بسم الله الرحمن الرحيم

أول إصدارة: ذي الحجة 1431 هجرية

الإصدارة الحالية : ربيع الأول 1432 هجرية

تأليف: معتز عبدالعظيم الطاهر
http://code.sd

مقدمة

بسم الله الرحمن الرحيم، والصلاة والسلام على أشرف الأنبياء والمرسلين، نبينا محمد وعلى آله وصحبه أجمعين، أما بعد، فبعد ما كان كتاب إبدأ مع لغة أوبجكت باسكال هو مدخل لهذه اللغة، أصبحت حاجة المبرمج أن يتقدم في خطوات أكثر عمقاً بإستخدام أوبجكت باسكال في البيئة العملية.
فنجد أن فصول هذا الكتاب تتناول مواضيع تُستخدم مباشرة في العمل والبرامج الكبيرة، ومن هذه المواضيع إدارة الذاكرة ، قواعد البيانات، برمجة الويب،والإتصالات. وهو يحتوي على أمثلة كثيرة مصاحبة له تُعتبر برامج عملية يمكن تطويرها وإستخدامها وإستقاء أفكار برمجية منها.

لغة أوبجكت باسكال

لغة أوبجكت باسكال هي لغة تتيح البرمجة الهيكلية والبرمجة الكائنية الموجهة بدون تعقيد، فهي من أكثر لغات البرمجة سهولة، لذلك تم إستخدامها في التعليم وكمدخل للدارسين الجدد في مجال البرمجة.
وتتميز البرامج التي تنتج عن هذه اللغة بسرعة التنفيذ، وعدم إحتياجها لمكتبات خارجية للتشغيل أو آلة إفتراضية، حيث أن البرنامج الذي ينتج عن مترجماتها هو برنامج طبيعي Native من وجهة نظر نظام التشغيل، وهو يتعامل مع تلك البرامج مباشرة دون وسطاء، وهذا من أسباب سرعتها. لذلك يمكن أن تُستخدم في عدد كبير من البرامج دون إستثناء، فيمكن أن تُكتب بها نواة نظام تشغيل Kernel ، ومحركات قواعد بيانات database engines، وبرامج إتصالات، وبرامج إنترنت مختلفة، وحتى لغات برمجة (حيث أن دلفي ولازاراس تم تصميمهما بإستخدام اللغة نفسها). وهذه البرامج الناتجة لاتقل جودة وسرعة عن نظيرتها لغة C++.

فري باسكال

في هذا الكتاب سوف نقوم بشرح اللغة والأمثلة بإستخدام مترجم فري باسكال Free Pascal الذي تم إنتاجه في عام 2000، ومازال يتطور إلى الآن. وهو مترجم مناسب للتعليم والإستخدام في بيئة العمل على حد سواء، وذلك لأنه حر ومفتوح المصدر ويوجد في عدد كبير من أنظمة التشغيل، فهو بذلك يمكن إستخدامه بدون شروط أو حدود.
والدارس لهذه اللغة بواسطة فري باسكال يستطيع تطبيق مادرسه في نظام لينكس ووندوز وماكنتوش وغيرها من الأنظمة المعروفة التي يدعمها مترجم فري باسكال، والأنظمة الجديدة التي بدأ يدعمها مترجم فري باسكال مثل الآي فون I-Phone. و يمكن إستخدام هذا الكتاب مع لغة دلفي مع الإنتباه لبعض التعديلات البسيطة في المكونات أو بيئة التطوير.

لازاراس

بيئة لازاراس Lazarus هي أداة تطوير متكاملة تسخدم مترجم فري باسكال، حيث أنها محرر للكود، ومصمم للفورمات والأجزاء المختلفة في البرامج، ومتابع للأخطاء Debugger، كذلك فإنها بيئة متكاملة تحتوي على مكتبات ومكونات مرئية وغير مرئية تسمى LCL . وهي مشابهة لبيئة دلفي، إلا أنها تعتبر البديل الحر له.

ميزات فري باسكال

1. **الوضوح**: لغة باسكال من اللغات الواضحة والكود المكتوب بها سهل الفهم والصيانة مقارنة بنظيرتها لغة سي. والسهولة والوضوح تتميز بها هذه اللغة بدون التضحية بالإمكانات العالية.

2. **سرعة الترجمة**: تتميز مترجمات باسكال بسرعة الترجمة، وهي لاتحتاج إلى Makefiles. فنجد أن البرامج الكبيرة تتم ترجمتها في ثواني.

3. **أداة التطوير المتكاملة**: يصاحب مترجم باسكال أدوات تطوير متكاملة مثل لازاراس ودلفي، وهي أدوات واضحة وقوية تحتوي على كل مايحتاجه المبرمج لتطوير برامج باسكال.

4. **تكامل مع الأسمبلي**: تتميز لغة أوبجكت باسكال بإمكانية كتابة كود أسمبلي لتسريع جزء معين من الكود والتعامل المباشر مع المعالج.

5. **مترجم ذكي**: بعد عملية الترجمة وربط الوحدات، يتم التخلص من المتغيرات والإجراءات غير المستخدمة، ممايقلل حجم البرنامج الناتج.

6. **الإستقلال في توزيعات لينكس**: هذه الميزة تجعل برامج فري باسكال تعمل في كل توزيعات لينكس مما يسهل علمية الدعم الفني في هذا النظام.

7. **متوفر في عدد من أنظمة التشغيل مع إختلاف المعماريات**: يدعم مترجم فري باسكال عدد كبير من أنظمة التشغيل ومعماريات الحاسوب والمعالجات المختلفة.

8. **التوافقية مع أوبجكت باسكال**: نجد أن مترجم فري باسكال متوافق مع معظم الكود الموجود في الإنترنت، فإبتداءً من كود توربو باسكال إلى دلفي، نجد أن معظم الوحدات يمكن الإستفادة منها وإستخدامها في برامج فري باسكال.

ترخيص الكتاب

ترخيص الكتاب هو كسابقه **وقفي**، وذلك لزيادة وإثراء مكتبتنا العربية والإسلامية من ناحية المحتويات العلمية والصناعية. حيث أن البرمجة تعتبر من أهم أساسيات تقنية المعلومات، والتي هي صناعة في حد ذاتها، كما وتدخل في كثير من الصناعات الأخرى والمؤسسات الحكومية والشركات، ولها أثر إقتصادي وإداري ملموس عند تطبيقها بالطرق الصحيحة.
وإحدى فروقات الدول النامية والدول المتقدمة هي الصناعة. حيث توصف الدول النامية بالمستهلكة، والدول المتقدمة بالدول الصناعية.

المحتويات

مقدمة.....2
لغة أوبجكت باسكال.....2
فري باسكال.....2
لازاراس.....2
ميزات فري باسكال.....3
ترخيص الكتاب.....3

الفصل الأول

الذاكرة

Memory

مقدمة.....8
الذاكرة الإفتراضية.....8
الذاكرة والبرامج.....9
الذاكرة والمتغيرات.....11
الدالة *Move*.....13
المؤشرات *Pointers*.....15
حجز الذاكرة *Memory allocation*.....18
تشارك موقع الذاكرة.....20
محاذير في التعامل مع المؤشرات.....21
المؤشر غير محدد النوع *Pointer*.....24
القائمة المتصلة *Linked List*.....26
برنامج القائمة المتصلة:.....26
القائمة المتصلة ذات المؤشرين *Doubly linked list*.....40
المقاطع والذاكرة.....49
Short String.....49
Null terminated string.....50
Ansi String.....51
الكائنات والمكونات *Objects and Components*.....54

الفصل الثاني

إدارة الملفات

Files Management

مقدمة.....56
إستعراض الملفات.....56
برنامج إستعراض دليل.....57
برنامج عدد الأسطر في برنامج باسكال.....57
برنامج حجم المجلد *Directory size*.....59
برنامج النسخ الذكي.....62

الفصل الثالث

قواعد البيانات العلائقية

Relational Databases

مقدمة.....68
قاعدة بيانات *FireBird*.....68
إحتياجات برامج قاعدة البيانات.....68
برنامج إدارة قاعدة البيانات *FireBird*.....69
برنامج المكتبة المدرسية.....69
إجراء البحث.....76
تعديل البيانات.....79
إستلاف وإرجاع الكتب.....81
طقم السجلات ثنائية الإتجاه *Bi Directional Record set*.....86
تقرير الكتب المُستلفة.....90
برنامج مرآب السيارات.....93
التقارير.....99
حزمة التقارير *Fortes*.....102
برنامج دفتر اليومية.....103
السرية في قاعدة البيانات *FireBird*.....117
نظم المحاسبة المالية.....119

الفصل الرابع

برامج الويب

Web Applications

مقدمة.....121
مخدم الويب *Web Server*.....122
برنامج الويب الأول.....123
برتوكول الـ *CGI*.....125
حزمة *Free Spider*.....125
برنامج *FreeSpider* الأول.....126
إستخدام المُدخلات.....128
إستخدام صفحة ثابتة.....129
المكون *Action*.....131
برنامج مستعرض الكتب.....132
طريقة *Get method*.....135
إستخدام الفورم.....135
طريقة *Post method*.....136
فورم إسبايدر *Spider Form*.....138
دورة حياة برنامج الـ *CGI*.....141
الكوكيز *Cookies*.....141
برنامج الأخبار.....144
فصل التصميم من البرنامج.....148

الفصل الخامس

برمجة إتصالات الشبكات

Socket programming

مقدمة.....152
حزمة *LNet*.....152
برنامج الأوامر *Commands*.....153
برنامج المحادثة.....158
بروتوكول *HTTP*.....163
برنامج طرفية *HTTP*.....163
برتوكول *FTP*.....165
برنامج *FTP CLient*.....165

الفصل الأول

الذاكرة

Memory

مقدمة

الذاكرة هي وعاء لتخزين البيانات بإستخدام الطاقة الكهربائية في الحاسوب. فعندما يكون الحاسوب في وضع تشغيل تستطيع الذاكرة الإحتفاظ بالبيانات فترة طويلة، وعندما يتم قطع الطاقة عن الحاسوب فإن هذه البيانات تزول نهائياً.
الذاكرة هي مكون ملموس Hardware من مكونات عتاد الحاسوب، وبدونها لايعمل الحاسوب أو أي برنامج عليه. وتعتبر الذاكرة من أقرب العتاد بالنسبة للمعالج Processor حيث يربطها به ناقل Bus يسمى ناقل البيانات، وهو يختلف حسب إختلاف نوع المعالج واللوحة الأم، فإذا كان المعالج واللوحة الأم من نوع 32 بت فإن سعة هذا الناقل يكون 32 بت، إي أربع بايتات، وإذا كان المعالج 64 بت فإن سعة هذا الناقل تكون 64 بت، إي ثماني بايتات. وهذه السعة هي مايستطيع المعالج نقله من وإلى الذاكرة في المرة الواحدة.
وتمتاز الذاكرة بسرعة عالية في التعامل مع المعالج بخلاف وحدات التخزين الأخرى مثل القرص الصلب أو القرص المدمج.

