق نقوم بإدخال إسم ملف موجود في نفس مكان البرنامج أو بكتابة الإسم كاملاً مثل:

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

أو

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

بإستخدام أسماء الملفين الأخيرين، فإن هذا البرنامج قادر على قراءة الذاكرة الكلية للحاسوب والذاكرة المستخدمة والذاكرة المتاحة، كذلك يمكنه قراءة نوع المعالج المستخدم في جهاز الحاسوب وسرعته.

نلاحظ أننا استخدمنا دوال وإجراءات وأنواع جديدة في هذه البرنامج وهي:

.1

```
F: TextFile;
```

وهي تعريف المتغير F بأن نوعه TextFile وهو النوع الذي يتعامل مع الملفات النصية. وفي البرنامج سوف نستخدم هذا المتغيير للتعبير عن الملف الفعلي.

.2

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

وهي دالة موجودة في المكتبة SysUtils وهي تختبر وجود هذا الإسم كملف، فإذا كان موجود تحقق الشرط وإن لم يكن موجود فإن عبارة else هي التي سوف تُنفّذ.

.3

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

بعد التأكد من أن الملف موجود يقوم إجراء AssignFile بربط إسم الملف الفعلي بالمتغير F، والذي عن طريقه يمكننا التعامل مع الملف من داخل باسكال.

.4

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

وهي العبارة التي تقوم بفتح الملف النصي للقراءة فقط، وهي تخبر نظام التشغيل أن هذا الملف محجوز للقراءة، فإذا حاول برنامج آخر فتح الملف للكتابة أو حذفه فإن نظام التشغيل يمنعه برسالة مفادها أن الملف مفتوح بواسطة تطبيق آخر أو أن السماحية غير متوفرة access denied

.5

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

وهو إجراء القراءة من الملف، حيث أن هذا الإجراء يقوم بقراءة سطر واحد فقط من الملف المفتوح F ووضع هذا السطر في المتغير Line.

.6

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

كما ذكرنا في الإجراء Readln فإنه يقوم بقراءة سطر واحد فقط، وبالنسبة للملف النصي فإننا لايمكننا معرفة كم سطر يحتوي هذا الملف قبل الإنتهاء من قراءته، لذلك إستخدمنا هذه الدالة eof والتي تعني نهاية الملف End Of File. إستخدمناها مع الحلقة while وذلك للإستمرار في القراءة حتى الوصول إلى نهاية الملف وتحقيق شرط end of file.

.7

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

بعد الفراغ من قراءة الملف، يجب إغلاقه وذلك لتحريره من جهة نظام التشغيل حتى يمكن لبرامج أخرى التعامل معه بحرية، ولابد من تنفيذ هذا الإجراء فقط بعد فتح الملف بنجاح بواسطة reset مثلاً. فإذا فشل فتح الملف أصلاً بواسطة reset فلايجب إغلاقه بواسطة CloseFile.

في المثال التالي سوف نقوم بإنشاء ملف نصي جديد والكتابة فيه:

برنامج إنشاء وكتابة ملف نصي

```
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.
```

في البرنامج السابق استخدمنا عدة أشياء وهي:

.1

```
  ReadyToCreate: Boolean;
```

النوع boolean يمكن لمتغيراته أن تحمل إحدى قيمتين فقط: True/False . وهذه القيم يمكن استخدامها مع عبارة if condition كما في المثال، كذلك يمكن إستخدامها مع حلقة while وحلقة repeat. حيث أن الشرط في النهاية يتحول إلى إحدى هتين القيمتين، فكما في أمثلة سابقة إستخدمنا هذه العبارة:

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

فهي إما أن تكون النتيجة فيتحول الشرط إلى True أو تكون خاطئة فيتحول الشرط إلى False. فعندما تتحول إلى True يتم تنفيذ الشرط:

```
  if True then
```

وإذا كانت قيمتها النهائية False لاتم تنفيذ إجراء الشرط وإنما يتم تنفيذ إجراء else إن وجد.

.2

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

هذه العبارة يتم تنفيذها في حالة وجود الملف، فيقوم البرنامج بتنبيه المستخدم بوجود الملف، وسؤاله إذا كان يرغب في حذف محتوياته والكتابة عليه من جديد (overwrite). فإذا قام بإدخال الحرف y صغيرة أو Y كبيرة فإن الشرط يتم تنفيذه في الحالتين، حيث أن الدالة upCase تقوم بتحويل الحرف إلى حرف كبير لمقارنته مع الحرف الكبير 'Y'. أما إذا ادخل المستخدم حرفاً كبيراً Y فإن الدالة upCase لاتقوم بأي تغيير وترجع الحرف كما هو Y.

.3

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

الإجراء Rewrite يقوم بإنشاء ملف جديد أو حذف محتويات الملف إذا كان موجود. كذلك فهو يفتح الملف للكتابة فقط في حالة الملف النصي.

.4

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

الإجراء Writeln(F يقوم بكتابة المقطع Line في الملف ثم إضافة علامة نهاية السطر وهي CR/LF. وهي عبارة عن رموز الواحد منها يمثل بايت وقيمتها هي كالآتي :

CR: Carriage Return = 13
LF: Line Feed = 10

وهذه الرموز من الأحرف الغير مرئية، حيث لاتتم كتابتها في الشاشة، إنما يظهر فقط مفعولها، وهي الإنتقال إلى سطر جديد.

.5

```
Inc(i);
```

يقوم الإجراء Inc بإضافة واحد إلى قيمة المتغير الصحيح، في هذه الحالة i وهو يعادل هذه العبارة:

```
i:= i + 1;
```

.6

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

كما ذكرنا سابقاً فإن الإجراء CloseFile يقوم بإغلاق الملف وإيقاف عملية الكتابة أو القراءة من الملف، وتحريره من الحجز بواسطة نظام التشغيل. إلا أن له وظيفة إضافية في حالة الكتابة. فكما نعلم أن الكتابة على القرص الصلب هي عملية بطيئة نسبياً مقارنة بالكتابة في الذاكرة، مثل إعطاء قيم للمتغيرات، أو الكتابة في مصفوفة. لذلك من غير المنطقي كتابة كل سطر على حده في القرص الصلب أو أي وسيط تخزين آخر، لذلك يقوم البرنامج بتخزين عدد معين من السطور في الذاكرة بعملية تسمى الـ Buffering ، فكلما إستخدمنا عبارة Write أو Writeln لكتابة سطر فإن البرنامج يقوم تلقائياً بكتابته في الذاكرة إلى أن يمتليء هذا الوعاء (Buffer) في الذاكرة فيقوم البرنامج تلقائياً بالكتابة الفعلية على القرص الصلب ثم حذف المحتويات من الذاكرة (Buffer)، وهذه العلمية تسمى Flushing وبهذه الطريقة نضمن السرعة في كتابة الملف، بإعتبار أن التكلفة الزمنية مثلاً لكتابة سطر واحد ربما تساوي تقريباً تكلفة كتابة 10 أسطر في القرص الصلب دفعة واحدة. وتكمن خطورة هذه الطريقة في إنقطاع الطاقة عن الحاسوب قبل الكتابة الفعلية، فيجد المستخدم أن الأسطر الأخيرة التي قام بكتابتها قد ضاعت. ويمكن إجبار البرنامج بالقيام بالكتابة الفعلية على القرص بإستخدام الإجراء Flush، كذلك فإن عملية الـ Flushing تحدث إيضاً عند إغلاق الملف بإستخدام CloseFile.

الإضافة إلى ملف نصي

سوف نقوم في هذا المثال بفتح ملف نصي يحتوي على بيانات ثم إضافة أسطر عليه بدون حذف الأسطر القديمة، ويتم ذلك بإستخدام الإجراء AppendFile

برنامج الإضافة إلى ملف نصي:

```
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 تنفيذ البرنامج يمكننا كتابة إسم ملف نصي موجود وإضافة بضعة أسطر عليه، بعد ذلك يمكننا أن نستعرض الملف عن طريق الأمر cat في لينكس، أو عن طريق تصفح الدليل الذي يوجد فيه البرنامج والضغط عليه بالماوس لفتحه.

ملفات الوصول العشوائي Random access files

كما سبق ذكره فإن النوع الثاني من الملفات من حيث الوصول هي ملفات الوصول العشوائي أو كما تسمى أحياناً بملفات الوصول المباشر. وهي تتميز بطول معروف للسجل ويمكن الكتابة والقراءة من الملف في آن واحد، كذلك يمكن الوصول مباشرة إلى أي سجل بغض النظر عن موقع القراءة أو الكتابة الحالي.
توجد طريقتين للتعامل مع الملفات ذات الوصول العشوائي في لغة باسكال: الطريقة الأولى هي إستخدام الملفات ذات النوع typed files، أما الطريقة الثانية فهي إستخدام الملفات الغير محددة النوع untyped files.

الملفات ذات النوع typed file

في هذه الطريقة يكون الملف المراد قراءته أو كتابته مرتبط بنوع معين، والنوع المعين يمثله سجل، فمثلاً يمكن أن يكون ملف من النوع الصحيح Byte، في هذه الحال نقول أن السجل هو عبارة عن عدد صحيح Byte وفي هذه الحالة يكون طول السجل 1 بايت.

المثال التالي يوضح طريقة كتابة ملف لأعداد صحيحة:

برنامج كتابة درجات الطلاب

```
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.
```

نلاحظ في البرنامج أننا قمنا بتعريف نوع الملف بهذه الطريقة:

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

وهي تعني أن الملف هو من نوع Byte أو أن سجلاته عبارة هي قيم للنوع Byte والذي يحتل في الذاكرة وفي القرص 1 بايت، ويمكنه تخزين القيم من 0 إلى 255.

كذلك قمنا بإنشاء الملف وتجهيزه للكتابة بإستخدام الأمر:

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

وقد قمنا بإستخدام الإجراء Write للكتابة في الملف:

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

حيث أن Writeln لاتصلح لهذا النوع من الملفات لأنها تقوم بإضافة علامة نهاية السطر CR/LF كما سبق ذكره، أما Write فهي تكتب السجل كما هو بدون أي زيادات، لذلك نجد أننا في المثال السابق يمكننا معرفة حجم الملف، فإذا قمنا بإدخال 3 درجات فإن عدد السجلات يكون 3 وبما أن طول السجل هو 1 بايت فإن حجم الملف يكون 3 بايت.
يمكن تغيير النوع إلى Integer الذي يحتل 4 خانات، ورؤية حجم الملف الناتج.

في المثال التالي نقوم بقراءة محتويات الملف السابق، فلاننسى إرجاع نوع الملف في البرنامج السابق إلى النوع Byte، لأن القراءة والكتابة لابد أن تكون لملف من نفس النوع.

برنامج قراءة ملف الدرجات

```
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
    Writeln('Please input students marks, write 0 to exit');

    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.
```

نلاحظ أن البرنامج السابق قام بإظهار الدرجات الموجودة في الملف. وأظن أن البرنامج واضح ولايحتاج لشرح.

في البرنامج التالي نقوم بفتح الملف السابق ونضيف إليه درجات جديدة بدون حذف الدرجات السابقة:

برنامج إضافة درجات الطلاب

```
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 after 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
        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 تنفيذ البرنامج السابق نقوم بتشغيل برنامج قراءة ملف الدرجات وذلك لرؤية أن قيم السجلات السابقة والجديدة موجودة معاً في نفس الملف.
نلاحظ في هذا البرنامج أننا إستخدمنا الإجراء Reset للكتابة بدلاً من Rewrite. وفي هذا النوع من الملفات يمكن إستخدام كليهما للكتابة، والفرق الرئيسي بينهما يكمن في أن Rewrite تقوم بإنشاء الملف إذا لم يكن موجود وحذف محتوياته إذا كان موجود، أما Reset فهي تفترض وجود الملف، فإذا لم يكون موجود حدث خطأ. لكن عبارة Reset تقوم بفتح الملف حسب قيمة FileMode :
فإذا كانت قيمتها 0 فإن الملف يفتح للقراءة فقط، وإذا كانت قيمته 1 يفتح للكتابة فقط، وإذا كانت قيمته 2 -وهي القيمة الإفتراضية- فإنه يفتح للقراءة والكتابة معاً:

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

كذلك فإن الدالة Reset تقوم بوضع مؤشر القراءة والكتابة في أول سجل، لذلك إذا باشرنا الكتابة فإن البرنامج السابق يقوم بالكتابة فوق محتويات السجلات السابقة، لذلك وجب تحريك هذا المؤشر للسجل الأخير، وذلك بإستخدام الإجراء Seek الذي يقوم بتحريك الموشر، وبهذه الطريقة تمت تسمية هذا النوع من الملفات بالملفات العشوائية أو ملفات الوصول المباشر، حيث أن الإجراء Seek يسمح لنا بالتنقل لأي سجل مباشرة إذا علمنا رقمه، شريطة أن يكون هذا الرقم موجود، فسوف يحدث خطأ مثلاً إذا قمنا بمحاولة توجيه المؤشر إلى السجل رقم 100 في حين أن الملف يحتوي على عدد سجلات أقل من 100.

استخدمنا الدالة FileSize والتي تقوم بإرجاع عدد السجلات:

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

نلاحظ أن المثال السابق يصلح فقط في حالة وجود ملف الدرجات، أما إذا لم يكن موجود وجب إستخدام البرنامج الأول لكتابة الدرجات.

يمكننا المزج بين الطريقتين، بحيث أن البرنامج يقوم بفحص وجود الملف، فإذا كان موجود يقوم بفتحه للإضافة عن طريق Reset وإذا لم يكن موجود يقوم بإنشائه بإستخدام Rewrite:

برنامج إنشاء وإضافة درجات الطلاب

```
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, 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.
```

بعد تشغيل البرنامج نجد أن القيم السابقة تم عرضها في البداية قبل الشروع في إضافة درجات جديدة. نلاحظ أننا في هذه الحالة لم نستخدم الإجراء Seek وذلك لأننا قمنا بقراءة كل محتويات الملف، ومن المعروف أن القراءة تقوم بتحريك مؤشر الملف إلى الأمام، لذلك بعد الفراغ من قراءة كافة سجلاته يكون المؤشر في الخانة الأخيرة في الملف، لذلك يمكن الإضافة مباشرة.
استخدمنا الدالة FilePos التي تقوم بإرجاع الموقع الحالي لمؤشر الملفات.

في المثال التالي سوف نستخدم سجل Record لتسجيل بيانات سيارة، نلاحظ أننا نقوم بكتابة وقراءة السجل كوحدة واحدة:

برنامج سجل السيارات

```
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.
```

في البرنامج السابق إستخدمنا سجل لمعلومات السيارة، والحقل الأول في السجل ModelName هو من النوع المقطعي، إلا أننا في هذه الحالة قمنا بتحديد طول المقطع بالعدد 20 وهو يمثل 20 حرف:

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

وبهذه الطريقة يكون طول المقطع معروف ومحدد ويأخذ مساحة معروفة من القرص، أما طريقة إستخدام النوع string مطلقاً والذي يسمى AnsiString فهي طريقة لها تبعاتها في طريقة تخزينها في الذاكرة، وسوف نتكلم عنها في كتاب لاحق إن شاء الله.

نسخ الملفات Files copy

الملفات بكافة أنواعها سواءً كانت ملفات نصية أو ثنائية، صور ، برامج، أو غيرها فإنها تشترك في أن الوحدة الأساسية فيها هي البايت Byte، حيث أن أي ملف هو عبارة عن مجموعة من البايتات، تختلف في محتوياتها، لكن البايت يحتوي على أرقام من القيمة صفر إلى القيمة 255، لذلك فإذا قرأنا أي ملف فنجد أن رموزه لاتخرج عن هذه الإحتمالات (0 - 255).
عملية نسخ الملف هي عملية بسيطة، فنحن نقوم بنسخ الملف حرفاً حرفاً بإستخدام متغير يحتل بايت واحد من الذاكرة مثل البايت Byte أو الرمز Char. في هذه الحالة لايهم نوع الملف، لأن النسخ بهذه الطريقة يكون الملف المنسوخ صورة طبق الأصل من الملف الأصلي:

برنامج نسخ الملفات عن طريق البايت

```
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.
```

عند تشغيل هذا البرنامج يجب كتابة إسم الملف المراد النسخ منه والملف الجديد كاملاً مثلا في نظام لينكس نكتب:

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

وفي نظام وندوز:

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

أما إذا كان البرنامج FileCopy موجود في نفس الدليل للملف المصدر والنسخة، فيمكن كتابة إسمي الملف بدون كتابة إسم الدليل مثلاً:

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

نلاحظ أن برنامج نسخ الملفات يأخذ وقت طويل عند نسخ الملفات الكبيرة مقارنة بنسخها بواسطة نظام التشغيل نفسه، وذلك يعني أن نظام التشغيل يستخدم طريقة مختلفة لنسخ الملفات. فهذه الطريقة بطيئة جداً بسبب قراءة حرف واحد في الدورة الواحدة ثم نسخة في الملف الجديد، فلو كان حجم الملف مليون بايت فإن الحلقة تدور مليون مرة، تتم فيها القراءة مليون مرة والكتابة مليون مرة. وكانت هذه الطريقة للشرح فقط، أما الطريقة المثلى لنسخ الملفات فهي عن طريق استخدام الملفات غير محددة النوع untyped files.

الملفات غير محددة النوع untyped files

وهي ملفات ذات وصول عشوائي، إلا أنها تختلف عن الملفات محددة النوع في أنها لاترتبط بنوع محدد، كذلك فإن القراءة والكتابة غير مرتبطة بعدد سجلات معين، حيث أن للمبرمج كامل الحرية في تحديد عدد السجلات التي يرغب في كتابتها أو قراءتها في كل مرة.

برنامج نسخ الملفات بإستخدام الملفات غير محددة النوع

```
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.
```

في المثال السابق نجد أن هناك أشياء جديدة وهي:

1.طريقة تعريف الملفات، وهي بتعريفها أن المتغير مجرد ملف:

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

2.المتغير الذي يستخدم في نسخ البيانات بين الملفين:

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

نجد أنه في هذه الحالة عبارة مصفوفة من نوع البايت تحتوي على كيلو بايت، ويمكن تغييرها إلى أي رقم يريده المبرمج.

3.طريقة فتح الملف إختلفت قليلاً:

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

فقد زاد مُدخل جديد وهو طول السجل، وفي حالة نسخ ملفات هذه يجب أن يكون دائماً يحمل القيمة واحد وهو يعني أن طول السجل واحد بايت. والسبب يكمن في أن الرقم واحد يقبل القسمة على جميع قيم حجم الملفات، مثلاً يمكن أن يكون حجم الملف 125 بايت، أو 23490 بايت وهكذا.

4.طريقة القراءة:

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

يستخدم الإجراء BlockRead مع الملفات غير محددة النوع، حيث يعتبر أن القيمة المراد قراءتها هي عبارة عن كومة أو رزمة غير معلومة المحتويات. والمدخلات لهذا الإجراء هي:

SourceF: وهو متغير الملف المراد القراءة منه.

Block : وهو المتغير أو المصفوفة التي يراد وضع محتويات القراءة الحالية فيها.

SizeOf(Block): وهو عدد السجلات المراد قراءتها في هذه اللحظة، ونلاحظ أننا استخدمنا الدالة SizeOf التي ترجع حجم المتغير من حيث عدد البايتات، وفي هذه الحالة هو الرقم 1024.

NumRead: عندما نقول أننا نريد قراءة 1024 بايت ربما ينجح الإجراء بقراءتها جميعاً في حالة أن تكون هناك بيانات متوفر في الملف، أما إذا كانت محتويات الملف أقل من هذه القيمة أو أن مؤشر القراءة وصل قرب نهاية الملف، ففي هذه الحالة لايستطيع قراءة 1024 بايت، وتكون القيمة التي قرأها أقل من ذلك وتخزن القيمة في المتغير NumRead. فمثلاً إذا كان حجم المف 1034 بايت، فيقوم الإجراء

بقراءة 1024 بايت في المرة الأولى، أما في المرة الثانية فيقوم بقراءة 10 بايت فقط ويرجع هذه القيمة في المتغير NumRead حتى يتسنى إستخدامها مع الإجراء BlockWrite.