ماتقوم الذاكرة بتخزينه هو أحد أمرين:

1. كود: عند تشغيل البرنامج فإن نظام التشغيل يقوم بتحميله من القرص الصلب إلى الذاكرة ليتسنى للمعالج نقل تعليمات الكود إلى مسجلاته processor registers ليتم تنفيذها. ولايستطيع المعالج تنفيذ الكود إلا عندما يكون موجود في الذاكرة.

2. بيانات: تستخدم البرامج الذاكرة لتخزين أوقراءة البيانات، فإذا احتاج البرنامج لقراءة ملف نصي مثلاً من القرص الصلب، فإن قراءة الملف ماهي إلا عملية تحميله من القرص الصلب إلى الذاكرة لتصبح بيانات هذا الملف في شكل متغيرات نصيةً مثلاً يمكن إستعراضها للمستخدم والتعامل معها.

الذاكرة الإفتراضية

عند إمتلاء الذاكرة لانستطيع تشغيل برنامج آخر، لذلك لجأت معظم نظم التشغيل لتوسعة الذاكرة من القرص الصلب في تقنية تسمى الذاكرة الإفتراضية Virtual memory وهي تخصيص جزء ثابت أومتغير من مساحة القرص الصلب لإستخدامه كإمتداد للذاكرة. وبما أن التعامل مع القرص الصلب بطيء جداً مقارنة بالذاكرة، وبما أن المعالج لايستطيع تشغيل البرامج إلا إذا كانت في الذاكرة، فإن إستخدام مساحة من القرص الصلب كذاكرة هو ليس إستخدام مطلق بنفس طريقة الذاكرة الحقيقية، إنما يستخدم لنسخ أجزاء وصفحات من الذاكرة الحقيقة لا تسخدم حالياً وهذه العملية تسمى التبديل Swaping. فيمكن أن نفترض أننا قمنا بفتح محرر النصوص وكتبنا فيه مقال مثلاً، ثم قمنا بتصغير برنامج محرر النصوص وقمنا بفتح برنامج متصفح الإنترنت، فيمكن لنظام التشغيل بنسخ محتويات الذاكرة التي يستخدمها محرر النصوص إلى القرص الصلب ثم حذف برنامج محرر النصوص من الذاكرة حتى نستفيد من هذه الذاكرة في برنامج متصفح الإنترنت. وفي حالة تكبير أو إعادة إظهار برنامج محرر النصوص، يقوم نظام التشغيل بإعادة نسخ برنامج محرر النصوص من القرص الصلب – الذاكرة الإفتراضية – إلى الذاكرة الحقيقية مرة أخرى ليتسنى للمعالج الإستمرار في تشغيل هذا البرنامج.

وبمعنى آخر إذا كان الحاسوب يحتوي على ذاكرة 1 غيغابايت، يمكن لنظام التشغيل حجز 2 غيغابايت من القرص الصلب، لتصبح الذاكرة الإفتراضية للحاسوب هي 3 غيغابايت، وفي هذه الحالة يمكن لنظام التشغيل تشغيل عدد من البرامج في آن واحد تصل إلى 3 غيغابايت.

عملية تبديل صفحات الذاكرة من وإلى القرص الصلب تأخذ وقت بالنسبة للمعالج، لذلك فإن حجز مساحة من القرص الصلب لتصبح ذاكرة إفتراضية يجب أن تكون بتناسب مع حجم الذاكرة الحقيقية،

حيث لايجب أن نجعل حجم الذاكرة الإفتراضية كبيراً جداً بالنسبة للذاكرة الحقيقة حتى لايتم تكرار عملية التبديل بشكل يُبطيء أداء الحاسوب.

الذاكرة والبرامج

عند تشغيل برنامج ما فإن هذا البرنامج تتم قراءته من القرص الصلب ثم تحميله إلى الذاكرة، وعندما يستقر البرنامج في الذاكرة يكون قد قُسم لذاكرته الخاصة (أو إقتطع جزء من الذاكرة) في شكل أربعة أجزاء أساسية:

1. **قطاع الكود Code segment** : وهو جزء ثابت من الذاكرة له حجم معروف عند تشغيل البرنامج لايتغير وهو الذي يتم تحميل الكود فيه، وفي حالة برامج باسكال فإن الكود هو عبارة عن مجموعة إجراءات مكتوبة بلغة الآلة يفهمها المعالج، وكمثال لها الإجراءات والدوال :

```
procedure X;
begin
  Writeln('This is a procedure X code');
end;
```

2. **قطاع البيانات Data segment**: وهو عبارة عن جزء ثابت لايتغير يحتوي على المتغيرات العامة Global variables والثوابت. والبيانات الموجودة في هذا الجزء يمكن أن تحتفظ بقيمتها أو مكانها إلى نهاية تنفيذ البرنامج. وكمثال لها قيمة X في البرنامج التالي:

```
program test;
var
  X: Integer;
begin
  Write('Input x: ');
  Readln(X);
  Writeln(X);
end.
```

3. **المكدسة Stack :** وهي جزء من الذاكرة يتغير حجمها أثناء تنفيذ البرنامج، وتعتبر ذاكرة مؤقتة للمتغيرات، حيث أن متغيراتها لاتلبث سوى وقت وجيز في هذا الجزء . وهي تحتوي على أنواع البيانات التالية:
   - المتغيرات المحلية للدوال والإجراءات
   - المدخلات للإجراءات والدوال
   - القيمة التي ترجع من الدالة

   وكمثال لها المتغيرات a, b, Result في المثال التالي:

```
function GetSumm(a, b: Integer): Integer;
begin
  Result:= a + b;
end;

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

4. **الكومة Heap**: وهي ذاكرة مؤقته تُستخدم لتخزين المتغيرات الديناميكية (التي يمكن حجز مساحة لها أو تحريرها أثناء تشغيل البرنامج). وحجم هذا الجزء من الذاكرة بالنسبة للبرنامج هو ماتبقى من الذاكرة في الحاسوب، إلا أنه ينقص عند تشغيل برنامج آخر، فهي ذاكرة تتشارك فيها البرنامج التي تعمل في آن واحد في نظام التشغيل (المقصود بالمشاركة حجز حيز من هذه الذاكرة، وليس المشاركة لنفس الموضع). وكمثال للبيانات التي تستخدم الكومة هي المقاطع الغير محددة الطول، والمصفوفة المرنة أو الديناميكية ، والمكونات مثل TStringList وغيرها كما في المثال التالي:

```
var
  x: array of Integer;
  MyName: string;
  i: Integer;
begin
  MyName:= 'Motaz'; // This will allocate value in heap
  SetLenth(x, 10); // This will allocate 10 cells in heap

  // Set Data
  for I:= 0 to High(x) do
    x[i]:= Random(100);

  // Display data
  for I:= 0 to High(x) do
    Writeln('X[', I, '] = ', X[i]);
  Write('Press enter key to close');

  SetLenth(x, 0); // This will free X from heap
  Readln;
end.
```

نلاحظ من هذا التقسيم أن هناك نوعان من أجزاء الذاكرة بالنسبة للبرنامج لايتم تغير مساحتها أثناء التشغيل، وهي جزء الكود وجزء البيانات، وهناك جُزآن آخران يتغير حجمها زيادةً ونقصاناً أثناء التنفيذ، وهي المكدسة والكومة. فعندما يتم حجز مساحة لمصفوفة مرنة فإن حجم الكومة المستخدمة بالنسبة للبرنامج سوف يزيد، وحجم الكومة التي تراها باقي البرامج أو نظام التشغيل ينقص، لأن البرنامج الأول قام بحجز مساحة جديدة منها: كما في المثال:

```
  SetLenth(x, 10); // This will allocate 10 cells in heap
```

كذلك فإنه عند الإنتهاء من إستخدام هذه المصفوفة وتحريرها، فإن الجزء المستخدم من الكومة ينقص بالنسبة للبرنامج، ويزيد بالنسبة لباقي البرنامج، ويمكن تحرير هذه المصفوفة وحذفها من الكومة بإستخدام الإجراء:

```
  SetLenth(x, 0); // This will free X from heap
```

أو يمكن تحريرها بطريقة أخرى:

```
  x:= nil; // This will free X from heap
```

أما بالنسبة للمكدسة فإن البرنامج يقوم بحجز غرفة منها عند نداء دالة، وهذا الحجز يكون أيضاً على حساب الكومة، حيث أن الكومة تتأثر بعدة عوامل منها: حجز مساحة لمتغير داينميكي، نداء دالة (حجز مكدسة) أو حتى تشغيل برنامج آخر، فكل هذه العوامل تقوم بإنقاص المساحة المتبقية من الكومة. وعند الفراغ من نداء الدالة والحصول على النتيجة، فإن هذه المكدسة المؤقتة يتم تحريرها من الذاكرة

ليتسنى إستخدام هذا المساحة بواسطة دالة أو إجراء آخر.
يتم تحرير الذاكرة المستخدمة من قبل المكسدة تلقائياً عند الفراغ من نداء الدوال والإجراءات، أما بالنسبة للكومة Heap فعلى المبرمج أن يفعل ذلك صراحة بعد الفراغ من إستخدام متغيراته، فإذا لم يفعل ، يحدث مايسمى بالتسرب في الذاكرة Memory leak وهي أن يقوم المبرمج أو البرنامج بحجز مواقع من ذاكرة الكومة دون أن يعيدها بعد الإنتهاء منها، وفي حالة البرامج التي تعمل لفترات طويلة، أيام أو أسابيع، فإن تراكم هذه الظاهرة يتنج عنه إمتلاء جميع ذاكرة الحاسوب وتظهر رسالة Out of memory. لذلك يجب الحرص عند التعامل مع المتغيرات المرنة أو الديناميكية أو الكائنات.

الذاكرة والمتغيرات

تتشكل المتغيرات في الذاكرة حسب حجمها، وحسب المكان الذي تم فيه تعريف المتغير. فمثلاً المتغيرات الصحيحة المعرفة محلياً في الإجراءات يتم حجز مساحة لها في المكدسة، وأما إذا كانت معرفة على مستوى البرنامج العام، فإنها تكون في قطاع البيانات.

في المثال التالي قمنا بتعريف ثلاث متغيرات a, b, c من نوع العدد الصحيح بايت:

```
var
  a, b, c: Byte;
begin
  a:= 10;
  b:= 13;
  c:= 4;
end.
```

فنجد أن شكلها في الذاكرة سوف يكون كالآتي:

10
13
4

حيث أن كل متغير قد احتاج خانة من الذاكرة.

أما في المثال التالي عند إستخدام متغيرات من نوع رمز char :

```
var
  a, b, c: Char;
begin
  a:= 'A';
  b:= 'B';
  c:= 'Z';
  Writeln(a, ' in memory = ', Ord(a));
  Writeln(b, ' in memory = ', Ord(b));
  Writeln(c, ' in memory = ', Ord(c));
  Readln;
end.
```

نجد أنها في الذاكرة توجد في شكل أرقام صحيحة (بايت) وهو مايسمى بالآسكي كود ASCII Code وهو القيمة الحقيقة التي ترمز للحروف أو الرموز، فمثلاً الحرف A قيمته في الأسكي كود هو 65.
نجد أن هذه المتغيرات في الذاكرة سوف تكون على الشكل التالي:

65
66
90

والخلية من الجدول أعلاه تمثل بايت أو خانة واحدة من الذاكرة، أي أن جملة المتغيرات في البرنامج التي قمنا بتعريفها إحتلت ثلاث خانات أو بايتات من الذاكرة.

أما إذا قمنا بتغير نوع المتغيرات إلى Word وهو متغير صحيح يحتاج لخانتين في الذاكرة كما في المثال التالي:

```
var
  a, b, c: Word;
begin
  a:= 100;
  b:= 50;
  c:= 9;
  Writeln(a, ' in memory = ', Lo(a), '-', Hi(a));
  Writeln(b, ' in memory = ', Lo(b), '-', Hi(b));
  Writeln(c, ' in memory = ', Lo(c), '-', Hi(c));
  Readln;
end.
```

نجد أنها هذه المرة تحتل ست خانات في الذاكرة وتكون على الشكل التالي:

100
0
50
0
9
0

والدوال Lo و Hi تقوم بإظهار بايت واحد من المتغير، فالدالة الأولى تقوم بإظهار البايت الأصغر قيمة Less significant والدالى الثانية تقوم بإظهار البايت الأعلى most significant. وفي الحالة السابقة نجد أن القيمة الثلاث أصغر من القيمة 255، لذلك إحتاجت لبايت واحد وكان البايت الآخر دائماً صفر.

أما إذا قمنا بوضع قيم أكبر سوف تتغير محتويات الذاكرة:

```
  a:= 256;
  b:= 800;
  c:= 1024;
```

فيكون شكلها في الذاكرة كالآتي:

0
1
32
3
0
4

ولفهم هذه القيم نقوم بالآتي:
إضافة البايت الأولى إلى البايت الثاني مضروب في 2 مرفوع إلى الأس 8، وهو يمثل عدد البتات التي يحتويها البايت، وقيمتها 256
ففي المتغير b ذو القيمة 800 تكون المعادلة كالآتي:

```
32 + (3 * 256) = 800
```

الدالة Move

تستخدم الدالة Move لنسخ محتوى من الذاكرة إلى متغير أو مصفوفة. ومدخلها الأول هو المتغير المصدر الذي يُراد النسخ منه، والمدخل الثاني هو المتغير الهدف الذي يُراد النسخ إليه، والمتغير الثالث والأخير هو الحجم بالبايت الذي يراد نسخه، مثلاً:

```
var
  Large: Integer;
  x: array [0 .. 3] of Byte;
  i: Integer;
begin
  Large:= 12345678;
  Move(Large, x, 4);
  for i:= 0 to 3 do
    Writeln('Byte # ', i, ' = ', x[i]);
  Readln;
end.
```

كذلك يمكن نسخ متغيرات من أنواع مختلفة مثلاً:

```
var
  x: array [0 .. 4] of Char;
  int: array [0 .. 4] of Byte;
  i: Integer;
begin
  x[0]:= 'M';
  x[1]:= 'o';
  x[2]:= 't';
  x[3]:= 'a';
  x[4]:= 'z';
```

```
  Move(x, Int, SizeOf(Int));

  for i:= 0 to 4 do
    Writeln(Int[i]);
  Readln;
end.
```

الدالة المستخدمة في المثال السابق SizeOf تقوم بإرجاع عدد خانات الذاكرة المستخدمة بواسطة متغير أو نوع، ففي المثال السابق سوف ترجع 4 وهو عدد الخانات بالبايت للمتغير Int.

المؤشرات Pointers

توجد طريقة غير مباشرة لتخزين المتغيرات وذلك بإستخدام المؤشرات. لكن قبل الشروع في إستخدام المتغيرات، دعنا نرجع إلى أنواع المتغيرات التي استخدمناها من قبل، مثل الأعداد الصحيحة. فمثلاً هذا التعريف:

```
var
  i: Integer;
```

نجد أنه يخبر البرنامج بحجز موقع في الذاكرة لمتغير صحيح، وهذا الموقع حجمه أربع بايتات.

وعندما نقوم بوضع العدد 100 مثلاً في هذا المتغير، سوف يتم وضعه في المكان الذي تم حجزه في بداية البرنامج للمتغير I. ونحن لانهتم كثيراً لعنوان الذاكرة Address الذي يشغله هذا المتغير،فقط نهتم بالقيمة الموجودة في ذلك العنوان.

أما إذا قمنا بمحاولة كتابة عنوان الذاكرة الذي يمثل المتغير I فإننا نستخدم الرمز @ قبل بداية المتغير مثل I@ وهو يعني عنوان المتغير I وليس قيمته كما في المثال التالي:

```
var
  i: Integer;
begin
  i:= 100;
  Writeln('Value of i = ', i);
  Writeln('Address of i = ', Integer(@i));
  Readln;
end.
```

وفي المثال التالي قمنا بإضافة متغير جديد j والذي سوف يتم تخزينه مباشرة بعد موقع المتغير I:

```
var
  i, j: Integer;
begin
  i:= 100;
  j:= 200;
  Writeln('Value of i = ', i);
  Writeln('Value of j = ', j);
  Writeln('Address of i = ', Integer(@i));
  Writeln('Address of j = ', Integer(@j));
  Readln;
end.
```

نلاحظ أن عنوان j يبعد عن عنوان I بأربع وحدات.

في المثال التالي سوف نقوم بالتعريف عن المتغير Pi صراحة على أنه متغير من النوع (مؤشر) وهو يؤشر على نوع من العدد الصحيح:

```
var
  i: Integer;
  Pi: ^Integer;
begin
  i:= 100;
  Pi:= @i;
  Writeln('Value of i = ', i);
  Writeln('Address of i = ', Integer(@i));
  Writeln('Value of Pi  = ', Integer(Pi));
  Readln;
end.
```

نلاحظ أن طريقة التعريف لمؤشر يؤشر إلى عدد صحيح هي كالآتي:

```
  Pi: ^Integer;
```

ونلاحظ أننا قمنا بإسناد موقع الذاكرة للمتغير I للمؤشر Pi. وعند طباعة قيمة Pi نجد أنها مساوية تماماً لموقع عنوان المتغير I.

قمنا بإضافة السطر التالي للمثال السابق وذلك لكتابة قيمة المتغير I بدلالة المؤشر Pi.