5.طريقة الكتابة:

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

وأظن أنها واضحة، والمتغير الأخيرة NumRead في هذه المرة عدد البايتات المراد كتابتها في الملف المنسوخ، وهي تعني عدد البياتات من بداية المصفوفة Block. وعند تشغيل هذا الإجراء فإن المتغير NumRead يحمل قيمة عدد البايتات التي تمت قراءتها عن طريق BlockRead

وعند تشغيل البرنامج سوف نلاحظ السرعة الكبيرة في نسخ الملفات، فمثلاً إذا كان طول الملف مليون بايت، فيلزم حوالي أقل من ألف دورة فقط للقراءة ثم الكتابة في الملف الجديد.

في المثال التالي، يقوم البرنامج بإظهار محتويات الملف بالطريقة المستخدمة في التخزين في الذاكرة أو الملف، فكما سبق ذكره فإن الملف هو عبارة عن سلسلة من الحروف أو البايتات.

برنامج عرض محتويات ملف بالبايت

```
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 تنفيذ البرنامج يمكن للمستخدم كتابة إسم ملف نصي حتى نقوم بعرض محتوياته.
نلاحظ أن بعد كل سطر نجد علامة الـ Line Feed LF وقيمتها بالبايت 10 في نظام لينكس، أما في نظام وندوز فنجد علامتي Carriage Return/Line Feed CRLF وقيمتهما على التوالي 13 و 10، وهي الفاصل الموجود بين السطور في الملف النصي. يمكن كذلك إستخدام البرنامج في فتح ملفات من نوع آخر لمعرفة طريقة تكوينها.
إستخدمنا في البرنامج الدالة Chr التي تقوم بتحويل البايت إلى حرف، مثلاً نجد أن البايت الذي قيمته 97 يمثل الحرف a وهكذا.

التاريخ والوقت Date and Time

التاريخ والوقت من الأساسيات المهمة في البرامج، فهي من الخصائص المهمة المرتبطة بالبيانات والمعلومات، فمعظم المعلومات والبيانات تحتاج لتاريخ يمثل وقت إدراج هذه المعلومة أو تعديلها. فمثلاً إذا كانت هناك معاملة بنكية أو صرف شيك فلابد من تسجيل الوقت الذي حدثت فيه هذه المعاملة نسبة لأهميتها في المراجعة لاحقاً عندما يطلب العميل كشف الحساب مثلاً، أو عندما يقوم البنك بحصر المعاملات التي حدثت في شهر ما مثلاً.

يتم تخزين التاريخ والوقت في متغير واحد من نوع TDateTime وهو عبارة عن عدد حقيقي (كسري) ذو دقة مضاعفة Double وهو يحتل ثماني خانات من الذاكرة. العدد الصحيح منه يمثل الأيام، والجزء الكسري يمثل الوقت. والقيمة صفر لمتغير من هذا النوع تمثل الأيام التي انقضت منذ يوم 30/12/1899 ميلادية.

في البرنامج التالي نستخدم الدالة Now والتي تقوم بإرجاع قيمة التاريخ والزمن الحاليين:

```
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.
```

نلاحظ أننا استخدمنا الدالة DateTimeToStr الموجودة في المكتبة SysUtils والتي تقوم بتفسير قيمة التاريخ والوقت وتحويلهما لمقطع مقروء حسب إعدادات المستخدم في طريقة إظهار التاريخ.
فإذا لم نستخدمها سوف نحصل على عدد حقيقي لايمكن فهمه:

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

كذلك توجد دوال لإظهار قيمة التاريخ فقط وأخرى لإظهار قيمة الوقت فقط مثلاً:

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

كذلك توجد دالتان مشابهتان للدالة Now لكنهما يرجعان التاريخ فقط والأخرى الزمن فقط وهما Date, Time

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

هذه الدوال تقوم بحذف الجزء الآخر أو تحويله إلى صفر، فالدالة Date ترجع فقط التاريخ وتقوم بإرجاع صفر في الجزء الكسري الذي يمثل الزمن. والقيمة صفر بالنسبة للزمن تمثل الساعة 12 صباحاً.
والدالة Time تقوم بإرجاع الزمن ووضع القيمة صفر في الجزء الصحيح الذي يمثل التاريخ والذي قيمته كما ذكرنا 30/12/1899. يمكن ملاحظة هذه القيم في المثال التالي عند استخدام الدالة DateTimeToStr مع الدوال الجديدة Date, Time

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

يمكن أن يتضح المثال السابق أكثر بإستخدام الدالة FormatDateTime والتي تقوم بتحويل التاريخ والوقت بحسب الشكل الذي يريده المبرمج بغض النظر عن إعدادات المستخدم في حاسوبه:

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

في المثال التالي سوف نعامل التاريخ كعدد حقيقي فنقوم بإضافة أو طرح قيم منه.

```
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.
```

نلاحظ عندما نقوم بزيادة الرقم 1 أو طرحه من قيمة التاريخ الحالي Now فإننا نقوم بزيادة يوم أو بإنقاص يوم على التوالي. وعندما نقوم بزيادة نصف يوم ½ فهي تعني 12 ساعة، في هذه الحالة أثرنا على الجزء الكسري في التاريخ الذي يمثل الوقت.

في المثال التالي نقوم بتكوين تاريخ معين من السنين، الشهور، والأيام:

```
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.
```

نلاحظ في المثال السابق أننا استخدمنا الدالة EncodeDate والتي مدخلاتها هي السنة، الشهر ثم اليوم، وهي تقوم بتحويل وتجميع هذه القيم في متغير واحد من النوع TDateTime.

في المثال التالي سوف نستخدم دالة مشابهة لكنها تختص بالوقت:

```
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.
```

مقارنة التواريخ والأوقات

يمكن مقارنة التواريخ والأوقات بنفس طريقة مقارنة الأعداد الحقيقية، فالرقم 9.3 أكبر من الرقم 5.1 .
فكما علمنا يمثل العدد الصحيح الأيام أو التاريخ، ويمثل العدد الكسري الوقت. فبالتأكيد قيمة Now + 1 والتي تعني غداً تكون دائماً أكبر من Now و قيمة Now + 1/24 والتي تعني بعد ساعة من الآن تكون أكبر من Now – 2/24 والتي تعني قبل ساعتين من الآن.

في البرنامج التالي نضع تاريخ معين وهو عام 2012، اليوم الأول من شهر يناير، فنقوم بمقارنته بالتاريخ الحالي لنعرف هل تجاوزنا هذا التاريخ أم ليس بعد.

```
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.
```

يمكن إضافة معلومة جديدة في البرنامج وهي عدد الأيام المتبقية لهذا التاريخ أو الأيام التي إنقضت بعده في المثال التالي:

```
var
  Year2012: TDateTime;
  Diff: Double;
begin
```

```
  Year2010:= 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.
```

في المثال السابق سوف يحتوي المتغير Diff قيمة فرق التارخين بالأيام. ونلاحظ أننا استخدمنا الدالة Abs والتي تقوم بإرجاع القيمة المطلقة لعدد (أي حذف السالب إن وجد).

مسجل الأخبار

البرنامج التالي يستخدم الملفات النصية لتسجيل عناوين الأخبار المكتوبة ويقوم بإضافة الزمن الذي تمت كتابة الخبر عنده بإعتبارة وقت حدوث الخبر. عند إغلاق البرنامج وتشغيله مرة أخرى يعرض البرنامج الأخبار السابقة التي تمت كتابتها مع عرض الزمن الذي كتبت فيه.

```
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.
```

الثوابت constants

الثوابت تشبه المتغيرات من حيث أن كل يمثل قيمة معينة مثل قيمة لعدد صحيح، حقيقي أو مقطع. إلا أن الثوابت تبقى قيمتها ثابتة لاتتغير، بل أن قيمتها محددة في وقت ترجمة البرنامج. بالتالي نستطيع أن نقول أن المترجم يعرف قيمة الثابت سلفاً أثناء قيامه بترجمة البرنامج وتحويله إلى لغة آلة.
مثال لقيمة ثابتة:

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

نجد أن البرنامج عند تنفيذ هذا السطر يكتب الرقم 5 في كل الأحوال ولايمكن أن تتغير هذه القيمة، كذلك فإن المقطع Hello لايتغير.

عندما نتكلم عن الثوابت فإننا نقصد الأرقام المحددة أو المتغيرات المقطعية أو أي نوع آخر تمت كتابته في البرنامج بالطريقة السابقة أو بالطريقة التالية:

برنامج إستهلاك الوقود

```
program PetrolConsumption;

{$mode objfpc}{$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 petrol: ');
  Readln(Payment);
  Write('What is the consumption of your car? (Kilos per Gallon): ');
  Readln(Consumption);

  Kilos:= (Payment / GallonPrice) * Consumption;

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

البرنامج السابق يحسب عدد الكيلومترات التي سوف تقطعها السيارة بناءً على عدة عوامل:

1. معدل إستهلاك السيارة للوقود، وقد استخدمنا المتغير Consumption حتى نضع فيه عدد الكيلومترات التي تقطعها السيارة بالجالون الواحد من الوقود
2. قيمة النقود التي دفعها سائق السيارة لشراء الوقود، وقد استخدمنا المتغير Payment
3. سعر الجالون، وقد استخدمنا الثابت GallonPrice لتخزين قيمة سعر الجالون والتي تعتبر قيمة ثابتة نسبياً قمنا بتثبيتها في البرنامج حتى لانطلب من المستخدم إدخالها في كل مرة مع الـ payment وال Consumption. فإذا تغير سعر الوقود ماعلينا إلا الذهاب إلى أعلى البرنامج وتغير قيمة هذه الثابت ثم إعادة ترجمة البرنامج.

يفضل استخدام هذه الطريقة للثوابت عندما نستخدمها لأكثر من مرة في البرنامج، فتغيير القيمة في التعريف تغني عن التغيير في كل أجزاء البرنامج.

الأنواع المعدودة Ordinal types

الأنواع المعدودة هي عبارة عن نوع يحمل قِيم عددية صحيحة ذات أسماء، ونعاملها بأسمائها لتسهيل قراءة الكود وفهم البرنامج. مثلاً في البرنامج التالي نريد أن نقوم بتخزين اللغة التي يريد استخدامها مستخدم البرنامج، وقد قمنا بتخييره بين اللغة العربية والإنجليزية، وكان من الممكن أن نستخدم متغيِّر صحيح ونضع فيه قيم مثل 1 للغة العربية و 2 للغة الإنجليزية. لكننا استخدمنا الأنواع المعدودة ليكون البرنامج أكثر وضوحاً للمبرمج:

```
program OrdinaryTypes;

{$mode objfpc}{$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.
```

الأرقام الصحيحة والحروف تعتبر من الأنواع المعدودة،أما الأعداد الكسرية والمقاطع فلاتعتبر من الأنواع المعدودة.

المجموعات sets

المجموعات هي عبارة عن نوع من المتغيرات يستطيع جمع عدد من القيم أو الخصائص، شرط أن تكون من الأنواع المعدودة، فمثلاً عندما نريد تصنيف برامج من حيث أنظمة التشغيل التي تعمل فيها نلجأ إلى الآتي:

1. تعريف نوع معدود يمثل أنظمة التشغيل، كما في المثال التالي TApplicationEnv
2. نقوم بتعريف متغيرات للبرامج المراد تصنفيها بالشكل التالي:

```
  AppName: set of TApplicationEnv;
```

3. نقوم بإسناد أنواع أنظمة التشغيل في شكل مجموعة كالتالي:

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

والمثال هو:

```
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.
```

يمكن كذلك إستخدام صيغة المجموعات مع الأنواع المعدودة أو الصحيحة كالآتي:
مثال لإستخدام متغير صحيح Month

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

كذلك يمكننا إستخدام الحروف كلآتي:

```
if Char in ['آ' ,'إ' ,'أ' ,'ا'] then
    Writeln('هذا حرف الألف');
```

معالجة الإعتراضات Exception handling

يوجد نوعان من الأخطاء: أخطاء أثناء الترجمة compilation errors مثل إستخدام متغيّر بدون تعريفه في قسم var، أو كتابة عبارة بطريقة خاطئة. فهذه الأخطاء لايمكن تجاوزها، وتحول دون ترجمة وربط البرامج، ويقوم المترجم بالتنبيه عن وجودها في نافذة Messages.
أما النوع الثاني من الأخطاء فهي الإعتراضات التي تحدث بعد تنفيذ البرنامج، مثلاً إذا طلب البرنامج من المستتخدم إدخال رقمين لإجراء القسمة عليهما، وقام المستخدم بإدخال صفر في المقسوم عليه، ففي هذه الحالة يحدث الخطأ المعروف بالقسمة على الصفر Division by Zero وهي من الأشياء الممنوعة في الرياضيات. أو مثل محاولة فتح ملف غير موجود، أو محاولة الكتابة في ملف ليس للمستخدم صلاحية عليه (من ناحية نظام التشغيل). فمثل هذه الأخطاء تسمى أخطاء في وقت التنفيذ run-time errors. وليس من السهولة أن يتوقعها المترجم، لذلك فمعالجة هذه الإستثناءات تُركت على عاتق المبرمج، حيث أن على المبرمج أن يقوم بكتابة برنامج أو كود شديد الإعتمادية reliable يستطيع التغلب على معظم الأخطاء التشغيلية التي يمكن أن تحدث.
معالجة الإعتراضات تتم بعدة طُرق منها حماية جزء معين من الكود بإستخدام عبارة try except أو try finally

عبارة try except

تكون بهذه الطريقة:

```
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 لبرنامج قسمة عددين بإستخدام except:

```
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 occured: ', e.message);
  end;
  end;
  Write('Press enter key to close');
  Readln;
end.
```

نلاحظ أن هناك قسمين للكود: الكود باللون الأصفر هو الكود المحمي، والكود الأحمر يمثل قسم معالجة الإستثناءات، ويتم نتفيذه فقط عند حدوث خطأ.

عندما نقوم بإدخال القيمة صفر للمتغير y فإن الكود في قسم معالجة الإستثناءات باللون الأحمر يتم تنفيذه. ومؤشر التنفيذ في البرنامج عند حدوث خطأ يقفز إلى قسم معالجة الإستثناءات ويترك تنفيذ باقي الكود المحمي، ففي هذه الحالة الكود التالي لايتم تنفيذه عند حدوث خطأ أثناء القسمة:

```
Writeln('x / y = ', Res);
```

أما في حالة عدم حدوث خطأ ، أي في الحالات العادية يتم تنفيذ كل الكود المحمى ولايتم تنفيذ الكود في قسم معالجة الأخطاء باللون الأحمر.

عبارة try finally

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

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

مثال لبرنامج قسمة عددين بإستخدام 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.
```

في هذه المرة فإن الكود في قسم finally باللون الأخضر الفاتح، يتم تنفيذه في كافة الأحوال، في حال حدوث خطأ أو في حال عدم حدوث خطأ. وبهذه الطريقة نضمن تنفيذ كود معين في كافّة الأحوال.

رفع الإستثناءات raise an exception

يمكن للمبرمج رفع إستثناء، وذلك عند حدوث شرط معين، فمثلاً إذا طلبنا من المستخدم الإلتزام بمدى معين لمُدخل ما، وقام المستخدم بتجاوز هذا المدى، حينئذٍ يمكن إفتعال إشتناء مصحوب برسالة خطأ معين. مثلاً البرنامج التالي:

```
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.
```

نلاحظ أنه إذا قام المستخدم بتجاوز المدى والذي هو محدد من 1 إلى 10، فإن الخطأ التالي سوف يحدث في البرنامج x is out of range وبما أننا في نفس البرنامج قد قمنا بحماية الكود بعبارة try except فإن هذا الخطأ سوف يظهر للمستخدم بطريقة لاتؤثر على إستمرارية البرنامج.

الفصل الثاني

البرمجة الهيكلية Structured Programming

مقدمة

البرمجة الهيكلية هي طريقة للبرمجة ظهرت الحاجة لها بعد التوسع الذي حدث في البرامج، حيث أصبح من الصعب كتابة وتنقيح وتطوير البرامج الكبيرة المكتوبة في شكل ملف واحد ككتلة واحدة.
البرمجة الهيكلية هي إتباع إسلوب تقسيم البرامج إلى وحدات أو مكتبات وإجراءات ودوال في شكل منظم ومجزاء يمكن الإستفادة منها أكثر من مرة مع سهولة القراءة وتتبع الأخطاء في الأجزاء التي بها مشاكل في البرامج. فنجد أن من أهم فوائد البرمجة الهيكلية هي:

1.تقسيم البرنامج إلى وحدات برمجية أصغر تُسّهل فهم البرامج
2.إمكانية إستخدام (نداء) هذه الإجراءات في أكثر من موضع في البرنامج الواحد أو عدد من البرامج بإستخدام الوحدات. وهذه الخاصية تسمى بإعادة إستخدام الكود code reusability
3.إمكانية مشاركة تصميم وكتابة برنامج واحد لأكثر من مبرمج، حيث يمكن لأي مبرمج كتابة وحدة منفصلة أو إجراء منفصل ثم يتم تجميعها في برنامج واحد.
4.سهولة صيانة البرنامج، حيث تسهل معرفة الأخطاء وتصحيحها بالتركيز على الدالة أو الإجراء الذي ينتج عنه الخطأ.

الإجراءات procedures

لعلنا استخدمنا البرمجة الهيكلية من قبل، فقد مر علينا كثير من الدوال والإجراءات والوحدات التي هي عبارة عن أجزاء من مكتبات فري باسكال، لكن الفرق الآن أن المبرمج في هذه المرحلة سوف يقوم بكتابة إجراءات ودوال خاصة به تمثل خصوصية البرنامج الذي يقوم بكتابته.

في المثال التالي قمنا بكتابة إجرائين: SayHello و SayGoodbye

```
program Structured;

{$mode objfpc}{$H+}

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

procedure SayHello;
begin
  Writeln('Hello there');
end;

procedure SayGoodbye;
begin
  Writeln('Good bye');
end;

begin  // Here main application starts
  Writeln('This is the main application started');
  SayHello;
```

```
  Writeln('This is structured application');
  SayGoodbye;
  Write('Press enter key to close');
  Readln;
end.
```

أظن أن الفكرة واضحة. ففي حالة مناداتنا لهذه الإجراءات يقوم البرنامج بتنفيذ الكود المكتوب في الإجراء والذي كأنه برنامج مصغّر.

المدخلات Parameters

في البرنامج التالي نقوم بعمل إجراء يحتوي على مدخلات parameters. والمدخلات هي قيم تمرر عند نداء الإجراء:

```
procedure WriteSumm(x, y: Integer);
begin
  Writeln('The summation of ', x, ' + ', y, ' = ', x + y)
end;

begin
  WriteSumm(2, 7);
  Write('Press enter key to close');
  Readln;
end.
```

نلاحظ أن القيمتان 2 و 7 مررتا عند نداء الإجراء WriteSumm والذي يقوم بإستقبال هذه القيم في المتغيرات x, y ليكتب حاصل جمعهما.

في المثال التالي نقوم بإعادة كتابة برنامج المطعم بطريقة الإجراءات:

برنامج المطعم بإستخدام الإجراءات

```
procedure Menu;
begin
  Writeln('Welcome to Pascal Resturant. 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;
end;

procedure GetOrder(AName: string; Minutes: Integer);
begin
  Writeln('You have ordered : ', AName, ', this will take ',
    Minutes, ' minutes');
```

```
end;


// Main application

var
  Meal: Byte;
begin

  Menu;
  Write('Please enter your selection: ');
  Readln(Meal);

  case Meal of
    1: GetOrder('Chicken', 15);
    2: GetOrder('Fish', 12);
    3: GetOrder('Meat', 18);
    4: GetOrder('Salad', 5);
    5: GetOrder('Orange juice', 2);
    6: GetOrder('Milk', 1);
  else
    Writeln('Wrong entry');
  end;
  Write('Press enter key to close');
  Readln;
end.
```

نلاحظ أن البرنامج الرئيسي أصبح أوضح وسهل الكتابة، والأجزاء الأخرى مثل القائمة الرئيسية Menu و الطلب منها أصبحتا إجراءات منفصلة. كذلك فإن الجزء الرئيسي في البرنامج Main أصبح أصغر وتم التخلص من أجزاء كبيرة من الكود لتصبح إجراءات منفصلة.
نلاحظ أن الإجراء GetOrder تم ندائه عدد من المرات وفي كل مرة بمدخلات مختلفة، وهنا تحقق إعادة إستخدام الكود.

الدوال functions

الدوال شبيهة بالإجراءات إلا أنها تزيد عليها بأنه تقوم بإرجاع قيمة. فقد استخدمنا دوال من قبل مثل Abs التي ترجع القيمة المطلقة، و UpperCase التي تستقبل المقطع كمدخل وتقوم بإرجاع مقطع جديد محول إلى الحروف الإنجليزية الكبيرة.

في البرنامج التالي قمنا بكتابة دالة GetSumm والتي تستقبل مدخلين من الأعداد الصحيحة فتقوم بإرجاع قيمة حاصل جمعهما فقط دون كتابته:

```
function GetSumm(x, y: Integer): Integer;
begin
  Result:= x + y;
end;

var
  Sum: Integer;
begin
  Sum:= GetSumm(2, 7);
  Writeln('Summation of 2 + 7 = ', Sum);
  Write('Press enter key to close');
  Readln;
end.
```

نلاحظ هذه المرة أن للدالة نوع وهو **Integer** في هذه الحالة. كذلك استخدمنا الكلمة Result والتي تعبر عن القيمة التي سوف ترجع عند مناداة هذه الدالة. وعند ندائها قمنا بإستخدام المتغير Sum لإستقبال قيمة حاصل الجمع. لكن يمكن أن نستغنى عن هذا المتغير ونقوم بإدخال نداء الدالة في الإجراء Writeln وهو فرق آخر للدوال عن الإجراءات، حيث لايمكن نداء إجراء بهذه الطريقة، فقط الدوال:

```
function GetSumm(x, y: Integer): Integer;
begin
  Result:= x + y;
end;

begin
  Writeln('Summation of 2 + 7 = ', GetSumm(2, 7));
  Write('Press enter key to close');
  Readln;
end.
```

في البرنامج التالي قمنا بإعادة كتابة برنامج المطعم بإستخدام الدوال:

برنامج المطعم بإستخدام الدوال وحلقة repeat

```
procedure Menu;
begin
  Writeln('Welcome to Pascal Resturant. 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;
end;

function GetOrder(AName: string; Minutes, Price: Integer): Integer;
begin
  Writeln('You have ordered: ', AName, ' this will take ',
    Minutes, ' minutes');
  Result:= Price;
end;


var
  Selection: Char;
  Price: Integer;
  Total: Integer;
begin
  Total:= 0;
  repeat
    Menu;
    Write('Please enter your selection: ');
    Readln(Selection);

    case Selection of
      '1': Price:= GetOrder('Checken', 15, 10);
      '2': Price:= GetOrder('Fish', 12, 7);
      '3': Price:= GetOrder('Meat', 18, 8);
      '4': Price:= GetOrder('Salad', 5, 2);
      '5': Price:= GetOrder('Orange juice', 2, 1);
      '6': Price:= GetOrder('Milk', 1, 1);
      'x', 'X': Writeln('Thanks');
      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.
```

المتغيرات المحلية Local Variables

يمكن تعريف متغيارت محلية داخل الإجراء أو الدالة بحيث يكون استخدامها فقط محلياً في الإجراء أو الدالة ولايمكن الوصول لهذه المتغيرات من البرنامج الرئيسي. مثلاً:

```
procedure Loop(Counter: Integer);
var
  i: Integer;
  Sum: Integer;
begin
  Sum:= 0;
  for i:= 1 to Counter do
    Sum:= Sum + i;
  Writeln('Summation of ', Counter, ' numbers is: ', Sum);
end;

begin   // Main program section
  Loop;
  Write('Press enter key to close');
  Readln;
end.
```

نلاحظ أن الإجراء Loop به تعريفات لمتغيرات محلية وهي i, Sum يتم حجز جزء من الذاكرة لهما مؤقتاً في مايعرف بالمكدسة Stack وهي ذاكرة للمتغيرات المؤقتة مثل المتغيرات المحلية(i, Sum) في المثال السابق والمدخلات للإجراءات والدوال مثل المدخل Counter في هذا المثال. وتتميز المتغيرات التي تخزن في هذا النوع من الذاكرة بقصر عمرها، إذ أن المتغيرات تكون صالحة فقط في الفترة التي يقوم البرنامج بنداء وتنفيذ هذه الدوال والإجراءات، فبعد نهاية تنفيذ الدالة والفراغ منها، كما في المثال السابق عند الوصول لهذه النقطة في البرنامج الرئيسي:

```
  Write('Press enter key to close');
```

تكون هذه المتغيرات قد إنقضت ولايمكن الوصول إليها أو لقيمها مرة أخرى بالطرق العادية.
وهذه بخلاف المتغيرات التي يقوم المبرمج بتعريفها للإستخدام في البرنامج الرئيسي كما اعتدنا استخدمناها سابقا، فهي تكون أكثر عمراً حيث أنها تكون صالحة للإستخدام منذ بداية تشغيل البرنامج إلى نهايته، فإذا كان البرنامج يحتوي على حلقة فإنها تعمّر فترة طويلة، وإذا تم تشغيل البرنامج لمدة ساعة، كان عمر هذه المتغيرت ساعة، وهكذا.

هذه الطريقة لتعريف المتغيرات محلياً هي إحدى طرق تحقيق البرمجة الهيكلية، حيث أنها توفر حماية وخصوصية لهذا الجزء القائم بذاته من البرنامج (الإجراء أو الدالة) وتجعله غير مرتبط بمتغيرات أخرى في البرنامج مما يسهّل نقله لبرامج أخرى أو الإستفادة منه عدة مرات. فمن أراد نداء هذه الدالة Loop ماعليه إلا مدها بقيمة Counter ثم يقوم الإجراء بتلبية إحتياجاته محلياً من المتغيرات التي تعينه على تنفيذ هذا الكود مما يحقق إعادة إستخدام الكود وسهولة الصيانة.

برنامج قاعدة بيانات الأخبار

البرنامج التالي فيه ثلاث إجراءات ودالة واحدة، وفيه عدة إمكانات وهي : إضافي عنوان خبر جديد، عرض كافة الأخبار والبحث عن خبر بإستخدام كلمة مفتاحية:

```
program news;

{$mode objfpc}{$H+}

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

type
  TNews = record
    ATime: TDateTime;
    Title: string[100];
  end;

procedure AddTitle;
var
  F: file of TNews;
  News: TNews;
begin
  AssignFile(F, 'news.dat');
  Write('Input current news title: ');
  Readln(News.Title);
  News.ATime:= Now;
  if FileExists('news.dat') then
  begin
    FileMode:= 2; // Read/Write
    Reset(F);
    Seek(F, System.FileSize(F)); // Go to last record to append
  end
  else
    Rewrite(F);
  Write(F, News);
  CloseFile(F);
end;

procedure ReadAllNews;
var
  F: file of TNews;
  News: TNews;
begin
  AssignFile(F, 'news.dat');
  if FileExists('news.dat') then
  begin
    Reset(F);
    while not Eof(F) do
    begin
      Read(F, News);
```

```
      Writeln('------------------------------');
      Writeln('Title: ', News.Title);
      Writeln('Time : ', DateTimeToStr(News.ATime));
    end;
    CloseFile(F);
  end
  else
    Writeln('Empty database');
end;


procedure Search;
var
  F: file of TNews;
  News: TNews;
  Keyword: string;
  Found: Boolean;
begin
  AssignFile(F, 'news.dat');
  Write('Input keyword to search for: ');
  Readln(Keyword);
  Found:= False;
  if FileExists('news.dat') then
  begin
    Reset(F);
    while not Eof(F) do
    begin
      Read(F, News);
      if Pos(LowerCase(Keyword), LowerCase(News.Title)) > 0 then
      begin
        Found:= True;
        Writeln('------------------------------');
        Writeln('Title: ', News.Title);
        Writeln('Time : ', DateTimeToStr(News.ATime));
      end;
    end;
    CloseFile(F);
    if not Found then
      Writeln(Keyword, ' Not found');
  end
  else
    Writeln('Empty database');
end;

function Menu: char;
begin
  Writeln;
  Writeln('...........News database..........');
  Writeln('1. Add news title');
  Writeln('2. Display all news');
  Writeln('3. Search');
  Writeln('x. Exit');
  Write('Please input a selection : ');
  Readln(Result);
end;
```

```
// Main application

var
  Selection: Char;

begin

  repeat
    Selection:= Menu;
    case Selection of
      '1': AddTitle;
      '2': ReadAllNews;
      '3': Search;
    end;
  until LowerCase(Selection) = 'x';
end.
```

نجد أن برنامج الأخبار هذه المرة، أصبح أكثر وضوحاً ومقسم إلى أقسام قائمة بذاتها، حيث يمكن للمبرمج حتى ولو لم يكن هو الذي كتب البرنامج أن يفهمه من أول وهلة وتعدليه أو تطويره أو زيادة ميزات أخرى (زيادة إجراءات أو دوال) بكل سهولة. كذلك يمكن الإستفادة من بعض أجزائه في برامج أخرى.

الدالة كمُدخل

الفرق الثاني والمهم بين الدوال والإجراءات هو أن الدالة يمكن ندائها من داخل إجراء أو دالة أخرى كمُدخل ، وهذه الميزة ناتجة عن الخاصية الأولى للدوال وهي أنها ترجع قيمة، أي يكن معاملتها كأنها قيمة ثابتة أو متغير نريد قراءة قيمته، مثلاً:

```
function DoubleNumber(x: Integer): Integer;
begin
  Result:= x * 2;
end;

// Main

begin
  Writeln('The double of 5 is : ', DoubleNumber(5));
  Readln;
end.
```

نلاحظ أننا قمنا بتمرير الدالة DoubleNumber داخل الإجراء Writeln ، وكان يمكن أن نستخدم الطريقة الطويلة كالآتي:

```
function DoubleNumber(x: Integer): Integer;
begin
  Result:= x * 2;
end;

// Main

var
  MyNum: Integer;
begin
  MyNum:= DoubleNumber(5);
  Writeln('The double of 5 is : ', MyNum);
  Readln;
end.
```

كذلك يمكن نداء الدوال داخل الشروط وهذه الميزات لاتتحقق في الإجراءات:

```
function DoubleNumber(x: Integer): Integer;
begin
  Result:= x * 2;
end;

// Main

begin
  if DoubleNumber(5) > 10 then
    Writeln('This number is larger than 10')
  else
    Writeln('This number is equal or less than 10);
  Readln;
end.
```

مخرجات الإجراءات والدوال

لاحظنا من الأمثلة السابقة أننا نستطيع جعل الدالة ترجع قيمة واحدة، لكن ماذا لو أردنا إرجاع قيمتين أو أكثر، ففي هذه الحالة لايصلح أن نستخدم فقط قيمة ماترجعه الدالة.
يكمن الحل لهذه المشكلة في إستخدام المدخلات للإجراءات والدوال. حيث نجد أن المدخلات نفسها يمكن أن تكون مخرجات بتغيير بسيط في طريقة التعريف. لكن لشرح هذا المفهوم دعنا نقوم بهذه التجربة:

```
procedure DoSomething(x: Integer);
begin
  x:= x * 2;
  Writeln('From inside procedure: x = ', x);
end;

//  main

var
  MyNumber: Integer;
begin
  MyNumber:= 5;

  DoSomething(MyNumber);
  Writeln('From main program, MyNumber = ', MyNumber);
  Writeln('Press enter key to close');
  Readln;
end.
```

نجد في التجربة السابقة أن الإجراء DoSomething يقوم بإستقبال مدخل في المتغير x ثم يقوم بمضاعفة قيمة هذا الرقم وإظهاره في الشاشة. وفي البرنامج الرئيسي قمنا بتعريف متغير MyNumber ووضعنا فيه القيمة 5 ثم قمنا بنداء الإجراء DoSomething واستخدمنا المتغير MyNumber كمدخل لهذا الإجراء. في هذه الحالة تنتقل قيمة المتغير MyNumber إلى المتغير x في الإجراء السابق الذكر. في هذه الحالة يقوم الإجراء بمضاعفة قيمة x والتي بدورها تحتوي على القيمة 5 لتصبح 10 ثم يقوم بإظهار هذه القيمة. بعد الفراغ من تنفيذ هذا الإجراء يعود مسار البرنامج إلى الجزء الرئيسي من البرنامج ونقوم بإظهار قيمة المتغير MyNumber الذي نجده يحتفظ بالقيمة 5، أي أن مضاعفة قيمة x لم تؤثر عليه. هذه الطريقة من النداء تسمى النداء بالقيمة calling by value، حيث أننا نسخنا قيمة المدخل MyNumber إلى متغير x جديد مؤقت لايؤثر على المتغير في البرنامج الرئيسي.

النداء بإستخدام المرجع calling by reference

إذا قمنا بإضافة الكلمة var إلى الإجراء السابق سوف تتغير الموازين فتصبح النتيجة مختلفة هذه المرة:

```
procedure DoSomething(var x: Integer);
begin
  x:= x * 2;
  Writeln('From inside procedure: x = ', x);
end;

//  main
```

```
var
  MyNumber: Integer;
begin
  MyNumber:= 5;

  DoSomething(MyNumber);
  Writeln('From main program, MyNumber = ', MyNumber);
  Writeln('Press enter key to close');
  Readln;

end.
```

في هذه الحالة نجد أن مضاعفة قيمة x أثرت مباشرة على قيمة MyNumber حيث أنهما يتشاركان الموقع أو العنوان في الذاكرة، أما في الحالة السابقة بدون var فكان لكل متغير منها حيز منفصل في الذاكرة.
لكن الشرط هذه المرة أن نمرر مدخل في شكل متغير من نفس النوع ولايصلح أن يكون ثابت، حيث لايمكن أن نقوم بنداء الدالة بهذه الطريقة:

```
  DoSomething(5);
```

في الحالة الأولى كان يمكن أن يكون هذا الإجراء صحيح لأن الطريقة كانت نداء بوساطة قيمة وفي هذه الحالة القيمة فقط هي المهمة وهي 5.

في المثال التالي قمنا بكتابة إجراء يقوم بإستبدال قيمتين ببعضهما أو مايسمى بالـ Swap

```
procedure SwapNumbers(var x, y: Integer);
var
  Temp: Integer;
begin
  Temp:= x;
  x:= y;
  y:= Temp;
end;

//  main

var
  A, B: Integer;
begin

  Write('Please input value for A: ');
  Readln(A);

  Write('Please input value for B: ');
  Readln(B);

  SwapNumbers(A, B);
  Writeln('A = ', A, ', and B = ', B);
  Writeln('Press enter key to close');
  Readln;
end.
```

الوحدات units

الوحدات هي عبارة عن مكتبات للإجراءات والدوال التي تستخدم بكثرة بالإضافة إلى الثوابت، و المتغيرات والأنواع المعرفة بواسطة المستخدم. ويمكن استخدام هذه الإجراءات والدوال الموجودة في هذه الوحدات ونقلها لعدة برامج مما يحقق إحدى أهداف البرمجة الهيكلية وهي إعادة الإستفادة من الكود. كذلك فهي تقلل من إزدحام البرنامج الرئيسي حيث يتم وضع الإجراءات والدوال التي تلبي حل مشكلة معينة في وحدة خاصة حتى تسهل صيانة وقراءة الكود.

لإنشاء وحدة جديدة ماعلينا إلى الذهاب إلى File/New Unit فنجد شكل وحدة جديدة فيها هذا الكود:

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils;

implementation

end.
```

بعد ذلك نقوم بإعطائها إسم وحفظها كملف بنفس الإسم، مثلاً إذا اخترنا اسم Test للوحدة بدلاً من Unit1 لابد من حفظها كملف بإسم Test.pas. بعد ذلك نقوم بكتابة الإجراءات والدوال التي نريد استخدامها لاحقاً:

```
unit Test;

{$mode objfpc}{$H+}

interface


uses
  Classes, SysUtils;

const
  GallonPrice = 6.5;

  function GetKilometers(Payment, Consumption: Integer): Single;


implementation


function GetKilometers(Payment, Consumption: Integer): Single;
begin
  Result:= (Payment / GallonPrice) * Consumption;
end;

end.
```

قمنا بكتابة ثابت GallonPrice ودالة GetKilometers ليتم استخدامهما في أي برنامج.
نلاحظ أننا قمنا بإعادة كتابة رأس الدالة في قسم الـ **Interface** أما كود الدالة فهو مكتوب في قسم **Implementation**. وضرورة إعادة كتابة رأس الدالة أو الإجراء في قسم الـ interface يُمكّن باقي البرامج من رؤية هذه الدالة ومن ثم مناداتها، فالدوال الغير مكتوبة في هذا القسم لايمكن ندائها من خارج الوحدة، بل يمكن فقط ندائها داخل هذه الوحدة.

قمنا بحفظ هذا الملف في نفس المجلد الذي يوجد فيه البرنامج التالي:

```
program PetrolConsumption;

{$mode objfpc}{$H+}


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


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

  Kilos:= GetKilometers(Payment , Consumption);

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

نلاحظ أننا قمنا بإضافة إسم الوحدة Test في قسم الـ Uses وقمنا بنداء الدالة GetKilometers في هذا البرنامج.

لإستخدام وحدةٍ ما في برنامج يجب تحقق واحد من هذه الشروط:

1. يكون ملف الوحدة موجود في نفس المجلد الذي يوجد فيه البرنامج
2. فتح ملف الوحدة المعينة ثم إضافتها للمشروع بواسطة Project/Add Editor File to project
3. إضافة مسار أو مجلد الوحدة في Project/Compiler Options/Paths/Other Unit Files

الوحدات وبنية لازاراس وفري باسكال

نجد أن الوحدات في مترجم فري باسكال وأداة التطوير لازاراس تمثل البنية الأساسية للغة البرمجة، حيث نجدهما يحتويان على عدد كبير من الوحدات التي تمثل مكتبة لغة أوبجكت باسكال والتي بدورها تحتوي على عدد كبير جداً من الدوال والإجراءات والأنواع وغيرها الكثير من الأشياء التي جعلت لغة أوبجكت باسكال لغة عنية ثرية بكافة الأدوات التي يحتاجها المبرمج لبناء النظم والبرامج الكبيرة بسرعة وبدون جهد كبير. والسبب في ذلك أن هذه اللغة أخذت وقت كافي من التطوير والإستخدام وقد صاحب هذا الإستخدام بناء وحدات تمثل الخبرة التي قام بجمعها المبرمجين من كافة أرجاء العالم ليصبوها في شكل مكتبات مُجربة يمكن الإستفادة منها في معظم البرامج.
هذه الوحدات التي يحتويها لازاراس هي وحدات عامة مثل Classes, SysUtils وغيرها.

الوحدات التي يكتبها المبرمج

أما الوحدات التي يحتاج أن يكتبها المبرمج فهي تعتبر وحدات خاصة تلبي حاجة برنامجه الذي يقوم بتصميمه وكتابته أو البرامج المشابة، مثلاً اذا افترضنا أن المبرمج يقوم بكتابة برنامج لإدارة صيانة السيارات، فإن الوحدات التي يكتبها يمكن أن تحتوي على إجراءات خاصة بالسيارات وبرنامج الصيانة، مثل إجراء تسجيل سيارة جديدة، أو البحث عن سيارة بدلالة رقم اللوحة، أو إضافة سجل صيانة، إلخ.
ففي هذه الحالة يقوم المبرمج بكتابة وحدة أو وحدات تحتوي على دوال وإجراءات تمكنه من بناء قاعدة من الأدوات التي تسهل له بناء البرنامج الرئيسي بسرعة وبوضوح. فمثلاً إذا قام المبرمج بكتابة إجراءات تمكنه من القيام بكافة مطلبات البرنامج، فإن كتابة البرنامج الرئيسي يكون عبارة عن نداء لتلك الدوال والإجراءات بطريقة تجعل البرنامج الرئيسي أكثر وضوحاً وأقل تعقيداً.
كذلك يمكن لعدد من المبرمجين أن ينفردوا بكتابة وحدات تمثل جزئية معينة من النظام، وعند تجميعها نكون قد تحصلنا على برنامج كبير قام بكتابته عدد من المبرمجين دون أن يكون هناك تعارض في أجزائه.

وحدة التاريخ الهجري

التاريخ الهجري له أهمية كبيرة للمسلمين حيث أن كل المعاملات الشرعية المرتبطة بتاريخ أو زمن معين فإن هذا الزمن أو الفترة تكون محسوبة بالتاريخ الهجري، مثلاً زكاة المال مرتبطة بتمام الحول وهو عام هجري.
التاريخ الهجري مبني على السنة القمرية وهي تحتوي على 12 شهر، كانت تستخدم من زمن الجاهلية، لكن عمر بن الخطاب رضى الله عنه اعتمد هجرة الرسول صلى الله عليه وسلم إلى المدينة هي بداية لعهد جديد يبدأ المسلمون بحساب السنوات إستناداً له، وهو يوم 16 يوليو 622 ميلادية.