```
  Writeln('Value of data pointed by Pi  = ', Pi^);
```

نجد أن القيمة الناتجة هي 100. ممايعني أن عبارة Pi^ تعني قيمة مايؤشر له المؤشر Pi وهو في هذه الحالة نفس قيمة المتغير I.

في المثال التالي سوف نقوم بإضافة متغير جديد j ومؤشر له Pj:

```
var
  i: Integer;
  j: Integer;
  Pi: ^Integer;
  Pj: ^Integer;
begin
  i:= 100;
  j:= 200;
  Pi:= @i;
  Pj:= @j;
  Writeln('Value of data pointed by Pi  = ', Pi^);
  Writeln('Value of data pointed by Pj  = ', Pj^);
  Readln;
end.
```

نجد أن شكل هذه المتغيرات في الذاكرة سوف يكون كالآتي:

محتوى الذاكرة	العنوان	إسم المتغير
100	1002030	I
200	1002034	J
1002030	1002038	Pi
1002034	1002042	Pj

إفترضنا أن بداية تخزين البيانات للبرنامج السابق هو 1002030، وهو مثال فقط لشكل موقع الذاكرة حتى يتسنى الفهم. لكن سوف يجد المبرمج رقم مختلف عند طباعة قيمة الذاكرة للمتغيرات السابقة. ونجد كذلك أن المتغيرات Pi, Pj تحتل هي نفسها موقع في الذاكرة، وهي تحتاج إلى أربع بايتات للإشارة إلى عنوان من النوع 32bit.

والجدول التالي سوف يوضح العبارات المختلفة التي يمكن إستخدامها مع المتغيرات السابقة:

قيمتها	العبارة
100	I
200	J
1002030	@I
1002034	@J
1002030	Pi
1002034	Pj
100	Pi^
200	Pj^

حجز الذاكرة Memory allocation

في الأمثلة السابقة لم نقم بحجز محل مخصص للمؤشرات Pi, Pj إنما إستخدمنا عناوين جاهزة ومتسخدمة وهي عناوين المتغيرات I, J . لكن يمكن التعامل مباشرة لحجز وإطلاق مواقع في الذاكرة للمتغيرات. وذلك بإستخدام الإجراء New الذي يقوم بحجز مكان في ذاكرة الكومة Heap ثم يقوم بإرجاع عنوان هذا الموقع في المتغير من النوع (مؤشر)، كما في المثال التالي:

```
var
  Pi: ^Integer;
  Pj: ^Integer;
begin
  New(Pi);
  New(Pj);
  Pi^:= 100;
  Pj^:= 200;
  Writeln('Value of data pointed by Pi  = ', Pi^);
  Writeln('Value of data pointed by Pj  = ', Pj^);
  Dispose(Pi);
  Dispose(Pj);
  Readln;
end.
```

نلاحظ أن الإجراء New قام بحجز موقع من أربع بايتات (حسب حجم المتغير الذي يؤشر له) في الكومة. وبعد ذلك أمككنا إستخدامها (وقبل ذلك كان لايمكن إستخدام أو وضع بيانات في المكان الذي يؤشر إليه Pi, Pj ، وذلك لأنها كانت تؤشر إلى لاشيء nil).
وبعد الفراغ من إستخدام هذه المؤشرات، يجب نداء الإجراء Dispose والذي يقوم بتحرير الذاكرة المحجوزة حتى يستنى الإستفادة منها لاحقاً.
ومن هذا المثال نجد أن المؤشرات تتيح للمبرمج الإستخدام الأمثل للذاكرة، فبدلاً من حجز جزء كبير من الذاكرة طوال حياة البرنامج، يمكن أن نقوم بتحويل هذا الجزء إلى مؤشر ثم نقوم بحجزه فترة إستخدامه فقط، ثم نقوم بإطلاقه وتحرير الذاكرة المستخدمة من قِبله.

في المثال التالي قمنا بتحويل برنامج لقراءة ملفات وعرض محتوياتها بطريقة ثنائية. قمنا بتحويل المصفوفة Block إلى مؤشر للمصفوفة PBlock:

```
program ReadContents;

{$mode objfpc}{$H+}

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


type
  TBlock = array [0 .. 1023] of Byte;
```

```
var
  FileName: string;
  F: file;
  PBlock: ^TBlock;
  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);

    New(PBlock);    // Allocate memory space for PBlock (size = 1k)

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

    Dispose(PBlock);    /// Free/Release memory allocated by PBlock (The 1k)
    CloseFile(F);

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

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

نلاحظ في البرنامج السابق النقاط المهمة التالية:

1. قمنا بتعريف نوع جديد TBlock وهو نوع يمثل مصفوفة من النوع بايت طولها كيلو بايت.

```
type
      TBlock = array [0 .. 1023] of Byte;
```

2. قمنا بتعريف مؤشر PBlock يؤشر لمتغيرات (مصفوفة) من النوع TBlock

```
  PBlock: ^TBlock;
```

3. قمنا بحجز المصفوفة التي تحتل كيلو بايت قبل بداية إستخدامها مباشرة (قبل الشروع في قراءة محتويات الملف)

```
  New(PBlock);    // Allocate memory space for PBlock (size = 1k)
```

4. قمنا بتحرير الذاكرة المحجوزة (كيلو بايت) بعد الفراغ من إستخدامها مباشرة

```
  Dispose(PBlock);     /// Free/Release memory allocated by PBlock (The 1k)
```

5. للوصول لمتغير في مصفوفة بإستخدام المؤشرات نستخدم الطريقة التالية:

```
  PBlock^[i]
```

في حالة عدم إستخدام المؤشرات وإستخدامنا لمصفوفة من البايتات، فإنها سوف يتم حجزها في جزء البيانات data segment طوال فترة تشغيل البرنامج، فإذا افترضنا أن البرنامج يعمل لمدة ساعة، فإن الحجز يدوم ساعة كاملة. أما بإستخدام المؤشرات فإن الحجز يتم فقط في فترة قراءة محتويات الملف فقط، والتي ربما تكون ثواني فقط، وبذلك نكون حققنا إستخدام أمثل للذاكرة.

تشارك موقع الذاكرة

يمكن لأكثر من مؤشر أن يؤشر إلى نفس الموقع، وذلك كما في المثال التالي:

```
var
  Pi: ^Integer;
  Pj: ^Integer;

begin
  New(Pi);

  Pi^:= 100;

  Pj:= Pi; // Pj and Pi are sharing the memory address

  Writeln('Value of data pointed by Pi  = ', Pi^);
  Writeln('Value of data pointed by Pj  = ', Pj^);

  Dispose(Pi); // you should dispose only one pointer,
               // because we have only one real address
  Readln;
end.
```

في هذه الحالة كلا المؤشرين يؤديان إلى نفس الموقع في الذاكرة (القيمة 100)، لكن يجب الحذر عند تحرير هذا الموقع، إذ يجب فقط تحرير مؤشر واحد فقط إما

```
 Dispose(Pi);
```

أو

```
 Dispose(Pj);
```

فإذا حاولنا تحرير الثاني وجدنا أنه لايؤشر إلى مكان محجوز وتحدث في هذه الحالة Access Violation

محاذير في التعامل مع المؤشرات

تعتبر طريقة التعامل مع المؤشرات طريقة غير آمنة للتعامل مع الذاكرة unsafe pointers. وذلك ناتج عن أن عملية حجز وتحرير الذاكرة يُترك على عاتق المبرمج، فإذا أخطأ فإن أقل مشكلة تحصل هي رسالة الخطأ الشهيرة (Access Violation) والتي تعني محاولة قراءة أو الكتابة في ذاكرة ليست لهذا البرنامج.
كذلك فإنه عند نسيان تحرير الذاكرة بعد إستخدامها أيضاً يمكن أن يحدث مايعرف بالتسرب في الذاكرة (Memory leak).
لذلك يجب على المبرمج أن يفهم المؤشرات والذاكرة فهم عميق حتى يستطيع كتابة برامج ليست بها المشاكل السابقة.

ملخص المشاكل والحلول هي كالآتي:

1. **المؤشر المتدلي Dangling pointer**: ,وهو المؤشر الذي لم تتم تهيئته (حجز موقع له في الذاكرة) أو تم تحريره.
وهذه المشكلة تكمن في نسيان المبرمج إستخدام الإجراء New ومحاولة التعامل مباشرة مع المكان الذي يؤشر إليه هذا المؤشر. أو إستخدام الدالة Dispose ثم محاولة إستخدامه.
ينتج عن هذه المشكلة رسالة الخطأ Access Violation وهي ربما تتسبب في إغلاق البرنامج وضياع البيانات غير المحفوظة.
يكمن حل هذه المشكلة في إسناد القيمة nil للمؤشر الذي لايؤشر إلى شيء. والقيمة nil بالنسبة للمتغيرت تعني أن المؤشر لايوشر إلى شيء، وعلى المبرمج فحص قيمة المؤشر قبل إستخدامه كما في المثال التالي:

```
var
  Pi: ^Integer;
  Pj: ^Integer;

begin
  Pi:= nil;
  Pj:= nil;
  New(Pi);
  //New(Pj);

  if Pi <> nil then
    Pi^:= 100;

  if Pj <> nil then
    Pj^:= 200;

  if Pi <> nil then
    Writeln('Value of data pointed by Pi  = ', Pi^)
  else
    Writeln('Pi is not initalized');

  if Pj <> nil then
    Writeln('Value of data pointed by Pj  = ', Pj^)
  else
    Writeln('Pj is not inialized');

  if Pi <> nil then
  begin
    Dispose(Pi);
```

```
    Pi:= nil;
  end;

  if Pj <> nil then
  begin
    Dispose(Pj);
    Pj:= nil;
  end;
  Readln;
end.
```

نلاحظ في المثال السابق أننا إذا قمنا بتعطيل تهيأة المؤشر Pj فإن ذلك لن يتسبب في حدوث خطأ وذلك لأننا نقوم بالتحقُق من أن المؤشر لايحمل القيمة nil قبل استخدامه، وبذلك نكون قد تجاوزنا هذه المشكلة.
عند تحرير المؤشر بإستخدام Dispose فإن الذاكرة المحجوزة يتم تحريرها ويمكن إستخدامها بواسطة متغيرات أو مؤشرات أخرى، لكن هذا الإجراء لايقوم بوضع القيمة nil في هذا المؤشر وعند محاولة التعامل مع مايؤشر له تحدث مشكلة access violation، لذلك بعد إستخدام Dispose يجب وضع القيمة nil في ذاك المؤشر إذا كان سوف يُستخدم مرة ثانية في البرنامج. أما إذا كان البرنامج مشرف على نهاية، كما في المثال السابق، فلامشكلة في عدم تحريره أو وضع nil. لأنه بنهاية البرنامج يتم تحرير كافة المؤشرات وكل ماتم حجزه من الذاكرة بواسطة البرنامج حتى لو نسى المبرمج تحرير المؤشرات صراحة.
يحدث رفع الإستثناء access violation عند محاولة الوصول لمؤشر متدلى يؤشر إلى ذاكرة محمية (يمكلها برنامج آخر، أو ذاكرة غير محجوزة). أما إذا كان المؤشر المتدلى يؤشر إلى عنوان كان له من قبل (تم تحريره بواسطة Dispose)، أو لعنوان محجوز بواسطة نفس البرنامج فإن محاولة الوصول والتعامل مع المكان الذي يؤشر إليه لاينتج عنه هذا الخطأ، إنما يحدث تغيير في بيانات في نفس البرنامج ربما يمكلها متغير آخر، والخطأ الأخير هو الأخطر، لأن access violation تمنع المؤشر المتدلي من التأثير في الذاكرة التي لايمكلها البرنامج الحالي، أما الحالة الثانية فالخطأ لايمكن معرفته بسهولة ولاينتج عنه إستثناء، إنما تضيع بيانات أويحدث تغييرات تكون لها تأثيرات في بيانات أخرى لايمكن معرفتها بسهولة.