السنة القمرية فيها 354.367056 يوم، والشهر القمري فيه 29.530588 يوم.
وقد قمت بكتابة وحدة للتحويل بين السنة الهجرية والميلادية بناءً على هذه الثوابت. ويمكن أن يكون الخطأ في التحويل هو يوم واحد فقط زيادةً أو نقصاناً.
وكود الوحدة هو:

```
{
*****************************************************************************

  HejriUtils:   Hejri Calnder converter, for FreePascal and Delphi
  Author:       Motaz Abdel Azeem
  email:        motaz1@yahoo.com
  Home page:    http://motaz.freevar.com/
  License:      LGPL
  Created on:   26.Sept.2009
  Last modifie: 26.Sept.2009

*****************************************************************************
}

unit HejriUtils;

{$IFDEF FPC}
{$mode objfpc}{$H+}
{$ENDIF}

interface

uses
  Classes, SysUtils;

const
  HejriMonthsEn: array [1 .. 12] of string = ('Moharram', 'Safar', 'Rabie Awal',
    'Rabie Thani', 'Jumada Awal', 'Jumada Thani', 'Rajab', 'Shaban', 'Ramadan',
    'Shawal', 'Thi-Alqaida', 'Thi-Elhajah');

  HejriMonthsAr: array [1 .. 12] of string = ('محرم', 'صفر' , 'ربيع أول',
    'ربيع ثاني', 'جمادى الأول', 'جمادى الآخر', 'رجب', 'شعبان', 'رمضان',
    'شوال', 'ذي القعدة', 'ذي الحجة');

  HejriYearDays = 354.367056;
  HejriMonthDays = 29.530588;

  procedure DateToHejri(ADateTime: TDateTime; var Year, Month, Day: Word);
  function HejriToDate(Year, Month, Day: Word): TDateTime;
```

```
  procedure HejriDifference(Year1, Month1, Day1, Year2, Month2, Day2: Word; var
YearD, MonthD, DayD: Word);


implementation

var
  HejriStart : TDateTime;

procedure DateToHejri(ADateTime: TDateTime; var Year, Month, Day: Word);
var
  HejriY: Double;
  Days: Double;
  HejriMonth: Double;
  RDay: Double;
begin
  HejriY:= ((Trunc(ADateTime) - HejriStart - 1) / HejriYearDays);
  Days:= Frac(HejriY);
  Year:= Trunc(HejriY) + 1;
  HejriMonth:= ((Days * HejriYearDays) / HejriMonthDays);
  Month:= Trunc(HejriMonth) + 1;
  RDay:= (Frac(HejriMonth) * HejriMonthDays) + 1;
  Day:= Trunc(RDay);
end;

function HejriToDate(Year, Month, Day: Word): TDateTime;
begin
  Result:= (Year - 1) * HejriYearDays + (HejriStart - 0) +
    (Month - 1) * HejriMonthDays + Day + 1;
end;

procedure HejriDifference(Year1, Month1, Day1, Year2, Month2, Day2: Word; var
  YearD, MonthD, DayD: Word);
var
  Days1: Double;
  Days2: Double;
  DDays: Double;
  RYear, RMonth: Double;
begin
  Days1:= Year1 * HejriYearDays + (Month1 - 1) * HejriMonthDays + Day1 - 1;
  Days2:= Year2 * HejriYearDays + (Month2 - 1) * HejriMonthDays + Day2 - 1;
  DDays:= Abs(Days2 - Days1);

  RYear:= DDays / HejriYearDays;
  RMonth:= (Frac(RYear) * HejriYearDays) / HejriMonthDays;
  DayD:= Round(Frac(RMonth) * HejriMonthDays);

  YearD:= Trunc(RYear);
  MonthD:= Trunc(RMonth);

end;

initialization

  HejriStart:= EncodeDate(622, 7, 16);

end.
```

وتحتوي هذه الوحدة على الدوال والإجراءات التالية:

1. DateToHejri: وتستخدم لتحويل تاريخ ميلادي لما يقابله من التاريخ الهجري، مثلاً:

```
program Project1;

{$mode objfpc}{$H+}

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

var
  Year, Month, Day: Word;
begin
  DateToHejri(Now, Year, Month, Day);
  Writeln('Today in Hejri: ', Day, '-', HejriMonthsEn[Month],
    '-', Year);
  Readln;
end.
```

2. HejriToDate: ويستخدم للتحويل من تاريخ هجري لمايقابله بالميلادي.
3. HejriDifference: ويستخدم لمعرفة الفارق الزمني بين تاريخين هجريين.

يوجد مثال آخر للتحويل بين التاريخ الهجري والميلادي في فصل الواجهة الرسومية، واسم المثال هو المحوِّل الهجري.

تحميل الإجراءات والدوال Procedure and function Overloading

المقصود بتحميل الإجراءات والدوال هو أن تكون هناك أكثر من دالة أو إجراء مشترك في الإسم لكن مع تغيير المدخلات من حيث العدد أو النوع. مثلاً لو أردنا أن نكتب دالة تقوم بجمع عددين صحيحين أو حقيقيين، بحيث عندما يقوم المبرمج بنداء الدالة Sum مع أي من نوع المتغيرات السابقة يقوم البرنامج بإختيار الدالة المناسبة:

```
program sum;

{$mode objfpc}{$H+}

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


function Sum(x, y: Integer): Integer; overload;
begin
  Result:= x + y;
end;

function Sum(x, y: Double): Double; overload;
begin
  Result:= x + y;
end;

var
  j, i: Integer;
  h, g: Double;
begin
  j:= 15;
  i:= 20;
  Writeln(J, ' + ', I, ' = ', Sum(j, i));

  h:= 2.5;
  g:= 7.12;
  Writeln(H, ' + ', G, ' = ', Sum(h, g));

  Write('Press enter key to close');
  Readln;

end.
```

نلاحظ أننا قمنا بكتابة الكلمة المحجوزة **overload** لكتابة دالتين بنفس الإسم مع إختلاف نوع المتغير وناتج الدالة.

القيم الإفتراضية للمدخلات default parameters

يمكن أن نضع قيم إفتراضية في مدخلات الدوال أو الإجراءات. وفي هذه الحالة يمكن تجاهل هذه المدخلات عند النداء أو إستخدامها كما في المثال التالي:

```
program defaultparam;

{$mode objfpc}{$H+}

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


function SumAll(a, b: Integer; c: Integer = 0;
  d: Integer = 0): Integer;
begin
  result:= a + b + c + d;
end;

begin
  Writeln(SumAll(1, 2));
  Writeln(SumAll(1, 2, 3));
  Writeln(SumAll(1, 2, 3, 4));
  Write('Press enter key to close');
  Readln;
end.
```

نجد أن كل النداءات السابقة صحيحة، وفي حالة عدم إرسال متغير معين يتم إسناد القيمة الإفتراضية له، وفي هذه الحالة القيمة صفر للمتغيرات c و d.
وفي حالة تجاهل المتغيرات وعدم إرسالها، يجب التقيّد بالترتيب، حيث يمكن تجاهل d فقط، أو تجاهل c و d، لكن لايمكن تجاهل المغير أو المُدخل c وإرسال d. فإذا تجاهلنا إرسال قيمة c لابد من تجاهل d تبعاً لذلك. لذلك على المبرمج أن يضع المدخلات الغير مهمة في نهاية تعريف الدالة والأكثر أهمية في بدايتها. أي أن الأقل أهمية في المثال السابق كان المتغير d.

ترتيب البيانات sorting

ترتيب البيانات هو موضوع يقع ضمن مفهوم هيكلية البيانات Data Structure، وقد قمنا بذكره في هذا الفصل كمثال للبرامج التي تعتمد البرمجة الهيكلية والإجراءات لتحقيق ترتيب البيانات.
ترتيب البيانات نحتاجه مع المصفوفات، فمثلاً إذا افترضنا أن لدينا مصفوفة فيها أرقام، ونريد ترتيب هذه الأرقام تصاعدياً أو تنازلياً، في هذه الحالة يمكننا استخدام عدة طرق لتحقيق هذا الهدف.

خوارزمية ترتيب الفُقّاعة bubble sort

وهي من أسهل طرق الترتيب، حيث يقوم إجراء الترتيب بمقارنة العُنصر الأول في المصفوفة مع العنصر الثاني، ففي حالة الترتيب التصاعدي ،إذا وُجد أن العنصر الأول ذو قيمة أكبر من العنصر الثاني، قام البرنامج بتبديل العناصر، ثم يقوم بمقارنة العنصر الثاني مع الثالث إلى نهاية المصفوفة. ثم يتم تكرار هذه العملية عدة مرات حتى يتم الترتيب الكامل للمصفوفة.

إذا افترضنا أن لدينا 6 عناصر في المصفوفة تم إدخالها بهذا الشكل:

```
7
10
2
5
6
3
```

نجد أن الإجراء يحتاج لأكثر من دورة لإتمام العملية. حيث أن في كل دورة تتم مقارنة العنصر الأول مع الثاني وإبدالهما إذا كان الأول أكبر من الثاني، ثم الثاني مع الثالث، والثالث مع الرابع، ثم الرابع مع الخامس، ثم الخامس مع السادس. فإذا لم يتم الترتيب الكامل فسوف نحتاج لدورة أخرى، ثم ثالثة ورابعة إلى أن يتم الترتيب الكامل.

في الدورة الأولى تصبح المتغيرات في المصفوفة كالآتي:

```
7
2
5
6
3
10
```

وفي الدورة الثانية:

```
2
5
6
3
7
10
```

وفي الدورة الثالثة:

```
2
5
3
6
7
10
```

وفي الدورة الرابعة:

```
2
3
5
6
7
10
```

نلاحظ أنه بنهاية الدورة الرابعة تم الترتيب الكامل للأرقام، لكن ليس من السهولة معرفة أن الترتيب قد تم إلا بإضافة دورة خامسة، وفي هذه الدورة نجد أن الإجراء لم يقم بإي تبديل، وبهذه الطريقة نعرف أن البيانات قد تم ترتيبها.
وجاء إسم الترتيب الفُقّاعي بسبب أن القيم الأقل تطفو للأعلى في كل دورة.

كود برنامج ترتيب الفقاعة:

```
program BubbleSortProj;

{$mode objfpc}{$H+}

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

procedure BubbleSort(var X: array of Integer);
var
  Temp: Integer;
  i: Integer;
  Changed: Boolean;
begin
  repeat  // Outer loop
    Changed:= False;
    for i:= 0 to High(X) - 1 do // Inner loop
      if X[i] > X[i + 1] then
      begin
        Temp:= X[i];
        X[i]:= X[i + 1];
        X[i + 1]:= Temp;
        Changed:= True;
      end;
  until not Changed;
end;
```

```
var
  Numbers: array [0 .. 9] of Integer;
  i: Integer;
begin
  Writeln('Please input 10 random numbers');
  for i:= 0 to High(Numbers) do
  begin
    Write('#', i + 1, ': ');
    Readln(Numbers[i]);
  end;

  BubbleSort(Numbers);
  Writeln;
  Writeln('Numbers after sort: ');

  for i:= 0 to High(Numbers) do
  begin
    Writeln(Numbers[i]);
  end;
  Write('Press enter key to close');
  Readln;
end.
```

يمكن تحويل الترتيب التنازلي إلى ترتيب تصاعدي (الأكبر أولاً ثم الأصغر) وذلك بتحويل معامل المقارنة من أكبر من إلى أصغر من كالتالي:

```
   if X[i] < X[i + 1] then
```

وفي المثال التالي نقوم بالترتيب التصاعدي لدرجات طلاب لنعرف الأول ثم الثاني إلى الأخير.
وفي هذا المثال استخدمنا مصفوفة سجلات لتسجيل إسم الطالب ودرجته:

برنامج ترتيب درجات الطلاب :

```
program smSort;   // Stuents mark sort

{$mode objfpc}{$H+}

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

type
  TStudent = record
    StudentName: string;
    Mark: Byte;
  end;
```

```
procedure BubbleSort(var X: array of TStudent);
var
  Temp: TStudent;
  i: Integer;
  Changed: Boolean;
begin
  repeat
    Changed:= False;
    for i:= 0 to High(X) - 1 do
      if X[i].Mark < X[i + 1].Mark then
      begin
        Temp:= X[i];
        X[i]:= X[i + 1];
        X[i + 1]:= Temp;
        Changed:= True;
      end;
  until not Changed;
end;

var
  Students: array [0 .. 9] of TStudent;
  i: Integer;

begin
  Writeln('Please input 10 student names and marks');
  for i:= 0 to High(Students) do
  begin
    Write('Student #', i + 1, ' name : ');
    Readln(Students[i].StudentName);

    Write('Student #', i + 1, ' mark : ');
    Readln(Students[i].Mark);
    Writeln;
  end;

  BubbleSort(Students);
  Writeln;
  Writeln('Marks after Buble sort: ');
  Writeln('----------------------');

  for i:= 0 to High(Students) do
  begin
    Writeln('# ', i + 1, '  ', Students[i].StudentName,
      ' with mark (', Students[i].Mark, ')');
  end;
  Write('Press enter key to close');
  Readln;
end.
```

طريقة ترتيب الفقاعة تمتاز بالبساطة، حيث أن المبرمج يمكن أن يحفظها ويكتب الكود كل مرة بدون الرجوع لأي مرجع. وهي مناسبة للمصفوفات الصغيرة والشبه مرتبة، أما إذا كانت المصفوفة تحتوي على عناصر كثيرة (عشرات الآلاف مثلاً) وكانت عشوائية تماماً ففي هذه الحالة سوف يستغرق هذا النوع من الترتيب وقتاً ويجب اللجوء إلى نوع آخر من الترتيب كما في الأمثلة اللاحقة.

نلاحظ أن خوارزمية ترتيب الفقاعة تعتمد على دورتين، دورة خارجية تستخدم repeat until وعدد لفاتها غير محددة، معتمدة على شكل البيانات، فإذا كانت البيانات مرتبة فإن عدد اللفات تكون مرة واحدة

فقط، أما إذا كانت عشوائية تماماً فإن عدد اللفات تكون بعدد عناصر المصفوفة. والدورة الداخلية بإستخدام for loop تكون لفاتها دائماً بعدد عناصر المصفوفة ناقص واحد.
فإذا إفترضنا أن لدينا 1000 عنصر في المصفوفة وكانت المصفوفة مُرتبة فإن عدد اللفات الكلي هو لفة واحدة خارجية مضروبة في 999، أي الحصيلة 999 لفة.
وإذا كانت نفس المصفوفة عشوائية تماماً فإن عدد اللفات الخارجية تكون 1000 مضروبة في 999 لفة داخلية، فتكون الحصيلة 999000 لفة. فكلما زاد عدد عناصر المصفوفة وزادت عشوائيتها فإن كفاءة هذا النوع من الترتيب يسوء.

خوارزمية الترتيب الإختياري Selection Sort

هذه الطريقة أشبه بطريقة الفُقّاعة، إلا أنها أسرع مع البيانات الكبيرة، حيث أنها تحتوي على دورتين، دورة خارجية مشابهة لدورة repeat until في المثال السابق، وهي تدور بعدد عناصر المصفوفة ناقص واحد، أما الدورة الداخلية فتقل في كل دورة بمقدار واحد، إلى أن تصل إلى دورتين.

```
program SelectionSort;

{$mode objfpc}{$H+}

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


procedure SelectSort(var X: array of Integer);
var
  i: Integer;
  j: Integer;
  SmallPos: Integer;
  Smallest: Integer;
begin
  for i:= 0 to High(X) -1 do // Outer loop
  begin
    SmallPos:= i;
    Smallest:= X[SmallPos];
    for j:= i + 1 to High(X) do // Inner loop
      if X[j] < Smallest then
      begin
        SmallPos:= j;
        Smallest:= X[SmallPos];
      end;
      X[SmallPos]:= X[i];
      X[i]:= Smallest;
  end;
end;
```

```
// Main application

var
  Numbers: array [0 .. 9] of Integer;
  i: Integer;

begin
  Writeln('Please input 10 random numbers');
  for i:= 0 to High(Numbers) do
  begin
    Write('#', i + 1, ': ');
    Readln(Numbers[i]);
  end;

  SelectSort(Numbers);
  Writeln;
  Writeln('Numbers after Selection sort: ');

  for i:= 0 to High(Numbers) do
  begin
    Writeln(Numbers[i]);
  end;
  Write('Press enter key to close');
  Readln;
end.
```

بالرغم من أن خوارزمية الترتيب الإختياري أسرع في حالة البيانات الكبيرة، إلا أنها لاترتبط بمدى ترتيب أو عشوائية المصفوفة، ففي كل الحالات نجد أن سرعتها شبه ثابتة، حيث يحدث أحياناً تبديل لنفس العنصر مع نفسه بعد فراغ الحلقة الداخلية.

خوارزمية الترتيب Shell

تمتاز هذه الخوارزمية بالسرعة العالية مع البيانات الكبيرة، وسلوكها مشابهة لخوارزمية الفُقّاعة في حالة البيانات المرتبة أو الشبه مرتبة، إلا أنها أكثر تعقيداً من الخوارزميتين السابقتين.
وسميت بهذا الإسم نسبة لمخترعها Donald Shell.

```
program ShellSort;

{$mode objfpc}{$H+}

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


procedure ShellS(var X: array of Integer);
var
```

```
  Done: Boolean;
  Jump, j, i: Integer;
  Temp: Integer;
begin
  Jump:= High(X);
  while (Jump > 0) do // Outer loop
  begin
    Jump:= Jump div 2;
    repeat          // Intermediate loop
      Done:= True;
      for j:= 0 to High(X) - Jump do // Inner loop
      begin
        i:= j + Jump;
        if X[j] > X[i] then // Swap
        begin
          Temp:= X[i];
          X[i]:= X[j];
          X[j]:= Temp;
          Done:= False;
        end;

      end; // end of inner loop
    until Done; // end of intermediate loop
  end; // end of outer loop
end;

var
  Numbers: array [0 .. 9] of Integer;
  i: Integer;

begin
  Writeln('Please input 10 random numbers');
  for i:= 0 to High(Numbers) do
  begin
    Write('#', i + 1, ': ');
    Readln(Numbers[i]);
  end;

  ShellS(Numbers);
  Writeln;
  Writeln('Numbers after Shell sort: ');

  for i:= 0 to High(Numbers) do
  begin
    Writeln(Numbers[i]);
  end;
  Write('Press enter key to close');
  Readln;
end.
```

ترتيب المقاطع

يمكن ترتيب المقاطع بنفس طريقة ترتيب الأسماء، حيث أن المقاطع يتم ترتيبها حسب الأحرف، فمثلاً الحرف A قيمته أقل من الحرف B. حيث أن قيمة الحرف A في الذاكرة هو عبارة عن الرقم 65 والحرف B قيمته 66. في المثال التالي أعدنا كتابة برنامج ترتيب درجات الطلاب، لكن هذه المرة يتم الترتيب هجائياً حسب الإسم وليس الدرجة:

برنامج ترتيب الطلاب بالأسماء

```
program smSort;   // Stuents mark sort by name

{$mode objfpc}{$H+}

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

type
  TStudent = record
    StudentName: string;
    Mark: Byte;
  end;


procedure BubbleSort(var X: array of TStudent);
var
  Temp: TStudent;
  i: Integer;
  Changed: Boolean;
begin
  repeat
    Changed:= False;
    for i:= 0 to High(X) - 1 do
      if X[i].StudentName > X[i + 1].StudentName then
      begin
        Temp:= X[i];
        X[i]:= X[i + 1];
        X[i + 1]:= Temp;
        Changed:= True;
      end;
  until not Changed;
end;

var
  Students: array [0 .. 9] of TStudent;
  i: Integer;

begin
  Writeln('Please input 10 student names and marks');
  for i:= 0 to High(Students) do
```

```
  begin
    Write('Student #', i + 1, ' name : ');
    Readln(Students[i].StudentName);

    Write('Student #', i + 1, ' mark : ');
    Readln(Students[i].Mark);
    Writeln;
  end;

  BubbleSort(Students);
  Writeln;
  Writeln('Marks after Bubble sort: ');
  Writeln('----------------------');

  for i:= 0 to High(Students) do
  begin
    Writeln('# ', i + 1, '  ', Students[i].StudentName,
     ' with mark (', Students[i].Mark, ')');
  end;
  Write('Press enterkey to close');
  Readln;
end.
```

مقارنة خوارزميات الترتيب

في هذا البرنامج سوف نقوم بإدخال أرقام عشوائية في مصفوفة كبيرة ونجري عليها أنواع الترتيب الثلاث السابقة ونقارن الزمن الذي تأخذه كل خوارزمية:

```
program SortComparison;

{$mode objfpc}{$H+}

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

//  Bubble sort

procedure BubbleSort(var X: array of Integer);
var
  Temp: Integer;
  i: Integer;
  Changed: Boolean;
begin
  repeat
    Changed:= False;
    for i:= 0 to High(X) - 1 do
      if X[i] > X[i + 1] then
      begin
        Temp:= X[i];
```

```
        X[i]:= X[i + 1];
        X[i + 1]:= Temp;
        Changed:= True;
      end;
  until not Changed;
end;


// Selection Sort

procedure SelectionSort(var X: array of Integer);
var
  i: Integer;
  j: Integer;
  SmallPos: Integer;
  Smallest: Integer;
begin
  for i:= 0 to High(X) -1 do // Outer loop
  begin
    SmallPos:= i;
    Smallest:= X[SmallPos];
    for j:= i + 1 to High(X) do // Inner loop
      if X[j] < Smallest then
      begin
        SmallPos:= j;
        Smallest:= X[SmallPos];
      end;
      X[SmallPos]:= X[i];
      X[i]:= Smallest;
  end;
end;

// Shell Sort

procedure ShellSort(var X: array of Integer);
var
  Done: Boolean;
  Jump, j, i: Integer;
  Temp: Integer;
begin
  Jump:= High(X);
  while (Jump > 0) do // Outer loop
  begin
    Jump:= Jump div 2;
    repeat         // Intermediate loop
      Done:= True;
      for j:= 0 to High(X) - Jump do // Inner loop
      begin
        i:= j + Jump;
        if X[j] > X[i] then // Swap
        begin
          Temp:= X[i];
          X[i]:= X[j];
          X[j]:= Temp;
          Done:= False;
        end;
```

```
      end; // inner loop
    until Done; // innermediate loop end
  end; // outer loop end
end;

// Randomize Data

procedure RandomizeData(var X: array of Integer);
var
  i: Integer;
begin
  Randomize;
  for i:= 0 to High(X) do
    X[i]:= Random(10000000);
end;

var
  Numbers: array [0 .. 59999] of Integer;
  i: Integer;
  StartTime: TDateTime;

begin

  Writeln('----------------- Full random data');
  RandomizeData(Numbers);
  StartTime:= Now;
  Writeln('Sorting.. Bubble');
  BubbleSort(Numbers);
  Writeln('Bubble sort tooks (mm:ss): ',
    FormatDateTime('nn:ss', Now - StartTime));
  Writeln;

  RandomizeData(Numbers);
  Writeln('Sorting.. Selection');
  StartTime:= Now;
  SelectionSort(Numbers);
  Writeln('Selection sort tooks (mm:ss): ',
    FormatDateTime('nn:ss', Now - StartTime));
  Writeln;

  RandomizeData(Numbers);
  Writeln('Sorting.. Shell');
  StartTime:= Now;
  ShellSort(Numbers);
  Writeln('Shell sort tooks (mm:ss): ',
   FormatDateTime('nn:ss', Now - StartTime));

  Writeln;
  Writeln('----------------- Nearly sorted data');
  Numbers[0]:= Random(10000);
  Numbers[High(Numbers)]:= Random(10000);
  StartTime:= Now;
  Writeln('Sorting.. Bubble');
  BubbleSort(Numbers);
  Writeln('Bubble sort tooks (mm:ss): ',
    FormatDateTime('nn:ss', Now - StartTime));
  Writeln;
```

```
  Numbers[0]:= Random(10000);
  Numbers[High(Numbers)]:= Random(10000);
  Writeln('Sorting.. Selection');
  StartTime:= Now;
  SelectionSort(Numbers);
  Writeln('Selection sort tooks (mm:ss): ',
    FormatDateTime('nn:ss', Now - StartTime));
  Writeln;

  Numbers[0]:= Random(10000);
  Numbers[High(Numbers)]:= Random(10000);
  Writeln('Sorting.. Shell');
  StartTime:= Now;
  ShellSort(Numbers);
  Writeln('Shell sort tooks (mm:ss): ',
    FormatDateTime('nn:ss', Now - StartTime));

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

الفصل الثالث

الواجهة الرسومية Graphical User Interface

مقدمة

الواجهة الرسومية أو واجهة المستخدم الرسومية هي بديل حديث لواجهة سطر الأوامر command line، أو الواجهة النصية text interface، وهي أكثر سهولة للمستخدم، وتُستخدم فيها إمكانات جديدة مثل الماوس، الأزرار، مربعات النصوص، وإلى ماذلك مما اعتاده المستخدم الآن. وكمثال للبرامج ذات الواجهة الرسومية في نظام لينكس: متصفح الإنترنت فيرفوكس، برامج المكتب Office ، الألعاب وغيرها. فبرامج الواجهة الرسومية أصبحت تشكل البنية الأساسية في كل أنظمة الحاسوب، مثل الأنظمة المحاسبية ، وبرامج التصميم والألعاب التي تحتوي على مكتبات متقدمة ومعقدة للواجهة الرسومية،كذلك فإن أجزاء أنظمة التشغيل مثل سطح المكتب Gnome و KDE هي عبارة عن مجموعة من برامج ذات واجهات رسومية.
لكتابة برامج ذات واجهة رسومية يجب استخدام احدى المكتبات المشهورة المتوفرة في هذا المجال مثل:

GTK+ library المستخدمة في نظام لينكس وبعض واجهات الهواتف النقالة
QT library المستخدمة في نظام لينكس ووندوز وماكنتوش
Win32/64 GDI المستخدمة في نظام وندوز

أو يمكن استخدام أداة تطوير متكاملة تستخدم احدى هذه المكتبات، مثل الازاراس. حيث نجد أن اللازاراس تسخدم GTK في نظام لينكس، و Win32/64 GDI في نظام ويندوز. ولايحتاج المبرمج أن يدخل في تفاصيل هذه المكتبات حيث أن لازاراس توفر طريقة ذات شفافية عالية في تصميم البرامج الرسومية، فماعلى المبرمج إلا تصميم هذه البرامج في احدى بيئات التشغيل ثم إعادة ترجمتها وربطها في بيئات التشغيل الأخرى ليحصل على نفس النسخة من البرنامج الأصلي يعمل في كافة أنظمة التشغيل. لكن احياناً يحتاج المبرمج أن يقوم بإنزال مكتبة GTK مثلاً في حال فشل البرنامج ذو الواجهة الرسومية من أن يعمل في حال عدم توفر هذه المكتبة مثلاً.

دعم اللغة العربية

في الوندوز لاتوجد مشكلة في دعم اللغة العربية واللغات العالمية عموماً، أما في نظام لينكس فهو يعتمد على إصدارة اللازاراس، فالإصدارة 0.9.26 والأقل منها لاتدعم اللغة العربية لأنها تستخدم مكتبة GTK+ 1 والذي لايدعم بما يعرف بالـ Unicode. و ابتداءً من النسخة رقم 0.9.27 فهي تدعم اللغة العربية لدعمها للـ Unicode لأنها مبنية على GTK+ 2. لذلك في نظام لينكس لابد من التأكد أن رقم نسخة لازاراس هو أعلى من الإصدارة 0.9.27

البرنامج ذو الواجهة الرسومية الأول

لعمل برنامج جديد ذو واجهة رسومية علينا إختيار :

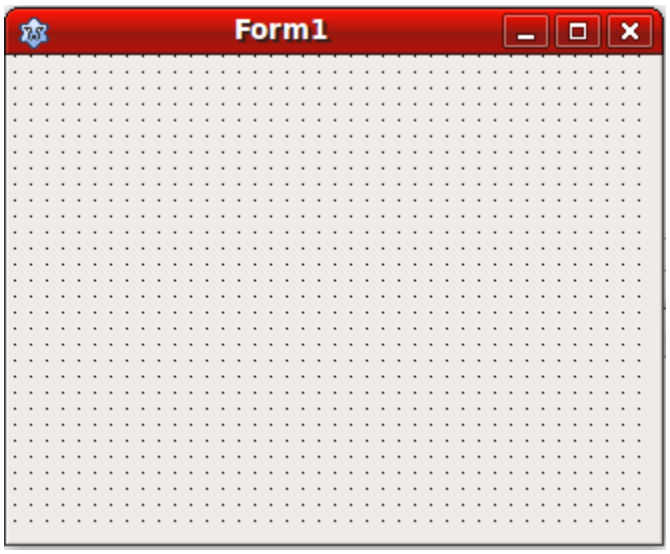
Project/New Project/Application

بعد ذلك نقوم بحفظه المشروع بواسطة:

File/Save All

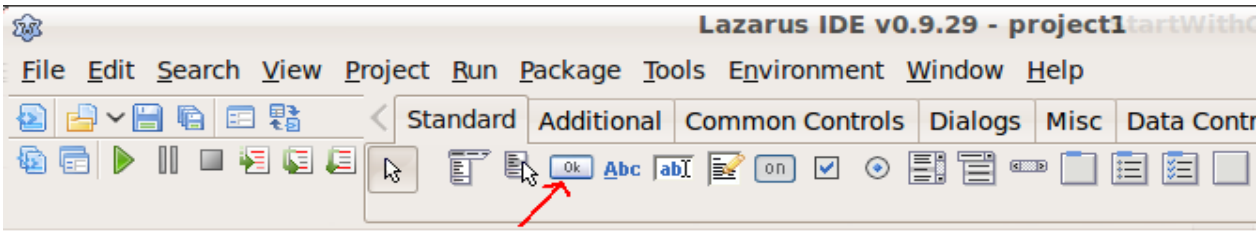
ثم نقوم بإنشاء مجلد جديد مثلاً firstgui لنقوم فيه بحفظ المشروع بكل متعلقاته.
بعد ذلك نقوم بحفظ الوحدة الرئيسية في المجلد السابق، ونسميه main.pas
ثم نقوم بحفظ ملف المشروع في نفس المجلد ونسميه firstgui.lpi

في الوحدة الرئيسية يمكننا الضغط على المفتاح F12 ليظهر لنا الفورم التالي:

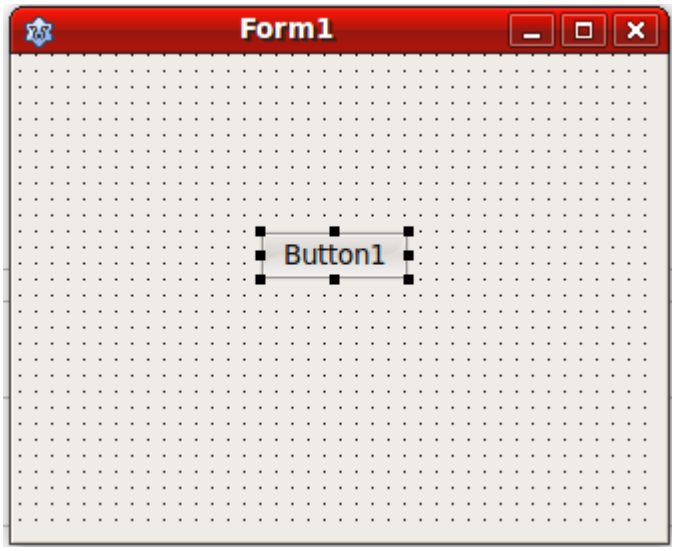


إذا قمنا بتشغيل البرنامج سوف نحصل على الشكل التالي:

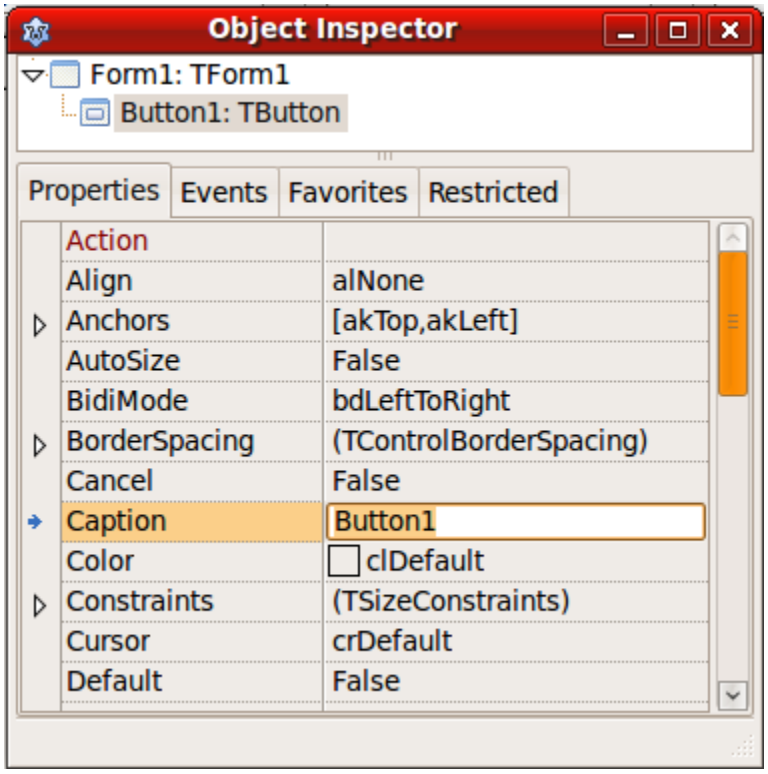
بعد ذلك نقوم بإغلاق البرنامج ونرجع لحالة التصميم.
ثم نقوم بإضافة زر Button من صفحة المكونات التالية:



نقوم بوضع هذا الزر في وسط الفورم السابق كالتالي:



بعد ذلك نقوم نذهب إلى صفحة الخصائص الموجودة في الـ Object Inspector. فإذا لم تكن الـ object Inspector موجودة يمكن إظهارها عن طريق Window/Object Inspector في القائمة الرئيسية، أو الوقوف على الفورم أو المكون ثم الضغط على المفتاح F11 لتظهر لنا الخصائص التالية:



لابد من التأكد من أننا نؤشر على الزر، ويمكن معرفة ذلك بقراءة القيمة المظللة في الـ Object Inspector والتي يجب أن تحتوي على:

Button1: Tbutton

في صفحة الخصائص Properties نقوم بتغيير قيمة الـ Caption من Button1 إلى كلمة Click.
بعد ذلك نذهب إلى صفحة الأحداث Events في نفس الـ Object Inspector ونقوم بالنقر المزدوج على قيمة الحدث OnClick فنحصل على الكود التالي:

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

ثم نقوم بكتابة السطر التالي بين begin end ونقوم بتشغيل البرنامج:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('Hello world, this is my first GUI application');
end;
```

بعد تنفيذ البرنامج نقوم بالضغط على الزر المكتوب عليه Click فنحصل على الرسالة أعلاه.
بهذا نكون قد كتبنا أول برنامج ذو واجهة رسومية.

في الدليل firstgui سوف نجد الملف التنفيذي firstgui إذا كنا في لينكس أو firstgui.exe إذا كنا في بيئة

وندوز، وهي ملفات تنفيذية يمكن نقلها وتشغيلها في أي حاسوب آخر به نفس نظام التشغيل.

في البرنامج السابق نريد الوقوف على عدد من النقاط المهمة وهي:

1. **البرنامج الرئيسي**: وهو الملف الذي قمنا بحفظه بالإسم: firstgui.lpi، وهو الملف الرئيسي الذي يمثل البرنامج. ويمكن فتحه عن طريق:

Project/Source

أو بالضغط على CTRL-F12 ثم اختيار firstgui. وسوف نجد هذا الكود الذي قام لازاراس بكتابته تلقائياً:

```
program firstgui;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Interfaces, // this includes the LCL widgetset
  Forms
  { you can add units after this }, main;

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

ومن النادر أن يقوم المبرمج بتعديل هذا الملف، فمعظم تصميم وكتابة الكود تكون في الوحدات الإضافية المصاحبة لهذا المشروع. كما ونلاحظ أن الوحدة الرئيسية main قد تم إضافتها تلقائياً لهذا الملف.

2. **الوحدة الرئيسية main unit** وهي تحتوي على الفورم الرئيسي الذي يظهر عند تشغيل البرنامج والكود المصاحب له. ولعمل برنامج ذو واجهة رسومية لابد من وجود وحدة بهذا الشكل تحتوي على فورم.

الكود الذي تحتويه هذه الوحدة بعد الإضافات التي قمنا بها بعد إضافة الزر هو كالآتي:

```
unit main;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs,
  StdCtrls;



type
```

```
  { TForm1 }

  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  Form1: TForm1;

implementation

{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('Hello world, this is my first GUI application');
end;

initialization
  {$I main.lrs}

end.
```

نجد أنه في بداية هذه الوحدة تم تعريف TForm1 على أنه من النوع Class وهو أشبه بالسجل الذي ذكرناه في دروس سابقة. وهو مايعرف بالبرمجة الكائنية والتي تعتبر درس متقدم سوف نذكره في الفصل القادم إن شاء الله. وداخل هذا الفورم يوجد زر من النوع TButton واسمه Button1.
يمكن فتح هذه الوحدة بواسطة CTRL-F12 ثم اختيار اسمها وهي في هذا المثال main.

3. **صفحة الخصائص Object Inspector/Properties:** وفي هذه الصفحة تظهر خصائص الكائنات أو المكونات مثل الأزرار والفورم. فمثلاً نجد أن للزر بعض الخصائص التي يمكن للمبرمج تغييرها وتؤثر مباشرة على شكل الزر أو بعض سلوكه مثل: Caption, Top, Left, Width, Height. كذلك الفورم له بعض الخصائص مثل: Color, Visible إلخ. وهي كأنها حقول ومتغيرات داخل السجل Form1 أو Button1. وهي في الحقيقة كائنات Objects وليس سجلات Records أنما شبهناها بالسجلات لتسهيل الفهم.

4. **صفحة الأحداث Object Inspector/Events:** وتحتوي هذه الصفحة على الأحداث التي يمكن استقبالها على الكائن المعين، مثلاً الزر يمكن أن يستقبل بعض الأحداث مثل: OnClick, OnMouseMove، أما الفورم فيستقبل أحداث مثل: OnCreate, OnClose, OnActivate ومن أسمائها يظهر جلياً متى يتم نداء هذا الحدث، فمثلاً الحدث OnMouseMove يتم نداءه عندما يمر الماوس فوق هذا الكائن. وعند النقر المزدوج على هذه الأحداث تظهر لنا صفحة الكود التي نقوم بكتابة ماسوف يقوم البرنامج بتنفيذه عندما يحدث هذا الحدث، مثلاً الحدث الذي قمنا باختياره عند النقر في الزر هو:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('Hello world, this is my first GUI application');
end;
```

وهذا الكود أو الإجراء يعرف بإجراء الحدث Event Handler ويتم استدعاء هذا الإجراء تلقائياً عند النقر في هذا الزر في حالة التشغيل.