2. **العنقود الضائع Lost cluster**: وهي مشكلة تؤدي إلى تسرب في الذاكرة Memory leak. ومعناها حجز موقع في الذاكرة ثم تغيير عنوان المؤشر بدون تحرير الموقع الأول. ويمكن حدوثها بأكثر من طريقة مثل:

الطريقة الأولى: تهيئة المؤشر أكثر من مرة:

```
  New(Pi); // Data pointed to by current Pi will be lost cluster and can not
           // be accessed or released until closing the application
  New(Pi); // This procedure causes the lost cluster
```

في المرة الأولى تم حجز أربع بايتات في عنوان ما، مثلاً نفرض أنه العنوان 10034، وتم حفظ قيمة هذا العنوان في المتغير Pi.
وفي المرة الثانية، سوف يتم حجز موقع آخر (4 بايتات) مثلاً نفترض 10038 ثم يتم حفظ هذا الموقع في المتغير Pi أيضاً مما يتسبب عنه ضياع العنوان الأول 10034. و الموقع الأول الذي حجز 4 بايتات لم يتم تحريرها، ولايستطيع البرنامج تحريرها لتعذر الوصول إليها، حيث أن الطريقة الوحيدة للوصول

للموقع الأول كانت عن طريق المؤشر Pi لكنه فقد هذا العنوان وأصبح يؤشر إلى عنوان جديد. بتكرار هذه الطريقة أثناء تشغيل البرنامج يحدث عدد كبير من العناقيد الضائعة مما يتسبب في إستهلاك الذاكرة بدون فائدة ونحتاج لإغلاق البرنامج ثم تشغيله من جديد لحل هذه المشكلة. هذه طريقة حل من وجهة نظر المستخدم، أما من وجهة نظر المبرمج، فعليه التأكد من أن عملية التهيئة تتم مرة واحدة، أو بعد تحرير. فلو عدلنا الكود إلى المثال التالي سوف نتجاوز هذه المشكلة:

```
New(Pi);
Dipose(Pi);

New(Pi);

// Later
Dispose(Pi);
```

بهذه الطريقة نضمن أن العنوان الأول تم تحريره ويمكن إستخدامه مع مؤشرات أخرى أو نفس المؤشر مرة أخرى بدون ضياعه.

الطريقة الثانية: تهيئة مؤشر ثم تغيير العنوان الذي يؤشر له: وذلك كما في المثال التالي:

```
New(Pi);
New(Pj);

Pi^:= 100;
Pj^:= 200;

Pj:= Pi; // This causes lost cluster for Pj
```

نجد أنه بعد ماتم حجز موقع في الذاكرة للمؤشر Pj فإننا قمنا بتغيير مايؤشر له Pj إلى Pi. وبالتالي سوف تضيع الذاكرة التي تحتوي على القيمة 200 ويصبح Pi, Pj يؤشران لنفس العنوان الذي فيه القيمة 100.
ولحل هذه المشكلة يجب عدم تهيئة المؤشر Pj والإكتفاء فقط بتهيئة Pi والذي سوف تتم مشاركته مع المؤشر Pj كما في التعديل التالي:

```
New(Pi);
Pi^:= 100;
Pj:= Pi;
```

الطريقة الثالثة: وضع القيمة nil بعد تهيئة مؤشر، كما في المثال التالي:

```
New(Pi);
Pi^:= 100;
Pi:= nil; // New the value 100 will be in lost cluster
```

نجد أن وضع القيمة nil في المؤشر لاتقوم بتحرير الذاكرة التي قام بحجزها، إنما تعني فقط أن هذا المؤشر يؤشر إلى لاشيء (nil).
ولحل هذه المشكلة يجب دائماً تحرير المؤشر الذي قمنا بتهيأته بإستخدام New وذلك بإستخدام Dispose قبل وضع nil:

```
New(Pi);
Pi^:= 100;
```

```
  Dispose(Pi);
  Pi:= nil;
```

المؤشر غير محدد النوع Pointer

يمكن تعريف مؤشر ما يكون غير مرتبط بأي نوع من البيانات. المؤشرات التي إستخدامناها سابقاً مثل Pi, Pj هي مؤشرات تُستخدم فقط مع نوع البياناتات Integer. وإذا أردنا إستخدام نوع آخر مثلاً الأعداد الحقيقية، لابد من تغيير التعريف إلى مؤشر لعدد حقيقي مثل:

```
  Pd: ^Double;
```

أما المؤشر الغير محدد النوع فيمكن إستخدامه مع أي بيانات شرط أن نعرف حجم هذه البيانات، مثلاً:

```
var
  MyPointer: Pointer;
  i: Integer;
begin
  i:= 1024;
  MyPointer:= GetMem(4);

  Move(i, MyPointer^, 4);
  Writeln('MyPointer point the value: ', Integer(MyPointer^));
  FreeMem(MyPointer);
  Readln;
end.
```

نجد من البرنامج السابق الآتي:

1. قمنا بالتعريف عنه بالطريقة التالية:

```
var
  MyPointer: Pointer;
```

2. قمنا بحجز مساحة له في الذاكرة بإستخدام الدالة GetMem وليس New. والإختلاف يكمن في أن الدالة New تستخدم مع المؤشرات معروفة النوع، وبالتالي معروفة الحجم، أما GetMem فهي تحتاج إلى مُدخل إضافي هو عبارة عن حجم البيانات المراد حجزها من ذاكرة الـ Heap.

3. لايمكن كتابة وقراءة محتويات هذا المؤشر مباشرة. فقد قمنا بإسناد قيمة له بإستخدام الإجراء Move والذي يقوم بنسخ محتويات ذاكرة بطول معين.

4. كذلك قمنا بقراءة محتويات MyPointer وكتابتها في الشاشة بعد النظر لها على أنها عدد صحيح:

```
Integer(MyPointer^)
```

5. قمنا بتحرير الذاكرة المستخدمة للمؤشر بإستخدام الإجراء FreeMem

في المثال التالي قمنا بكتابة إجراء لتبديل قيمتين، الميزة هذه المرة أن هذا الإجراء يمكنه أن يعمل مع كل أنواع البيانات، فقط نخبره بحجم البيانات فيقوم بعملية التبديل:

```
program PointerSwap;

{$mode objfpc}{$H+}

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


procedure SwapData(var x, y; DataSize: Byte);
var
  Temp: Pointer;
begin
  Temp:= GetMem(DataSize);
  Move(X, Temp^, DataSize);
  Move(Y, X, DataSize);
  Move(Temp^, Y, DataSize);
  FreeMem(Temp);
end;

var
  a, b: Integer;
  c, d: Double;
begin
  a:= 10;
  b:= 20;
  c:= 1.2;
  d:= 5.4;

  SwapData(a, b, SizeOf(a));

  SwapData(c, d, SizeOf(c));

  Writeln('a = ', a);
  Writeln('b = ', b);
  Writeln('c = ', c);
  Writeln('d = ', d);

  Readln;
end.
```

نلاحظ في البرنامج السابق أننا إستخدمنا طريقة جديدة لتعريف المدخلات، وهي المدخلات غير معرفة النوع:

```
procedure SwapData(var x, y;
```

وهي تستطيع إستقبال أي نوع من البيانات، وهذه الطريقة تصلح فقط للإستخدام مع مدخلات الإجراءات أو الدوال، ولايمكن تعريفها مستقلة كمتغيرات عادية.

القائمة المتصلة Linked List

القائمة المتصلة هي من أشهر مواضيع هيكلية البيانات Data Structure . وذكرناها في هذا الفصل لأنها تستخدم المؤشرات. ولها خوارزمية ذكية وكتابتها ممتعة.
وهي عبارة عن سلسلة من البيانات غير محدودة الطول، فهي تتمدد بزيادة البيانات وتتقلص بنقصانها.
فهي تشبه المصفوفة المرنة Dynamic Array لكنها أكثر مرونة منها، حيث نجد أن المصفوفة المرنة تتم الزيادة فيها عند طرف واحد، عند النهاية، كذلك الحذف أو تقليص طولها يتم فقط عند النهاية، ولايمكن إدخال البيانات بطريقة الإزاحة Insert في وسط البيانات لغرض الترتيب مثلاً. ونجد أن هذه الميزات (الإضافة في الوسط، أو في البداية) ممكنة في حالة القائمة المتصلة.

برنامج القائمة المتصلة:

لعمل قائمة متصلة بطريقة كائنية Object Oriented نقوم بالخطوات التالية:

1. نقوم بإنشاء وحدة جديدة نسميها LList مثلاً.
2. نقوم بتعريف عقدة Node والتي تحتوي على حاوية بيانات ومؤشر لعقدة بعدها:

```
type

   TNodePointer = ^TNode;
   TNode = record
          Data: Pointer;        // Any type of data
          Next: TNodePointer;   // Pointer to the next node
   end;
```

نلاحظ أن النوع TNode يمثل سجل يحتوي على الحقول Data, Next،
والنوع TNodePointer هو عبارة عن مؤشر للسجل السابق.
ونجد أنه من الغريب أننا قمنا بتعريف الحقل Next من نوع المؤشر TNodePointer، كأنما يؤشر لعقدة ثانية من نفس النوع، وهذا التعريف مسموح به في حالة المؤشرات.

3. نقوم بتعريف رأس القائمة أو السلسلة المتصلة ونسميه fHead وهو من النوع TNodePointer :

```
       fHead: TNodePointer;
```

نجد أن القائمة المتصلة يكون شكلها كالتالي:

	Node 1		Node 2		Node 3	
Head →	Data	Next →	Data	Next →	Data	Next → Nil

هذا كان مثال لقائمة متصلة بها ثلاث عقدات. حيث نجد أن الحقل Next للعقدة الأخيرة يؤشر إلى Nil مما يعني نهاية القائمة المتصلة، فإذا أردنا إضافة عقدة رابعة، فما علينا إلا حجز عقدة رابعة في الذاكرة ثم نجعل المؤشر Next في العقدة الأخيرة يؤشر للعقدة الجديدة.

4. قمنا بكتابة وحدة لكائن القائمة المتصلة بأبسط صورها كالتالي:

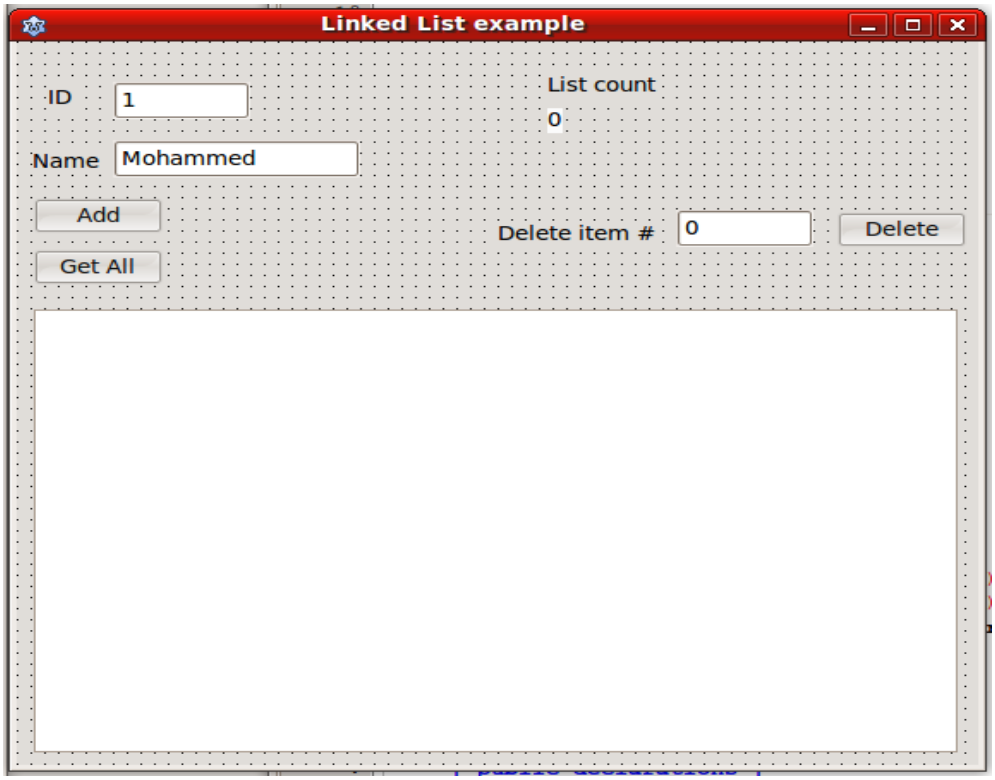
```
unit LList;

interface

type
   // TNodePointer : linked list node, contains one item (Data)
   // and pointer to the next node

   TNodePointer = ^TNode;
   TNode = record
          Data: Pointer;       // Any type of data
          Next: TNodePointer;  // Pointer to the next node
   end;


 //  Can by used with any data types such as Integer, Double, Records, etc

   { TLinkedList }

   TLinkedList = class
   private
       fHead: TNodePointer;
       fSize: Integer;
       fDataSize: Word;
       function GetItem(Index: Integer; var Pred: TNodePointer): TNodePointer;
       function Compare(p1, p2: Pointer): Byte;
   public
       constructor Create(ADataSize: Word);
       destructor Destroy;
       function Add(var Item): Integer;
       function Get(Index: Integer; var Item): Boolean;
       function Find(var Item): Integer;
       function Delete(Index: Integer): Boolean;
       function Count: Integer;
   end;


implementation

constructor TLinkedList.Create(ADataSize: Word);
begin
  fHead:= nil;
  fDataSize:= ADataSize;
  fSize:= 0;
end;

destructor TLinkedList.Destroy;
begin
  while fHead <> nil do
    Delete(0);
end;




(******************  Compare: Compare data  *******************)
```

```
function TLinkedList.Compare(p1, p2: Pointer): Byte;
type
  X = array [0 .. 1000] of Byte;
var
  i: Integer;
  First, Second: ^ X;
begin
  Result:= 0;
  First:= P1;
  Second:= P2;
  for i:= 0 to fDataSize - 1 do
  if First^[i] <> Second^[i] then
  begin
    if First^[i] > Second^[i] then
      Result:= 1
    else
      Result:= 2;
    Break;
  end;
end;

(***************  Count: Get linked list items total number  *****************)

function TLinkedList.Count: Integer;
begin
  Result:= fSize;
end;

(****************  GetItem: get item pointer (Internal use) **************)

function TLinkedList.GetItem(Index: Integer; var Pred: TNodePointer):
TNodePointer;
var
  i: Integer;
  Temp: TNodePointer;
begin
  Temp:= fHead;
  i:= 0;
  Result:= nil; // Default: Not found
  Pred:= nil;
  while Temp <> nil do
  begin
    if i = Index then
    begin
      Result:= Temp;
      Break;
    end;
    Inc(i);
    Pred:= Temp;
    Temp:= Temp^.Next;
  end;
end;
```

```
 (****************************** Add item ***********************************)

function TLinkedList.Add(var Item): Integer;
var
   Temp, Pred: TNodePointer;
begin
  if fHead = nil then // Empty list
  begin
    // allocate Head in memory (Heap)
    New(fHead);

    // allocate data in memory (Heap)
    GetMem(fHead^.Data, fDataSize);

    // Put item in the list
    Move(Item, fHead^.Data^, fDataSize);
    fHead^.Next:= nil;
    Result:= 0; // Addition Position
  end
  else  // Add in list tail
  begin
    Temp:= fHead;
    Result:= 0;
    // Search for last pointer
    while Temp <> nil do
    begin
      Inc(Result);
      Pred:= Temp;
      Temp:= Temp^.Next;
    end;

    // Now Temp becomes nil, end of list, Pred contains last item

    New(Temp);
    GetMem(Temp^.Data, fDataSize);
    Move(Item, Temp^.Data^, fDataSize);
    Temp^.Next:= nil;
    Pred^.Next:= Temp; // Link the new item with the previous one
  end;
  Inc(fSize);
end;

(********* Find: Find first occurrence of data ***************)

function TLinkedList.Find(var Item): Integer;
var
  Temp: TNodePointer;
  i: Integer;
begin
  Result:= -1; // default: not found
  Temp:= fHead;
  i:= 0;
  while Temp <> nil do
  begin

    if Compare(Temp^.Data, @Item) = 0 then
    begin
```

```
      Result:= i;
      Break;
    end;
    Inc(i);
    Temp:= Temp^.Next;
  end;
end;


(*************************   Delete node ***************************)

function TLinkedList.Delete(Index: Integer): Boolean;
var
  Temp, Pred: TNodePointer;
begin
  Temp:= GetItem(Index, Pred);
  Result:= Temp <> nil;
  if Result then
  if Temp = fHead then  (***  Found in head  ***)
  begin
    Temp:= fHead;
    fHead:= fHead^.Next; // New head: next item
    FreeMem(Temp^.Data, fDataSize);
    Dispose(Temp);
    Dec(fSize);
  end
  else
  begin // Normal item in the list
    Pred^.Next:= Temp^.Next;
    FreeMem(Temp^.Data, fDataSize);
    Dispose(Temp);
    Dec(fSize);
  end;
end;


(***********  Get data from node referenced by position Index  *************)

function TLinkedList.Get(Index: Integer; var Item): Boolean;
var
  Temp: TNodePointer;
  Pred: TNodePointer;
begin
  Temp:= GetItem(Index, Pred);
  Result:= Temp <> nil;
  if Result then
    Move(Temp^.Data^, Item, fDataSize);
end;

end.
```

5. قمنا بإنشاء برنامج جديد من نوع Application وأدرجنا فيه عدد من المكونات فأصبح بالشكل التالي:



6. قمنا بإضافة الوحدة LList إلى uses في الفورم الرئيسي للبرنامج.
7. قمنا بكتابة الكود التالي في الوحدة المصاحبة للفورم:

```
unit main;

{$mode objfpc}{$H+}

interface

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

type

  { TfmMain }

  TRec = record
    ID: Integer;
    AName: string[20];
  end;

  TfmMain = class(TForm)
    btAdd: TButton;
    btGetAll: TButton;
    btDelete: TButton;
    edID: TEdit;
    edDelete: TEdit;
    edName: TEdit;
    Label1: TLabel;
    Label2: TLabel;
```

```
    Label3: TLabel;
    Label4: TLabel;
    laCount: TLabel;
    meLog: TMemo;
    procedure btAddClick(Sender: TObject);
    procedure btDeleteClick(Sender: TObject);
    procedure btGetAllClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
    procedure FormCreate(Sender: TObject);
  private
    { private declarations }
  public
    List: TLinkedList;
    { public declarations }
  end;

var
  fmMain: TfmMain;

implementation

{ TfmMain }

procedure TfmMain.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  List:= TLinkedList.Create(SizeOf(TRec));
end;

procedure TfmMain.btAddClick(Sender: TObject);
var
  Rec: TRec;
begin
  Rec.ID:= StrToInt(Trim(edID.Text));
  Rec.AName:= edName.Text;
  List.Add(Rec);
  laCount.Caption:= IntToStr(List.Count);
end;

procedure TfmMain.btDeleteClick(Sender: TObject);
begin
  List.Delete(StrToInt(Trim(edDelete.Text)));
  laCount.Caption:= IntToStr(List.Count);
end;

procedure TfmMain.btGetAllClick(Sender: TObject);
var
  i: Integer;
  Rec: TRec;
begin
  meLog.Clear;

  for i:= 0 to List.Count - 1 do
  begin
    List.Get(i, Rec);
    meLog.Lines.Add('Item # ' + IntToStr(i));
```

```
    meLog.Lines.Add('ID   = ' + IntToStr(Rec.ID));
    meLog.Lines.Add('Name = ' + Rec.AName);
    meLog.Lines.Add('-------------------------');
  end;
end;

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

initialization
  {$I main.lrs}

end.
```

سوف نقوم بعمل بعض التعديلات والإضافات في الكائن TLinkedList وذلك لإضافة بعض الميزات مثل حفظ البيانات في ملف، والإضافة في وسط أو بداية القائمة. سوف نقوم بتسمية البرنامج List2:.
ويمكن الرجوع إلى البرنامج كاملاً في الأمثلة المصاحبة للكتاب.