البرنامج الثاني: برنامج إدخال الإسم

نقوم بإنشاء برنامج جديد بنفس الطريقة السابقة، ونقوم بإنشاء مجلد جديد لهذا المشروع نسميه inputform حيث نحفظ فيه الوحدة الرئيسية بإسم main والبرنامج بإسم inputform.
ثم نقوم بإضافة مكونات أو كائنات وهي :

2 Lables
Edit box
Button

ونقوم بتغيير الخصائص التالية في هذه المكونات:

Form1:
Name: fmMain
Caption: Input form application

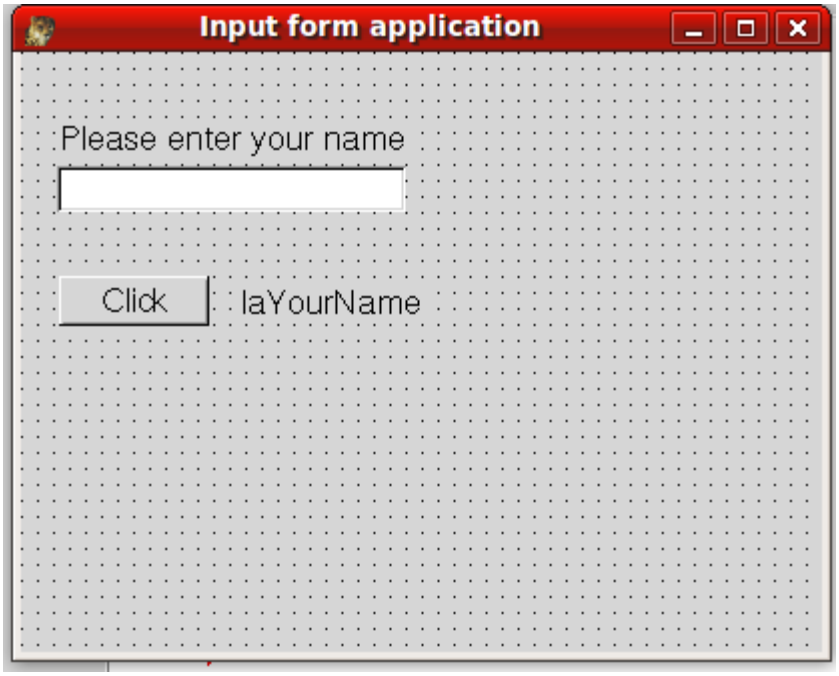
Label1:
Caption: Please enter your name

Label2:
Name: laYourName

Edit1:
Name: edName
Text:

Button1:
Caption: Click

فيصبح لدينا الشكل التالي



في الحدث OnClick للزر نضع هذا الكود:

```
procedure TfmMain.Button1Click(Sender: TObject);
begin
  laYourName.Caption:= 'Hello ' + edName.Text;
end;
```

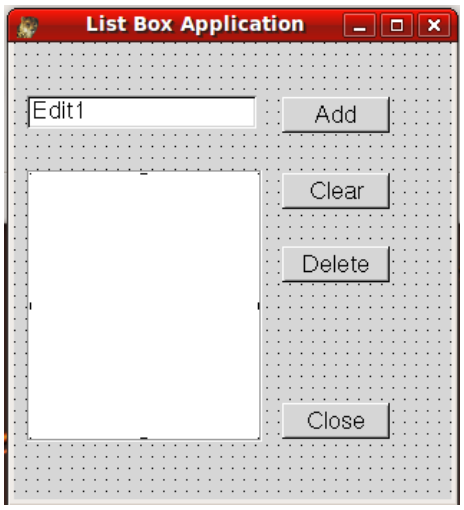
بعد ذلك نقوم بتشغيل البرنامج وكاتبة الإسم في مربع النص، ثم الضغط على الزر Click ونشاهد تنفيذ البرنامج.

في البرنامج السابق نلاحظ أننا استخدمنا الحقل Text الموجود في الكائن edName لقراءة ما أدخله المستخدم من بيانات. وكذلك استخدمنا الحقل Caption الموجود في المكون laYourName لكتابة إسم المستخدم.

برنامج الـ ListBox

نقوم بإنشاء برنامج جديد، ونضع فيه أربع أزرار ومربع نص و ListBox كما في الشكل أدناه. ونقوم بتغيير الـ Caption بالنسبة للأزرار كما هو موضح، كذلك نقوم بتغيير أسماء هذه الإزرار إلى:

btAdd, btClear, btDelete, btClose



الكود المصاحب لهذه الأزرار هو:

```
procedure TForm1.btAddClick(Sender: TObject);
begin
  ListBox1.Items.Add(Edit1.Text);
end;

procedure TForm1.btClearClick(Sender: TObject);
begin
  ListBox1.Clear;
end;

procedure TForm1.btDeleteClick(Sender: TObject);
var
  Index: Integer;
begin
  Index:= ListBox1.ItemIndex;
  if Index <> -1 then
    ListBox1.Items.Delete(Index);
end;

procedure TForm1.btCloseClick(Sender: TObject);
begin
  Close;
end;
```

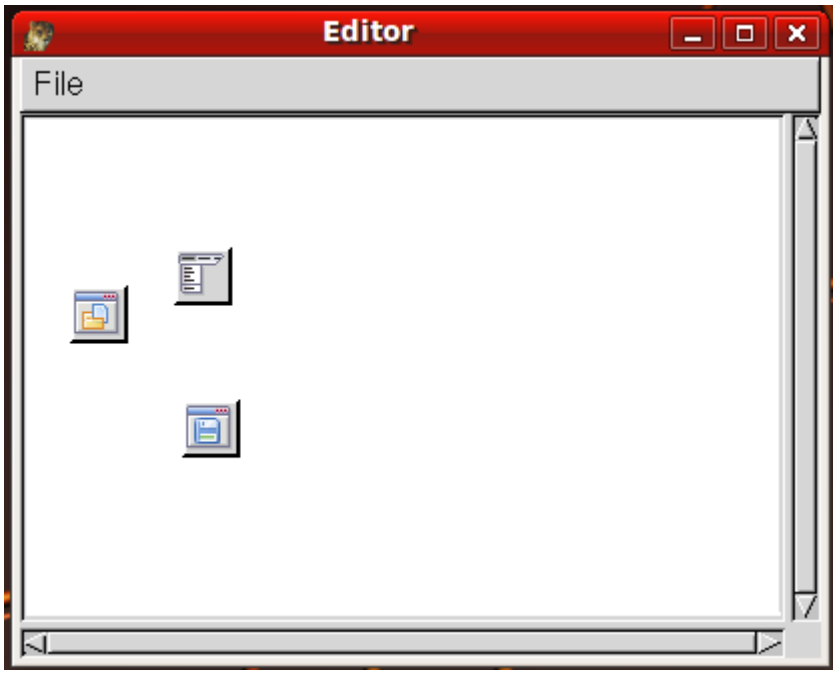
برنامج محرر النصوص Text Editor

نقوم بإنشاء برنامج جديد عليه المكونات التالية:

- قائمة رئيسية TMainMenu
- مذكرة TMemo ونقوم بتغيير قيمة الـ Align إلى alClient و الـ ScrollBars إلى ssBoth
- TOpenDialog من صفحة Dialogs
- TSaveDialog

ثم نقوم بالنقر المزدوج على القائمة الرئيسية MainMenu1 ونقوم بإضافة القائمة File وتحتها قائمة فرعية SubMenu فيها الخيارات التالية: Open File, Save File, Close

فيكون شكل البرنامج كالآتي:



نقوم بكتابة الكود التالي:

في الخيار Open File

```
if OpenDialog1.Execute then
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

وفي Save File

```
if SaveDialog1.Execute then
  Memo1.Lines.SaveToFile(SaveDialog1.FileName);
```

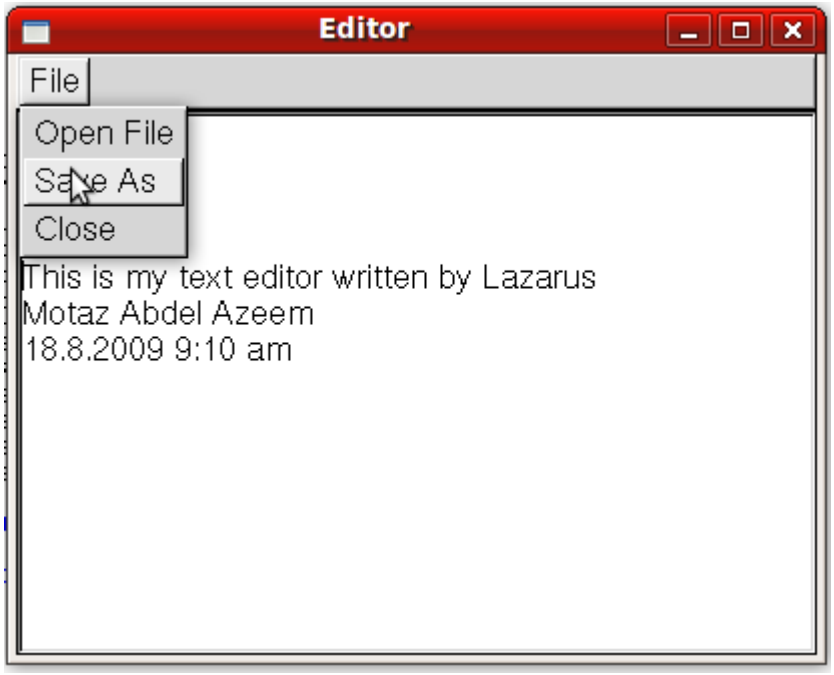
وفي Close

```
Close;
```

بعد تشغيل البرنامج يمكن كتابة نص ثم حفظه، أو فتح ملف نصي موجود على القرص.

مثلاً يمكن فتح ملف ينتهي بالإمتداد .pas فهو يعتبر ملف نصي.

وهذا هو شكل البرنامج بعد التنفيذ:



برنامج الأخبار

هذه المرة نقوم بكتابة برنامج لتسجيل عناوين الأخبار يكون له واجهة رسومية:

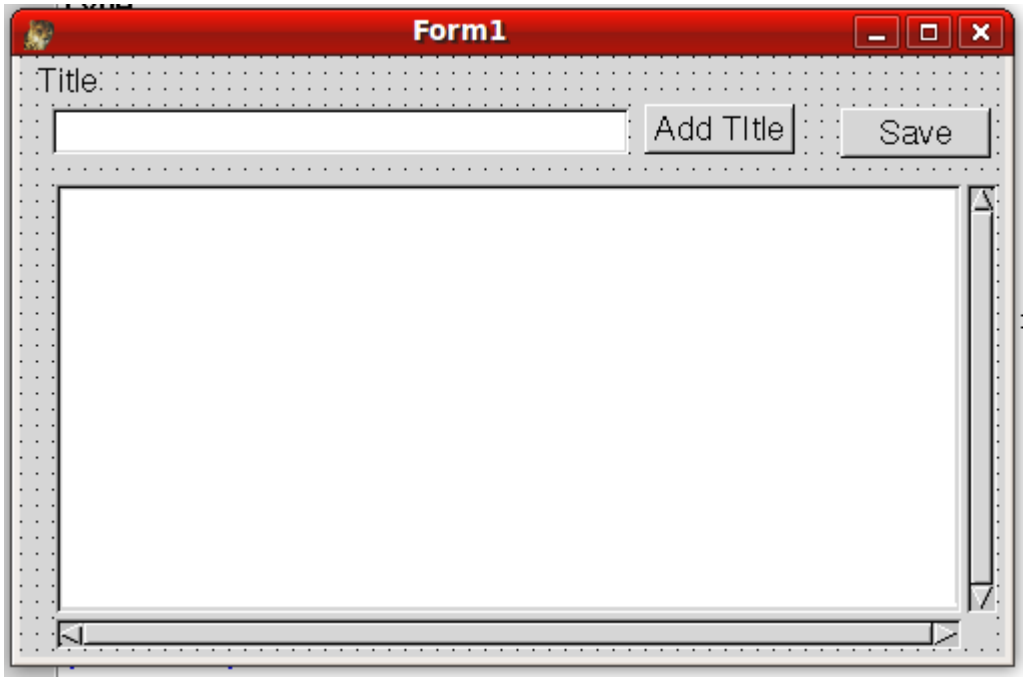
- قم بإنشاء برنامج جديد وسمه gnews
- قم بإضافة زرين من نوع TButton
- قم بإضافة مربع نص TEdit
- قم بإضافة مذكرة TMemo

وخواصهم هي كالتالي:

Button1
Caption: Add Title
Button2
Caption: Save Anchors: Left=False, Right=True
Edit1:
Text=

Memo1
ScrollBars: ssBoth
ReadOnly: True
Anchors: Top=True, Left=Ture, Right=True, Bottom=Ture

ويكون بالشكل التالي:



أما كود Add Title فهو:

```
Memo1.Lines.Insert(0,
  FormatDateTime('yyyy-mm-dd hh:nn', Now) + ': ' + Edit1.Text);
```

وكود Save

```
Memo1.Lines.SaveToFile('news.txt');
```

والكود بالنسبة للفورم في حالة إغلاقه: Tform OnClose event

```
Memo1.Lines.SaveToFile('news.txt');
```

وكود الفورم في حالة إنشائه: Tform OnCreate event

```
if FileExists('news.txt') then
  Memo1.Lines.LoadFromFile('news.txt');
```

برنامج الفورم الثاني

كما يظهر لنا في معظم البرامج ذات الواجهة الرسومية فهي تحتوي على أكثر من فورم. ولعمل ذلك في لازاراس نقوم بالآتي:

1. إنشاء برنامج جديد نحفظه في دليل نقوم بتمسيته secondform
2. نقوم بحفظ الوحدة الرئيسية بإسم main.pas وتسمية الفورم الرئيسي بالإسم fmMain ونقوم بتسمية المشروع بإسم secondform.lpi
3. نقوم بإضافة فورم آخر بواسطة File/ New Form ونقوم بحفظ الوحدة بإسم second.pas ونمي الفورم fmSecond
4. نقوم بإضافة Label نتكب فيه Second Form بخط كبير. يمكن تكبير الخط في خاصية الـ Font.Size الموجودة في الـ Label
5. نرجع للفورم الرئيسي main ونقوم بوضع زر فيه.
6. نقوم بكتابة هذا السطر في كود الوحدة الرئيسية main تحت قسم implementation:

```
uses second;
```

7. في الحدث OnClick بالنسبة للزر الموجود في الفورم الرئيسي fmMain نقوم بكتابة الكود التالي:

```
  fmSecond.Show;
```

نقوم بتشغيل البرنامج ونضغط الزر في الفور الأول ليظهر لنا الفورم الثاني.

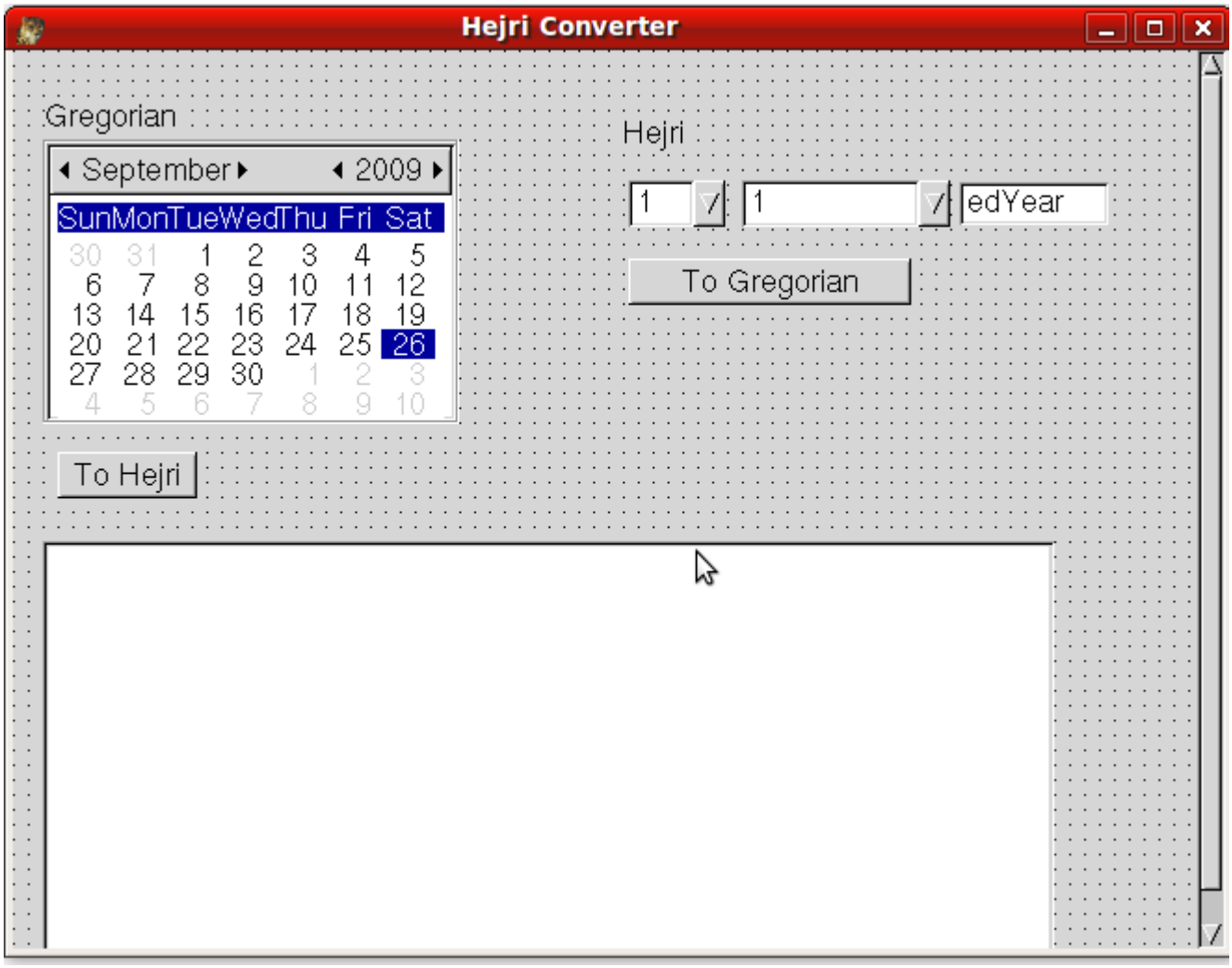
برنامج المحول الهجري

هذا البرنامج يستخدم الوحدة HejriUtils الموجودة في الفصل السابق، ويمكن الرجوع إليها في الأمثلة المصاحبة للكتاب.

نقوم بإنشاء برنامج جديد ونقوم بإدراج الكائنات التالية في الفورم:

```
2 Tlabel
TCalnder
2 TCompoBox
2 Tbutton
Tmemo
TEdit
```

ونقوم بتصميم الفورم كالشكل التالي:



وكود الوحدة المصاحبة للفورم الرئيسي هو:

```
unit main;

{$mode objfpc}{$H+}

interface


uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs,
  StdCtrls, Calendar, HejriUtils;


type

  { TfmMain }

  TfmMain = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Calendar1: TCalendar;
```

```
    cbDay: TComboBox;
    cbMonth: TComboBox;
    edYear: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Memo1: TMemo;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  fmMain: TfmMain;

implementation

{ TfmMain }



procedure TfmMain.Button1Click(Sender: TObject);
var
  Year, Month, Day: Word;
begin
  DateToHejri(Calendar1.DateTime, Year, Month, Day);
  Memo1.Lines.Add(DateToStr(Calendar1.DateTime) + '  =  ' +
   Format('%d.%s.%d', [Day, HejriMonthsAr[Month], Year]));
  edYear.Text:= IntToStr(Year);
  cbMonth.ItemIndex:= Month - 1;
  cbDay.ItemIndex:= Day - 1;
end;

procedure TfmMain.Button2Click(Sender: TObject);
var
  ADate: TDateTime;
  CYear, CMonth, CDay: Word;
  YearD, MonthD, DayD: Word;
begin
  ADate:= HejriToDate(StrToInt(edYear.Text), cbMonth.ItemIndex + 1,
    cbDay.ItemIndex + 1);
  Memo1.Lines.Add(FormatDateTime('yyyy-mm-dd ddd', ADate));