الوحدة المعدلة:

```
unit LList; { Enhanced one }

interface

type
   // TNodePointer : linked list node, contains one item (Data)
   // and pointer to the next node

   TNodePointer = ^TNode;
   TNode = record
          Data: Pointer;       // Any type of data
          Next: TNodePointer;  // Pointer to the next node
   end;



 //  Can by used with any data types such as Integer, Double, Records, etc

   { TLinkedList }

   TLinkedList = class
   private
       fHead: TNodePointer;

       fSize: Integer;
       fDataSize: Word;
       function GetItem(Index: Integer; var Pred: TNodePointer): TNodePointer;
       function Compare(p1, p2: Pointer): Byte;
   public
       constructor Create(ADataSize: Word);
       destructor Destroy;
       function Add(var Item): Integer;
```

```
      function Get(Index: Integer; var Item): Boolean;
      function Find(var Item): Integer;
      function Delete(Index: Integer): Boolean;
      function Count: Integer;

      function Insert(Index: Integer; var Item): Boolean;
      function Save(AFileName: string): Boolean;
      function Load(AFilename: string): Boolean;
   end;


implementation

constructor TLinkedList.Create(ADataSize: Word);
begin
  fHead:= nil;

  fDataSize:= ADataSize;
  fSize:= 0;
end;

destructor TLinkedList.Destroy;
begin
  while fHead <> nil do
    Delete(0);
end;


(******************  Compare: Compare data  *******************)

function TLinkedList.Compare(p1, p2: Pointer): Byte;
type
  X = array [0 .. 10000] of Byte;
var
  i: Integer;
  First, Second: ^ X;
begin
  Result:= 0;
  First:= P1;
  Second:= P2;
  for i:= 0 to fDataSize - 1 do
  if First^[i] <> Second^[i] then
  begin
    if First^[i] > Second^[i] then
      Result:= 1
    else
      Result:= 2;
    Break;
  end;
end;

(***************  Count: Get linked list items total number  ******************)

function TLinkedList.Count: Integer;
begin
  Result:= fSize;
end;
```

```
function TLinkedList.Save(AFileName: string): Boolean;
var
  F: file;
  Temp: TNodePointer;
begin
  try
    AssignFile(F, AFilename);
    Rewrite(F, 1);

    Temp:= fHead;

    while Temp <> nil do
    begin
      BlockWrite(F, Temp^.Data^, fDataSize);
      Temp:= Temp^.Next; // Goto next item
    end;
    CloseFile(F);
    Result:= True; // Successful operation

  except
    Result:= False;
  end;
end;

function TLinkedList.Load(AFilename: string): Boolean;
var
  F: file;
  Temp: TNodePointer;
  Data: Pointer;
begin
  try
    AssignFile(F, AFilename);
    Reset(F, 1);

    Data:= GetMem(fDataSize); // Temporary buffer

    while Delete(0) do; // delete all current data in memory

    while not eof(F) do
    begin
      BlockRead(F, Data^, fDataSize);
      Add(Data^);
    end;
    FreeMem(Data);
    CloseFile(F);
    Result:= True; // Successful operation

  except
    Result:= False;
  end;

end;



(****************  GetItem: get item pointer (Internal use) **************)
```

```
function TLinkedList.GetItem(Index: Integer; var Pred: TNodePointer):
TNodePointer;
var
  i: Integer;
  Temp: TNodePointer;
begin
  Temp:= fHead;
  i:= 0;
  Result:= nil; // Default: Not found
  Pred:= nil;
  while Temp <> nil do
  begin
    if i = Index then
    begin
      Result:= Temp;
      Break;
    end;
    Inc(i);
    Pred:= Temp;
    Temp:= Temp^.Next;
  end;
end;

 (****************************** Add item **********************************)

function TLinkedList.Add(var Item): Integer;
var
   Temp, Pred: TNodePointer;
begin
  if fHead = nil then // Empty list
  begin
    // allocate Head in memory (Heap)
    New(fHead);

    // allocate data in memory (Heap)
    GetMem(fHead^.Data, fDataSize);

    // Put item in the list
    Move(Item, fHead^.Data^, fDataSize);
    fHead^.Next:= nil;
    Result:= 0; // Addition Position
  end
  else  // Add in list tail
  begin
    Temp:= fHead;
    Result:= 0;
    // Search for last pointer
    while Temp <> nil do
    begin
      Inc(Result);
      Pred:= Temp;
      Temp:= Temp^.Next;
    end;

    // Now Temp becomes nil, end of list, Pred contains last item
```

```
    New(Temp);
    GetMem(Temp^.Data, fDataSize);
    Move(Item, Temp^.Data^, fDataSize);
    Temp^.Next:= nil;
    Pred^.Next:= Temp; // Link the new item with the previous one
  end;
  Inc(fSize);
end;

  (****** Find: Find first occurrence of data **************)

function TLinkedList.Find(var Item): Integer;
var
  Temp: TNodePointer;
  i: Integer;
begin
  Result:= -1; // default: not found
  Temp:= fHead;
  i:= 0;
  while Temp <> nil do
  begin
    if Compare(Temp^.Data, @Item) = 0 then
    begin
      Result:= i;
      Break;
    end;
    Inc(i);
    Temp:= Temp^.Next;
  end;
end;

(*************************   Delete node ***************************)

function TLinkedList.Delete(Index: Integer): Boolean;
var
  Temp, Pred: TNodePointer;
begin
  Temp:= GetItem(Index, Pred);
  Result:= Temp <> nil;
  if Result then
  if Temp = fHead then  (***  Found in head  ***)
  begin
    Temp:= fHead;
    fHead:= fHead^.Next; // New head: next item
    FreeMem(Temp^.Data, fDataSize);
    Dispose(Temp);
    Dec(fSize);
  end
  else
  begin // Normal item in the list
    Pred^.Next:= Temp^.Next;
    FreeMem(Temp^.Data, fDataSize);
    Dispose(Temp);
    Dec(fSize);
  end;
end;
```

```
function TLinkedList.Insert(Index: Integer; var Item): Boolean;
var
  Temp, Pred: TNodePointer;
  CurrentItem: TNodePointer;
begin
  if (Index < 0) or (Index >= fSize - 1) then
    Result:= False
  else
  if Index = 0 then // Add before head
  begin
    New(CurrentItem);
    CurrentItem^.Data:= GetMem(fDataSize);
    Move(Item, CurrentItem^.Data^, fDataSize);
    CurrentItem^.Next:= fHead;
    fHead:= CurrentItem;
    Inc(fSize);
    Result:= True;
  end
  else // Insert in the middle of list
  begin
    Temp:= GetItem(Index, Pred);

    New(CurrentItem);
    CurrentItem^.Data:= GetMem(fDataSize);
    Move(Item, CurrentItem^.Data^, fDataSize);

    CurrentItem^.Next:= Temp;
    Pred^.Next:= CurrentItem;
    Inc(fSize);
    Result:= True;
  end;
end;

(****  Get data from node referenced by position Index **************)

function TLinkedList.Get(Index: Integer; var Item): Boolean;
var
  Temp: TNodePointer;
  Pred: TNodePointer;
begin
  Temp:= GetItem(Index, Pred);
  Result:= Temp <> nil;
  if Result then
    Move(Temp^.Data^, Item, fDataSize);
end;

end.
```

إذا نظرنا للبرنامج السابق أو الوحدة السابقة المحتوية على الكائن TLinkedList من وجهة نظر البرمجة الكائنية. نجد أننا إستفدنا من ميزة الـ encapsulation في البرمجة الكائنية حيث قمنا بإخفاء البيانات الحساسة التالية: Information hiding

```
private
    fHead: TNodePointer;

    fSize: Integer;
    fDataSize: Word;
    function GetItem(Index: Integer; var Pred: TNodePointer): TNodePointer;
    function Compare(p1, p2: Pointer): Byte;
```

والتي لانرغب أن يصل إليها المستخدم (المبرمج المستخدم لهذه الوحدة)، لأنه لو وصل إليها يمكن أن يتسبب في ضياع البيانات، فمثلاً إذا استطاع أن يصل إلى المتغير fHead وقام بوضع nil فيه فإنه سوف يحّول كل القائمة المتصلة إلى عناقيد ضائعة سوف تتسبب في حجز مكان في الذاكرة بدون فائدة، ولايمكن التخلص منها إلا بإغلاق البرنامج.
كذلك الحال لباقي المتغيرات المهمة مثل fSize, fDataSize.
وعموماً فإن المتغيرات المستخدمة في أي كائن نخفيها عادة من المبرمج الذي سوف يستخدم هذا الكائن لاحقاً، وعوضاً عنها سوف نسمح له بقراءتها بدون تعديلها بإستخدام دوال فمثلاً الدالة Count تقوم بإرجاع قيمة fSize بدون أن يستطيع المبرمج أن يغيرها.