  DateToHejri(Now, CYear, CMonth, CDay);
  HejriDifference(StrToInt(edYear.Text), cbMonth.ItemIndex + 1,
   cbDay.ItemIndex + 1, CYear, CMonth,
    CDay, YearD, MonthD, DayD);
  Memo1.Lines.Add('Difference = ' +
    Format('%d year, %d month, %d day ', [YearD, MonthD, DayD]));
  Calendar1.DateTime:= ADate;

end;
```

```
procedure TfmMain.FormCreate(Sender: TObject);
var
  i: Integer;
  Year, Month, Day: Word;
begin
  Calendar1.DateTime:= Now;
  cbMonth.Clear;
  for i:= 1 to 12 do
    cbMonth.Items.Add(HejriMonthsAr[i]);
  DateToHejri(Now, Year, Month, Day);
  cbDay.ItemIndex:= Day - 1;
  cbMonth.ItemIndex:= Month - 1;
  edYear.Text:= IntToStr(Year);

end;

initialization
  {$I main.lrs}

end.
```

الفصل الرابع

البرمجة الكائنية المنحى

Object Oriented Programming

مقدمة

ترتكز فكرة البرمجة الكائنية على أنها تقوم بتقسيم البرنامج ووحداته ومكوناته إلى كائنات. فالكائن Object هو عبارة عن:

1. **مجموعة خصائص Properties** التي يمكن أن نعتبرها متغيرات تمثل حالة الكائن.
2. **إجراءات ودوال** تسمى **Methods** يتم تنفيذها فتؤثر على خصائص الكائن.
3. **أحداث Events** تحدث له مثل نقرة بالماوس أو إظهار له في الشاشة وغيرها.
4. **إجراءات الأحداث Event Handlers** مثل الإجراء الذي يتم تنفيذه عند نقرة الماوس.

وتتكامل هذه الخصائص والإجراءات لتكوِّن كائن. وتقوم هذه الإجراءات بالتأثير مباشرة على الخصائص.

بيانات + كود = كائن

وكمثال لكائن هو مااستخدمناه في الفصل السابق من برمجة الواجهة الرسومية، فهو يعكس الإستخدام اليومي للمبرمج للبرمجة الكائنية المنحى. فنجد أن الفورم هو عبارة عن كائن به بعض الخصائص مثل الـ Caption, Width, Height وبه بعض الإجراءات التي تؤثر عليه تأثير مباشر مثل Close, Show, ShowModal, Hide وغيرها. كذلك فإن له أحداث مثل OnClick, OnCreate, OnClose وله إجراءات مصاحبة لهذه الأحداث، وهي في هذه الحالة الكود الذي يكتبه المبرمج ليتم تنفيذه إستجابة لحدث معين.

المثال الأول، برنامج التاريخ والوقت

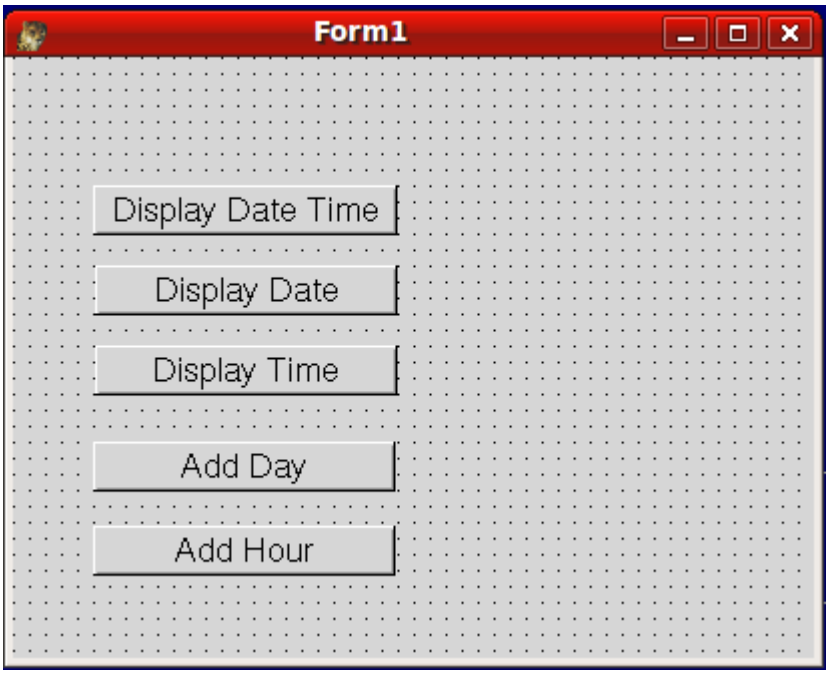
نفرض أننا نريد أن نقوم بكتابة كود لكائن يحتوي على تاريخ ووقت، وبعض الإجراءات التي تخص التاريخ والوقت.
قمنا بإنشاء برنامج جديد، ثم وحدة جديدة أسميناها DateTimeUnit وفيها كائن يسمى TmyDateTime، وهذا هو الكود الذي قمنا بكتابته في هذه الوحدة:

```
unit DateTimeUnit;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils;

type
  { TMyDateTime }

  TMyDateTime = class
  private
    fDateTime: TDateTime;
  public
    function GetDateTime: TDateTime;
    procedure SetDateTime(ADateTime: TDateTime);
    procedure AddDays(Days: Integer);
```

```
    procedure AddHours(Hours: Single);
    function GetDateTimeAsString: string;
    function GetTimeAsString: string;
    function GetDateAsString: string;
    constructor Create(ADateTime: TDateTime);
    destructor Destroy; override;
  end;

implementation

{ TMyDateTime }

function TMyDateTime.GetDateTime: TDateTime;
begin
  Result:= fDateTime;
end;

procedure TMyDateTime.SetDateTime(ADateTime: TDateTime);
begin
  fDateTime:= ADateTime;
end;

procedure TMyDateTime.AddDays(Days: Integer);
begin
  fDateTime:= fDateTime + Days;
end;

procedure TMyDateTime.AddHours(Hours: Single);
begin
  fDateTime:= fDateTime + Hours / 24;
end;

function TMyDateTime.GetDateTimeAsString: string;
begin
  Result:= DateTimeToStr(fDateTime);
end;

function TMyDateTime.GetTimeAsString: string;
begin
  Result:= TimeToStr(fDateTime);
end;

function TMyDateTime.GetDateAsString: string;
begin
  Result:= DateToStr(fDateTime);
end;

constructor TMyDateTime.Create(ADateTime: TDateTime);
begin
  fDateTime:= ADateTime;
end;

destructor TMyDateTime.Destroy;
begin
  inherited Destroy;
end;
```

```
end.
```

وفي الوحدة الرئيسية للبرنامج main قمنا بإضافة 5 أزرار كما في الشكل التالي:



وقمنا بكتابة الكود التالي في الأزرار:

```
unit main;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs,
  StdCtrls, DateTimeUnit;

type

  { TForm1 }


  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
```

```
  private
    { private declarations }
  public
    MyDT: TMyDateTime;
    { public declarations }
  end;

var
  Form1: TForm1;

implementation


{ TForm1 }

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyDT:= TMyDateTime.Create(Now);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(MyDT.GetDateTimeAsString);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ShowMessage(MyDT.GetDateAsString);
end;


procedure TForm1.Button3Click(Sender: TObject);
begin
  ShowMessage(MyDT.GetTimeAsString);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  MyDT.AddHours(1);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  MyDT.AddDays(1);
end;


initialization
  {$I main.lrs}

end.
```

نلاحظ في البرنامج السابق التالي:

في الوحدة DateTimeUnit

1. قمنا بتعريف TmyDateTime على أنه من النوع Class وهذه هي طريقة تعريف كائن والذي يمكن أن يحتوي على إجراءات، دوال ومتغيرات كما في هذا المثال
2. قمنا بتعريف إجراء اسمه Create نوع هذا الإجراء هو: Constructor ، وهو إجراء من نوع خاص يستخدم لإنشاء وتفعيل وتهيئة متغيرات كائن ما في الذاكرة حتى يمكن إستخدامه.
3. قمنا بتعريف إجراء اسمه Destroy نوعه Destructor ، وهو إجراء من نوع خاص تتم مناداته عند الإنتهاء من العمل بهذا الكائن وإرادة حذفه من الذاكرة. وفي نهايته كلمة override سوف نتطرق لها في كتاب قادم إن شاء الله.
4. نلاحظ أن هناك جزئين، الجزء الأول هو private , ومتغيراته ودواله وإجراءاته لايمكن الوصول لها من وحدة أخرى عند استخدام هذا الكائن.
5. الجزء الثاني هو public وهو ذو متغيرات، دوال وإجراءات يمكن الوصول لها من خارج وحدة هذا الكائن، وهي تمثل الواجهة المرئية لهذا الكائن مثل Interfaces بالنسبة للوحدات.

أما في الوحدة الرئيسية للبرنامج main فنجد الآتي:

1. قمنا بإضافة إسم الوحدة DateTimeUnit مع باقي الوحدات في الجزء Uses

2. قمنا بتعريف الكائن MyDT داخل كائن الفورم:

```
MyDT: TMyDateTime;
```

3. قمنا بإنشاء وتهيئة الكائن عند الحدث OnFormCreate:

```
MyDT:= TMyDateTime.Create(Now);
```

وهذه هي طريقة إنشاء وتهيئة الكائن في لغة أوبجكت باسكال.

برنامج الأخبار بطريقة كائنية

في المثال التالي قمنا بإعادة كتابة برنامج الأخبار بطريقة كائنية. وفي هذه المرة قمنا بتصنيف الأخبار، فكل صنف يتم تسجيله في ملف بيانات منفصل من نوع ملفات الوصول المباشر.
قمنا بإنشاء برنامج جديد ذو واجهة رسومية وأسميناه oonews، وقمنا بإضافة وحدة جديدة كتبنا فيها كود الكائن TNews كالآتي:

```
unit news;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils;

type
  TNewsRec = record
    ATime: TDateTime;
    Title: string[100];
  end;

  { TNews }

  TNews = class
    private
      F: file of TNewsRec;
      fFileName: string;
    public
      constructor Create(FileName: string);
      destructor Destroy; override;
      procedure Add(ATitle: string);
      procedure ReadAll(var NewsList: TStringList);
      function Find(Keyword: string;
        var ResultList: TStringList): Boolean;
  end;

implementation

{ TNews }

constructor TNews.Create(FileName: string);
begin
  fFileName:= FileName;
end;

destructor TNews.Destroy;
begin
  inherited Destroy;
end;

procedure TNews.Add(ATitle: string);
var
  Rec: TNewsRec;
```

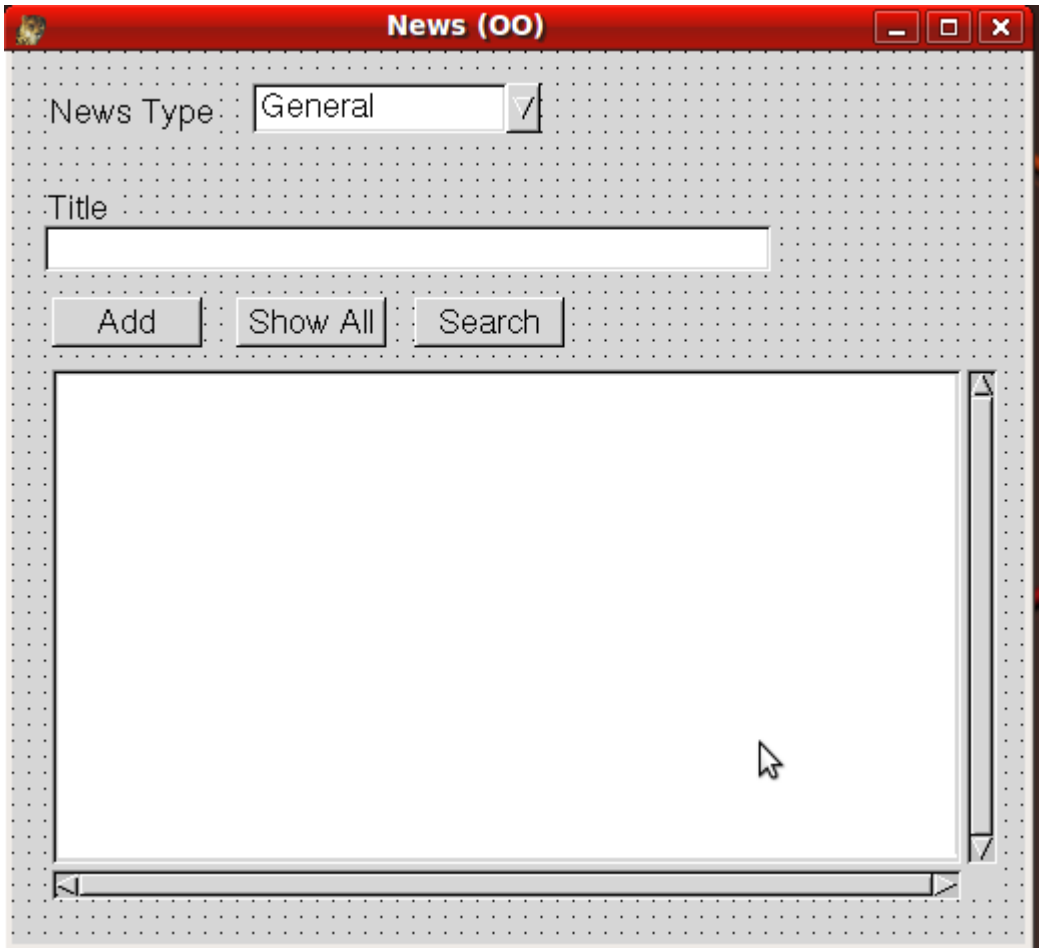
```
begin
  AssignFile(F, fFileName);
  if FileExists(fFileName) then
  begin
    FileMode:= 2; // Read/write access
    Reset(F);
    Seek(F, FileSize(F));
  end

  else
    Rewrite(F);

  Rec.ATime:= Now;
  Rec.Title:= ATitle;
  Write(F, Rec);
  CloseFile(F);

end;

procedure TNews.ReadAll(var NewsList: TStringList);
var
  Rec: TNewsRec;
begin
  NewsList.Clear;
  AssignFile(F, fFileName);
  if FileExists(fFileName) then
  begin
    Reset(F);
    while not Eof(F) do
    begin
      Read(F, Rec);
      NewsList.Add(DateTimeToStr(Rec.ATime) + ' : ' + Rec.Title);
    end;
    CloseFile(F);
  end;

end;

function TNews.Find(Keyword: string; var ResultList: TStringList): Boolean;
var
  Rec: TNewsRec;
begin
  ResultList.Clear;
  Result:= False;
  AssignFile(F, fFileName);
  if FileExists(fFileName) then
  begin
    Reset(F);
    while not Eof(F) do
    begin
      Read(F, Rec);
      if Pos(LowerCase(Keyword), LowerCase(Rec.Title)) > 0 then
      begin
        ResultList.Add(DateTimeToStr(Rec.ATime) + ' : ' + Rec.Title);
        Result:= True;
      end;
    end;
```

```
    CloseFile(F);
  end;

end;

end.
```

وفي البرنامج الرئيسي قمنا بإضافة هذه المكونات:

Edit, ComboBox, 3 buttons, Memo, 2 lables

فأصبح شكل الفورم كالآتي:



وقمنا بكتابة الكود التالي للوحدة المصاحبة للفورم الرئيسي:

```
unit main;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics,
    Dialogs, News, StdCtrls;

type
```

```
  { TForm1 }

  TForm1 = class(TForm)
    btAdd: TButton;
    btShowAll: TButton;
    btSearch: TButton;
    cbType: TComboBox;
    edTitle: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Memo1: TMemo;
    procedure btAddClick(Sender: TObject);
    procedure btSearchClick(Sender: TObject);
    procedure btShowAllClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { private declarations }
  public
    NewsObj: array of TNews;
    { public declarations }
  end;

var
  Form1: TForm1;

implementation

{ TForm1 }

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  SetLength(NewsObj, cbType.Items.Count);
  for i:= 0 to High(NewsObj) do
    NewsObj[i]:= TNews.Create(cbType.Items[i] + '.news');

end;

procedure TForm1.btAddClick(Sender: TObject);
begin
  NewsObj[cbType.ItemIndex].Add(edTitle.Text);
end;

procedure TForm1.btSearchClick(Sender: TObject);
var
  SearchStr: string;
  ResultList: TStringList;
begin
  ResultList:= TStringList.Create;
  if InputQuery('Search News', 'Please input keyword', SearchStr) then
    if NewsObj[cbType.ItemIndex].Find(SearchStr, ResultList) then
    begin
      Memo1.Lines.Clear;
      Memo1.Lines.Add(cbType.Text + ' News');
      Memo1.Lines.Add('--------------------------------------------------');
```

```
      Memo1.Lines.Add(ResultList.Text);
    end
    else
      Memo1.Lines.Text:= SearchStr + ' not found in ' +
        cbType.Text + ' news';
  ResultList.Free;
end;

procedure TForm1.btShowAllClick(Sender: TObject);
var
  List: TStringList;
begin
  List:= TStringList.Create;
  NewsObj[cbType.ItemIndex].ReadAll(List);
  Memo1.Lines.Clear;
  Memo1.Lines.Add(cbType.Text + ' News');
  Memo1.Lines.Add('------------------------------------------------------------');
  Memo1.Lines.Add(List.Text);
  List.Free;
end;

procedure TForm1.FormClose(Sender: TObject; var CloseAction: TCloseAction);
var
  i: Integer;
begin
  for i:= 0 to High(NewsObj) do
    NewsObj[i].Free;

  NewsObj:= nil;

end;

initialization
  {$I main.lrs}

end.
```

نلاحظ في البرنامج السابق الآتي:

1. أننا استخدمنا نوع جديد من تعريف المصفوفات وهو مايعرف **بالمصفوفة المرنة** Dynamic Array، وهي مصفوفة غير محددة الطول في وقت كتابة الكود، لكن طولها يتغير زيادة أو نقصاناً أثناء تشغيل البرنامج حسب الحاجة، وطريقة التعريف هي كالآتي:

```
  NewsObj: array of TNews;
```

وفي أثناء تنفيذ البرنامج وقبل إستخدامها يجب أن نقوم بحجز مساحة لها في الذاكرة بإستخدام الإجراء SetLength، مثلاً إذا قمنا بكتابة الكود التالي:

```
  SetLength(NewsObj, 10);
```

فهذا يعني أننا قمنا بحجز عشر خانات ، فهي تماثل في هذه الحالة هذا التعريف:

```
  NewsObj: array [0 .. 9] of Tnews;
```

وفي حالة برنامج الأخبار قمنا بحجز خانات تمثل عدد أنواع الأخبار الموجودة في الـ Combo Box:

```
SetLength(NewsObj, cbType.Items.Count);
```

وبهذه الطريقة تجعل عدد الكائنات معتمدة على عدد أنواع الأخبار المكتوبة في الكائن ComboBox.Items فكلما قام المبرمج بزيادتها، كلما زادت تلقائياً بدون الحاجة لإعادة تغيير الطول.

2. نوع الكائن **TNews** هو عبارة عن Class وهي تمثل النوع، ولايمكن استخدامه مباشرة إلا بتعريف متغيرات منه تسمى كائنات Objects. بنفس الطريقة التي نعتبر فيها أن النوع الصحيح Integer لايمكن إستخدامه مباشرة بل يجب استخدام متغيرات من نوعه، مثل I, j, x.

3. في نهاية البرنامج قمنا بتحرير الكائنات في المصفوفة من الذاكرة أولاً ثم قمنا بتحرير وحذف المصفوفة المرنة ثانياً بواسطة الكود التالي :

```
for i:= 0 to High(NewsObj) do
  NewsObj[i].Free;

NewsObj:= nil;
```

برنامج الصفوف

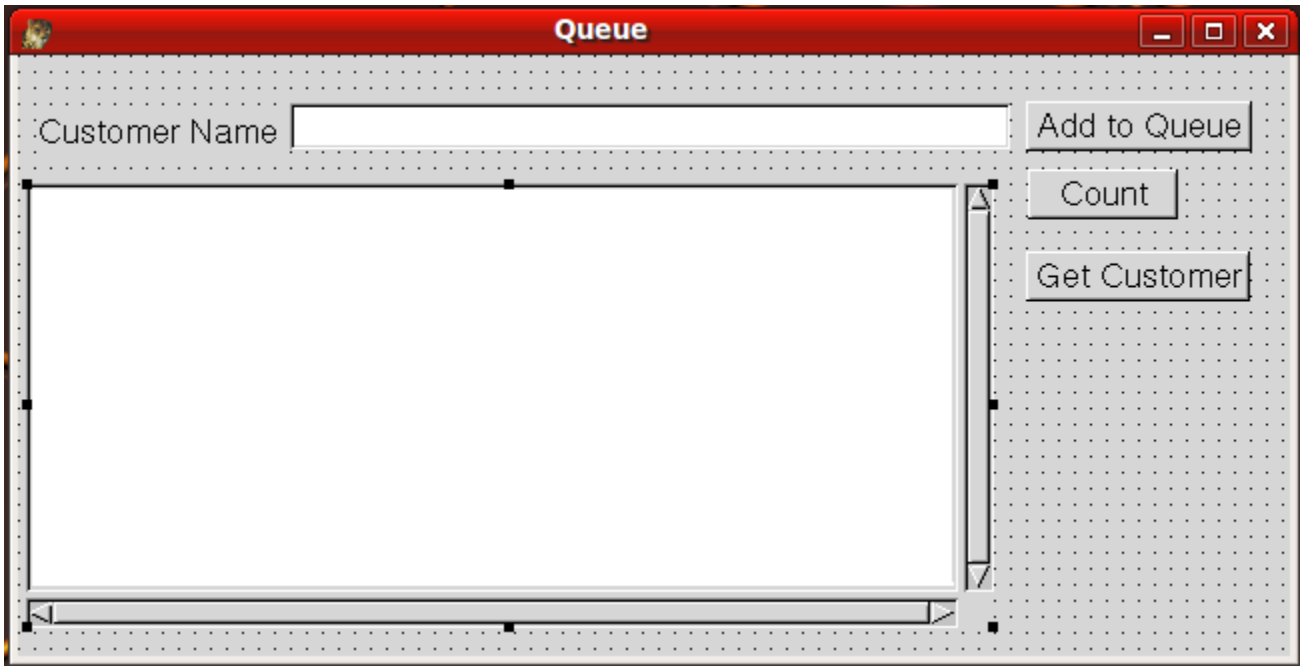
الصفوف هي إحدى دروس هيكلية البيانات Data structure، وهي تستخدم كحلول لمعاجة الصف أو الطابور. ويتميز الصف بأن من يدخله أولاً يخرج أولاً إذا لم تكن هناك أولوية. في هذه الحال تكون الأولوية هي زمن الدخول في الصف.
في البرنامج التالي قمنا بكتابة وحدة Queue تحتوي على نوع الكائن TQueue يمكن استخدامه لإضافة بيانات (مثلاً أسماء) إلى صف ثم إستخراج هذه البيانات من الصف. والصف يختلف عن المصفوفة في أن القراءة (الإستخراج) منه تؤثر على البيانات وطول الصف، فمثلاً إذا كان هناك صف يحتوي على 10 عناصر، ثم قمنا بقراءة والحصول على ثلاث عناصر منها فإن عدد من في الصف ينقص إلى سبعة. وإذا قمنا بقراءة العناصر جميعاً أصبح الصفُ فارغاً.

كود وحدة الصف:

```
unit queue;
// This unit contains TQueue class,
// which is suitable for any string queue

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils;

type

  { TQueue }

  TQueue = class
    private
      fArray: array of string;
      fTop: Integer;
    public
      constructor Create;
      destructor Destroy; override;
      function Put(AValue: string): Integer;
      function Get(var AValue: string): Boolean;
      function Count: Integer;
      function ReOrganize: Boolean;
  end;

implementation

{ TQueue }

constructor TQueue.create;
begin
  fTop:= 0;
end;

destructor TQueue.destroy;
begin
  SetLength(fArray, 0); // Erase queue array from memory
  inherited destroy;
end;
```

```
function TQueue.Put(AValue: string): Integer;
begin
  if fTop >= 100 then
    ReOrganize;

  SetLength(fArray, Length(fArray) + 1);
  fArray[High(fArray)]:= AValue;
  Result:= High(fArray) - fTop;
end;

function TQueue.Get(var AValue: string): Boolean;
begin
  AValue:= '';
  if fTop <= High(fArray) then
  begin
    AValue:= fArray[fTop];
    Inc(fTop);
    Result:= True;
  end
  else // empty
  begin
    Result:= False;

    // Erase array
    SetLength(fArray, 0);
    fTop:= 0;
  end;
end;

function TQueue.Count: Integer;
begin
  Result:= Length(fArray) - fTop;
end;

function TQueue.ReOrganize: Boolean;
var
  i: Integer;
  PCount: Integer;
begin
  if fTop > 0 then
  begin
    PCount:= Count;
    for i:= fTop to High(fArray) do
      fArray[i - fTop]:= fArray[i];

    // Truncate unused data
    setLength(fArray, PCount);
    fTop:= 0;
    Result:= True; // Re Organize is done

  end
  else
    Result:= False; // nothing done
  end;
end;
end.
```

الفورم الرئيسي لبرنامج الصف:



كود الوحدة الرئيسية للفورم:

```
unit main;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics,
 Dialogs, Queue, StdCtrls;

type

  { TfmMain }

  TfmMain = class(TForm)
    bbAdd: TButton;
    bbCount: TButton;
    bbGet: TButton;
    edCustomer: TEdit;
    Label1: TLabel;
    Memo1: TMemo;
    procedure bbAddClick(Sender: TObject);
    procedure bbCountClick(Sender: TObject);
    procedure bbGetClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
    procedure FormCreate(Sender: TObject);
  private
```

```
    { private declarations }
  public
    MyQueue: TQueue;
    { public declarations }
  end;

var
  fmMain: TfmMain;

implementation


{ TfmMain }

procedure TfmMain.FormCreate(Sender: TObject);
begin
  MyQueue:= TQueue.Create;
end;

procedure TfmMain.FormClose(Sender: TObject; var CloseAction: TCloseAction);
begin
  MyQueue.Free;
end;

procedure TfmMain.bbCountClick(Sender: TObject);
begin
  Memo1.Lines.Add('Queue length is: ' + IntToStr(MyQueue.Count));
end;

procedure TfmMain.bbAddClick(Sender: TObject);
var
  APosition: Integer;
begin
  APosition:= MyQueue.Put(edCustomer.Text);
  Memo1.Lines.Add(edCustomer.Text + ' has been added as # ' +
    IntToStr(APosition + 1));
end;

procedure TfmMain.bbGetClick(Sender: TObject);
var
  ACustomer: string;
begin
  if MyQueue.Get(ACustomer) then
  begin
    Memo1.Lines.Add('Got: ' + ACustomer + ' from the queue');
  end
  else
    Memo1.Lines.Add('Queue is empty');
end;

initialization
  {$I main.lrs}

end.
```

في البرنامج السابق في الإجراء Put يتم زيادة سعة المصفوفة المرنة لإستيعاب العنصر الجديد. وعند إستخراج عنصر عن طريق الدالة Get يتم تحريك المؤشر fTop الذي يؤشر على أول العانصر بالصف. وبعد هذا الإجراء لايمكن حذف المصفوفة من البداية، حيث أن المصفوفة المرنة فقط يمكن تقليصها أو توسيعها من النهاية، لذلك اكتفينا فقط بتحريك المؤشر مع بقاء العناصر القديمة تحتل مكاناً في المصفوفة. ولمعالجة هذه المشكلة قمنا بعمل الإجراء ReOrganize لتفريغ الصف من العناصر الغير مستخدمة والتي تم إستخراجها من قبل حتى لانحتل ذاكرة بدون فائدة.
الإجراء ReOrganize ببساطة يعمل كلما يصل عدد المستخرجين من الصف حوالي مائة، حيث يتم نداء هذا الإجراء من الدالة Put. ويمكن زيادة هذا الرقم إلى ألف مثلاً حتى لايتسبب هذا الإجراء في إبطاء الإضافة، فقط يعمل كلما يصل طول الصف الفعلي في الذاكرة إلى ألف.
في هذا الإجراء نقوم بنقل العناصر الموجودة في الصف إلى مكان العناصر القديمة المستخرجة مسبقاً، بهذه الطريقة:

```
for i:= fTop to High(fArray) do
  fArray[i - fTop]:= fArray[i];
```

ثم نقوم بعد ذلك بقص المصفوفة عند آخر عنصر داخل الصف ونضع مؤشر الصف في بداية المصفوفة كالتالي:

```
// Truncate unused data
setLength(fArray, PCount);
fTop:= 0;
```

من البرنامج السابق نجد أن البرمجة الكائنية وفرت لنا حماية البيانات Information hiding حيث أن البيانات الحساسة التي ربما تسبب في سلوك غير معلوم إذا أتحنا للمستخدم الوصول المباشر لها. لذلك قمنا بإخفاء هذه المتغيرات المهمة والتي لايحتاج ولايفترض أن يقوم المستخدم بالوصول لها مباشرة وهي:

```
private
  fArray: array of string;
  fTop: Integer;
```

حيث أن المبرمج الذي يستخدم هذا الكائن لايتسطيع الوصول لها من خلال برنامجه الرئيسي ولو وصل لها لتسبب في ضياع البيانات أو حدوث أخطاء. مثلاً نفرض أنه قام بتغيير قيمة fTop إلى 1000 في حين وجود فقط 10 عناصر في الصف، فهذا يتسبب بخطأ أثناء التشغيل.
وكبديل سمحنا له بالوصول لإجراءات ودوال آمنة طبيعية تتناسب وطبيعة تحقيق الهدف، مثل Put, Get. فمهما إستخدمها المبرمج لانتوقع منها حدوث خطأ أو شيء غير منطقي. وهذه الطريقة اشبه بإستخدام بوابات معروفة للوصول للبيانات. وفي هذه الحالة البوابات هي الإجراءات والدوال الموجودة في هذا الكائن، حيث لايمكن إستخدام إجراءات أخرى غير الموجودة في هذا الكائن للوصول للبيانات. وهذه الميزة في البرمجة الكائنية تسمى encapsulation.

ومما سبق نجد أن الكائن بالنسبة للمُستخدم (المبرمج الذي يستخدمه في برامجه) هو عبارة عن إجراءات ودوال تقبع خلفها بيانات تخص هذا الكائن.

الملف الكائني Object Oriented File

رأينا في الفصل الأول كيفية التعامل مع الملفات بأنواعها المختلفة. وقد كانت طريقة التعامل معتمدة على نوع البرمجة الهيكلية، أي أنها تتعامل مع دوال وإجراءات ومتغيرات فقط، أما هذه المرة نريد الوصول للملفات عن طريق الكائن الموجود في مكتبات فري باسكال ولازاراس وهو TFileStream.
والكائن TFileStream يتعامل مع الملفات بطريقة أشبه بنوع الملف غير محدد النوع Untyped File، والذي بدوره يصلح لكافة أنواع الملفات.
تتميز طريقة إستخدام الملف الكائني بأنها تحتوي على إجراءات ودوال وخصائص غنيّة تمتاز بكل ماتمتاز به البرمجة الكائنية من سهولة الإستخدام والمنطقية في التعامل والقياسية وتوقع المبرمج لطريقة الإستخدام.

برنامج نسخ الملفات بواسطة TFileStream

في هذا المثال نريد نسخ ملف بإستخدام هذا النوع للوصول للملفات، فنقوم بإنشاء برنامج جديد ذو واجهة رسومية ونقوم بإدراج هذه المكونات:

TButton, TOpenDialog, TSaveDialog

ثم نقوم بكتابة الكود التالي ي الحدث OnClick بالنسبة للزر:

```
procedure TfmMain.Button1Click(Sender: TObject);
var
  SourceF, DestF: TFileStream;
  Buf: array [0 .. 1023] of Byte;
  NumRead: Integer;
begin
  if OpenDialog1.Execute and SaveDialog1.Execute then
  begin
    SourceF:= TFileStream.Create(OpenDialog1.FileName, fmOpenRead);
    DestF:= TFileStream.Create(SaveDialog1.FileName, fmCreate);
    while SourceF.Position < SourceF.Size do
    begin
      NumRead:= SourceF.Read(Buf, SizeOf(Buf));
      DestF.Write(Buf, NumRead);
    end;
    SourceF.Free;
    DestF.Free;
    ShowMessage('Copy finished');
  end;
end;
```

وهي طريقة مشابهة لطريقة نسخ الملفات بإستخدام الملفات غير محددة النوع التي استخدمناها في الفصل الأول.

ويمكن أيضاً نسخ الملف بطريقة مبسطة وهي كالآتي:

```
procedure TfmMain.Button1Click(Sender: TObject);
var
  SourceF, DestF: TFileStream;
begin
  if OpenDialog1.Execute and SaveDialog1.Execute then
  begin
```

```
    SourceF:= TFileStream.Create(OpenDialog1.FileName, fmOpenRead);
    DestF:= TFileStream.Create(SaveDialog1.FileName, fmCreate);
    DestF.CopyFrom(SourceF, SourceF.Size);
    SourceF.Free;
    DestF.Free;
    ShowMessage('Copy finished');
  end;
end;
```

حيث أن الإجراء CopyFrom يقوم بعملية النسخ لمحتويات الملف كاملاً لأننا حددنا حجم الجزء المراد نسخه وهو SourceF.Size الذي يمثل الحجم الكامل للملف بالبايت.

الوراثة Inheritance

الوراثة في البرمجة الكائنية تعني إنشاء كائن جديد من كائن موجود مسبقاً. وهو يعني أن الكائن الجديد يرث صفات الكائن القديم ويمكن أن يزيد عليه بعض الخصائص والأجراءات والدوال.

كمثال للوراثة نريد عمل كائن لصف من النوع الصحيح، فبدلاً من كتابته من الصفر يمكن الإعتماد على كائن الصف الذي قمنا بكتابته سابقاً والذي يستخدم المقاطع. ولوراثة كائن ما نقوم بإنشاء وحدة جديدة ونقوم بإستخدام الوحدة القديمة المحتوية على الكائن القديم. ونقوم بتعريف الكائن الجديد كالآتي:

```
  TIntQueue = class(TQueue)
```

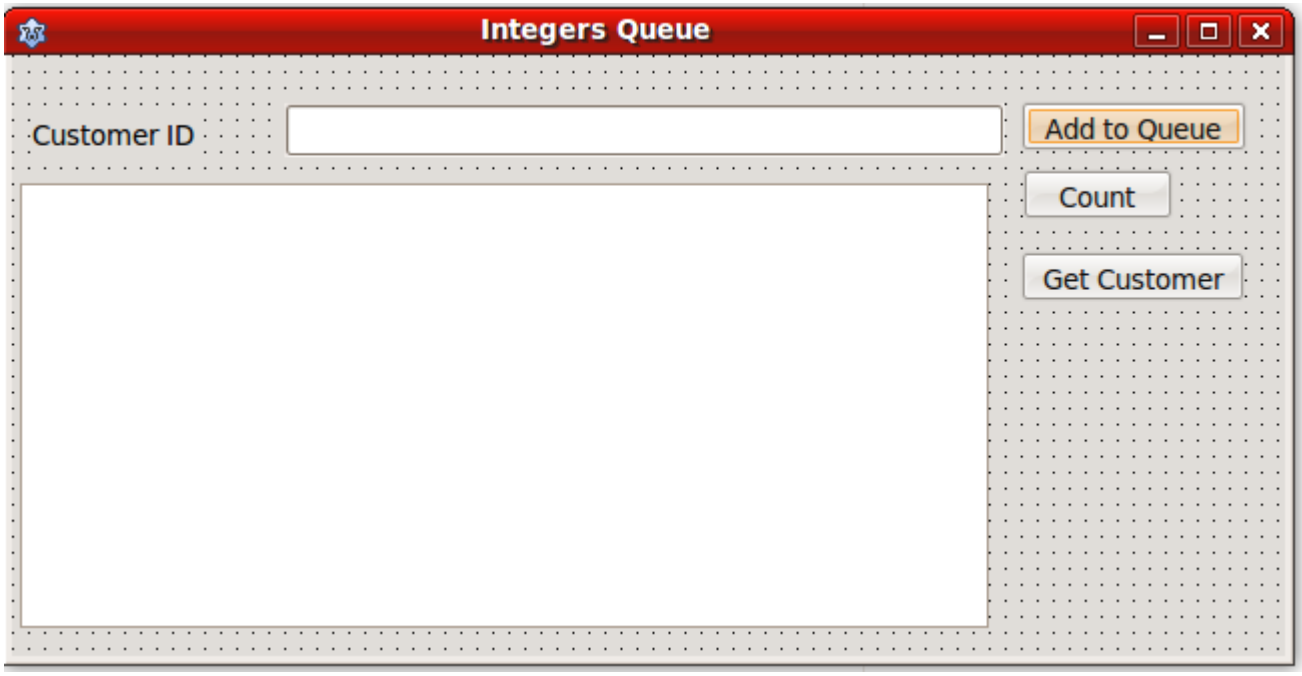
وقد أسمينا الوحدة الجديدة IntQueue. وقمنا بإضافة فقط دالتين في الكائن الجديد PutInt, GetInt.
وكود الوحدة الجديدة كاملاً هو:

```
unit IntQueue;

// This unit contains TIntQueue class, which is inherits TQueue
// class and adds PutInt, GetInt methods to be used with
// Integer queue

{$mode objfpc}{$H+}

interface


uses
  Classes, SysUtils, Queue;

type

  { TIntQueue }

  TIntQueue = class(TQueue)

  public
    function PutInt(AValue: Integer): Integer;
    function GetInt(var AValue: Integer): Boolean;
```

```
  end;

implementation

{ TIntQueue }

function TIntQueue.PutInt(AValue: Integer): Integer;
begin
  Result:= Put(IntToStr(AValue));
end;

function TIntQueue.GetInt(var AValue: Integer): Boolean;
var
  StrValue: string;
begin
  Result:= Get(StrValue);
  if Result then
    AValue:= StrToInt(StrValue);

end;

end.
```

نلاحظ أننا لم نقم بتكرار بعض العمليات مثل Count, Create, Destroy لأنها موروثة من الكائن TQueue.

لإستخدام الوحدة الجديدة قمنا بإنشاء الفور التالي:



وكود الفورم السابق هو:

```
unit main;

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

```
interface

uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics,
    Dialogs, IntQueue, StdCtrls;

type

  { TfmMain }

  TfmMain = class(TForm)
    bbAdd: TButton;
    bbCount: TButton;
    bbGet: TButton;
    edCustomerID: TEdit;
    Label1: TLabel;
    Memo1: TMemo;
    procedure bbAddClick(Sender: TObject);
    procedure bbCountClick(Sender: TObject);
    procedure bbGetClick(Sender: TObject);
    procedure FormClose(Sender: TObject;
       var CloseAction: TCloseAction);
    procedure FormCreate(Sender: TObject);
  private
    { private declarations }
  public
    MyQueue: TIntQueue;
    { public declarations }
  end;

var
  fmMain: TfmMain;

implementation


{ TfmMain }

procedure TfmMain.FormCreate(Sender: TObject);
begin
  MyQueue:= TIntQueue.Create;
end;

procedure TfmMain.FormClose(Sender: TObject; var CloseAction: TCloseAction);
begin
  MyQueue.Free;
end;

procedure TfmMain.bbCountClick(Sender: TObject);
begin
  Memo1.Lines.Add('Queue length is: ' + IntToStr(MyQueue.Count));
end;

procedure TfmMain.bbAddClick(Sender: TObject);
var
  APosition: Integer;
```

```
begin
  APosition:= MyQueue.PutInt(StrToInt(edCustomerID.Text));
  Memo1.Lines.Add(edCustomerID.Text + ' has been added as # '
   + IntToStr(APosition + 1));
end;

procedure TfmMain.bbGetClick(Sender: TObject);
var
  ACustomerID: Integer;
begin
  if MyQueue.GetInt(ACustomerID) then
  begin
    Memo1.Lines.Add('Got: Customer ID : ' + IntToStr(ACustomerID) +
    ' from the queue');
  end
  else
    Memo1.Lines.Add('Queue is empty');
end;

initialization
  {$I main.lrs}

end.
```

نلاحظ أننا قمنا بنداء الإجراءات والدوال الموجودة في الكائن TQueue والدوال الجديدة الموجودة فقط في TIntQueue.

في هذه الحالة تُسمي الكائن TQueue الكائن الأساس base class أو السلف ancestor. ونسمي الكائن الجديد TIntQueue المُنحدر descender أو الوريث.

ملحوظة:

كان من الممكن عدم استخدام الوراثة في المثال السابق وإضافة الدوال الجديدة مباشرة في الكائن الأساسي TQueue، لكن لجأنا لذلك لشرح الوراثة، وهناك سبب آخر، هو أننا ربما لانمتلك الكود المصدر للوحدة Queue، ففي هذه الحالة يتعذر تعديلها، ويكون السبيل الوحيد هي الوراثة منها ثم تعديلها.
ويمكن للمبرمج أن يقوم بتوزيع الوحدات في شكل ملفات ذات الإمتداد **ppu** كما في فري باسكال **أو** **dcu** كما في دلفي، في هذه الحال لايمكن الإطلاع على الكود المصدري، حيث يمكن الإطلاع على الكود فقط في حالة الحصول على الملف **pas**. .

وفي الختام نتمنى أن يُنال بهذا الكتاب فائدة المسلمين.

و بعد إنتهاء الدارس من هذا الكتاب، يمكنه قراءة الكتاب الآخر "الخطوة الثانية مع أوبجكت باسكال – صناعة البرمجيات"

Code.sd

ذي الحجة 1431
3.12.2010