الدوال والأجراءات المسموح بإستخدامها والتي تعتبر المدخل الطبيعى للتعامل مع الكائن (وآتو البيوت من أبوابها) هي ماتقع تحت القسم public:

```
public
    constructor Create(ADataSize: Word);
    destructor Destroy;
    function Add(var Item): Integer;
    function Get(Index: Integer; var Item): Boolean;
    function Find(var Item): Integer;
    function Delete(Index: Integer): Boolean;
    function Count: Integer;

    function Insert(Index: Integer; var Item): Boolean;
    function Save(AFileName: string): Boolean;
    function Load(AFilename: string): Boolean;
end;
```

القائمة المتصلة ذات المؤشرين Doubly linked list

وهو نوع من أنواع القوائم المتصلة تتميز بوجود مؤشر آخر Prior يؤشر للعنصر السابق، وكذلك يوجد بها Head يؤشر لبداية القائمة و Tail يؤشر لمؤخرة القائمة، فعندما نريد إضافة عنصر في المؤخرة فما علينا إلا استخدام المؤشر Tail مباشرة.

وشكلها في الذاكرة يكون كالآتي:

	Node 1			Node 2			Node 3			
Head →	Nil ← Prior	Data	Next →	← Prior	Data	Next →	← Prior	Data	Next → Nil	← Tail

وهذه هي الوحدة المحتوية على الكائن TDLinkedList:

```
unit DList;

interface

type
   // TNodePointer : linked list node, contains one item (Data)
   // and pointer to the next node

   TNodePointer = ^TNode;
   TNode = record
          Data: Pointer;        // Any type of data
          Prior: TNodePointer; // Pointer to previous node
          Next: TNodePointer;  // Pointer to the next node
   end;

   { TDLinkedList }

   TDLinkedList = class
   private
       fHead: TNodePointer;
       fTail: TNodePointer;
       fLastPointer: TNodePointer;
       fLastPos: Integer;

       fSize: Integer;
       fDataSize: Word;
       function GetItem(Index: Integer): TNodePointer;
       function Compare(p1, p2: Pointer): Byte;
   public
       constructor Create(ADataSize: Word);
       destructor Destroy;
       function Add(var Item): Integer;
       function Get(Index: Integer; var Item): Boolean;
```

```
      function Find(var Item): Integer;
      function Delete(Index: Integer): Boolean;
      function Insert(Index: Integer; var Item): Boolean;
      function Replace(Index: Integer; var Item): Boolean;
      function Count: Integer;
      function Save(AFileName: string): Boolean;
      function Load(AFilename: string): Boolean;
   end;

implementation

constructor TDLinkedList.Create(ADataSize: Word);
begin
  fHead:= nil;
  fTail:= nil;
  fLastPointer:= nil;
  fLastPos:= -1;

  fDataSize:= ADataSize;
  fSize:= 0;
end;

destructor TDLinkedList.Destroy;
begin
  while fHead <> nil do
    Delete(0);
end;


(******************  Compare: Compare data  *******************)

function TDLinkedList.Compare(p1, p2: Pointer): Byte;
type
  X = array [0 .. 10000] of Byte;
var
  i: Integer;
  First, Second: ^ X;
begin
  Result:= 0;
  First:= P1;
  Second:= P2;
  for i:= 0 to fDataSize - 1 do
  if First^[i] <> Second^[i] then
  begin
    if First^[i] > Second^[i] then
      Result:= 1
    else
      Result:= 2;
    Break;
  end;
end;

(***************  Count: Get linked list items total number  ******************)

function TDLinkedList.Count: Integer;
begin
  Result:= fSize;
```

```
end;

(******** Save list into a file  **********)

function TDLinkedList.Save(AFileName: string): Boolean;
var
  F: file;
  Temp: TNodePointer;
begin
  try
    AssignFile(F, AFilename);
    Rewrite(F, 1);

    Temp:= fHead;

    while Temp <> nil do
    begin
      BlockWrite(F, Temp^.Data^, fDataSize);
      Temp:= Temp^.Next; // Goto next item
    end;
    CloseFile(F);
    Result:= True; // Sucessfull operation

  except
    Result:= False;
  end;
end;

(******** Load list from file  **********)

function TDLinkedList.Load(AFilename: string): Boolean;
var
  F: file;
  Temp: TNodePointer;
  Data: Pointer;
begin
  try
    AssignFile(F, AFilename);
    Reset(F, 1);

    Data:= GetMem(fDataSize); // Temporary buffer

    while Delete(0) do; // delete all current data in memory

    while not eof(F) do
    begin
      BlockRead(F, Data^, fDataSize);
      Add(Data^);
    end;
    FreeMem(Data);
    CloseFile(F);
    Result:= True; // Sucessfull operation

  except
    Result:= False;
  end;
```

```
end;

(****************  GetItem: get item pointer (Internal use) **************)

function TDLinkedList.GetItem(Index: Integer): TNodePointer;
var
  i: Integer;
  Temp: TNodePointer;

  DeltaHead, DeltaLast, DeltaTail: Integer;
  Delta: Integer;
  TempPos: Integer;
begin
  Temp:= fHead;
  i:= 0;
  Result:= nil; // Default: Not found
  if Index in [0 .. fSize -1] then
  begin
    if fLastPos = -1 then
    begin
      fLastPos:= 0;
      fLastPointer:= fHead;
    end;
    // determine which is the closest pointer (fHead, fLastPointer, or fTail)
    DeltaHead:= Index;
    DeltaLast:= Abs(Index - fLastPos);
    DeltaTail:= Abs(Index - fSize - 1);

    // Closer to head
    if (DeltaHead <= DeltaTail) and (DeltaHead <= DeltaLast)  then
    begin
      Delta:= DeltaHead;
      Temp:= fHead;
      TempPos:= 0;
    end
    else
    //  Closer to last
    if (DeltaLast <= DeltaHead) and (DeltaLast <= DeltaTail)  then
    begin
      Delta:= DeltaLast;
      Temp:= fLastPointer;
      TempPos:= fLastPos;
    end
    else
    // Closer to tail
    begin
      Delta:= DeltaTail;
      Temp:= fTail;
      TempPos:= fSize - 1;
    end;

    // Traverse
    while Index <> TempPos do
    begin
      if TempPos < Index then
      begin
        Inc(TempPos);
```

```
        Temp:= Temp^.Next;
      end
      else
      begin
        Dec(TempPos);
        Temp:= Temp^.Prior;
      end;

    end; // while
    fLastPos:= TempPos;
    fLastPointer:= Temp;
    Result:= Temp;
  end; // if Index in ..
end;

 (****************************** Add item ***********************************)

function TDLinkedList.Add(var Item): Integer;
var
  NewItem: TNodePointer;
begin

  // allocate data in memory (Heap) for the new item
  New(NewItem);
  GetMem(NewItem^.Data, fDataSize);

  // Put item in it's pointer
  Move(Item, NewItem^.Data^, fDataSize);
  NewItem^.Next:= nil; // Last item

  if fHead = nil then // Empty list
  begin
    fHead:= NewItem;
    fTail:= NewItem;
    fHead^.Prior:= nil;
    Result:= 0; // Addition Position
  end
  else  // Add in list tail
  begin
    Result:= fSize;

    NewItem^.Prior:= fTail;
    fTail^.Next:= NewItem;
    fTail:= NewItem;
  end;
  fLastPointer:= NewItem;
  fLastPos:= Result;
  Inc(fSize);
end;

(******************* Find: Find first occurrence of data ****************)

function TDLinkedList.Find(var Item): Integer;
var
  Temp: TNodePointer;
  i: Integer;
begin
  Result:= -1; // default: not found
```

```
  Temp:= fHead;
  i:= 0;
  while Temp <> nil do
  begin
    if Compare(Temp^.Data, @Item) = 0 then
    begin
      Result:= i;
      Break;
    end;
    Inc(i);
    Temp:= Temp^.Next;
  end;
end;

  (*************************   Delete node ***************************)

function TDLinkedList.Delete(Index: Integer): Boolean;
var
  Temp: TNodePointer;
begin
  Temp:= GetItem(Index);
  Result:= Temp <> nil;
  if Result then
  if Temp = fHead then  (***  Found in head  ***)
  begin
    Temp:= fHead;
    fHead^.Prior:= nil;
    fHead:= fHead^.Next; // New head: next item
    FreeMem(Temp^.Data, fDataSize);
    if Temp = fTail then
     fTail:= fHead; // New head
    fLastPointer:= fHead;
    if fLastPointer = nil then
      fLastPos:= -1
    else
      fLastPos:= 0;
    Dispose(Temp);
    Dec(fSize);
  end
  else
  begin // Normal item in the list
    Temp^.Prior^.Next:= Temp^.Next;
    if Temp^.Next <> nil then
      Temp^.Next^.Prior:= Temp^.Prior;

    if Temp = fTail then // last item
    begin
      fTail:= Temp^.Prior;
      fTail^.Next:= nil;
      fLastPointer:= fTail;
      fLastPos:= Index - 1;
    end;
    fLastPointer:= Temp^.Next;

    FreeMem(Temp^.Data, fDataSize);
    Dispose(Temp);
    Dec(fSize);
```

```
  end;
end;

(******* Insert before item # Index  ************)

function TDLinkedList.Insert(Index: Integer; var Item): Boolean;
var
  NewItem: TNodePointer;
  CurrentItem: TNodePointer;
begin
  if (Index = 0) and (fSize = 0) then
  begin
    Add(Item);
    Result:= True;
  end
  else
  if (Index < 0) or (Index >= fSize) then
    Result:= False
  else
  if Index = 0 then // Add befor head
  begin
    New(NewItem);
    NewItem^.Data:= GetMem(fDataSize);
    Move(Item, NewItem^.Data^, fDataSize);

    NewItem^.Prior:= nil;
    NewItem^.Next:= fHead;
    fHead:= NewItem;
    if fTail = NewItem then
      fTail:= fHead;
    Result:= True;
    fLastPos:= Index;
    fLastPointer:= NewItem;
    Inc(fSize);
  end
  else // Insert in the middle of list
  begin
    New(NewItem);
    NewItem^.Data:= GetMem(fDataSize);
    Move(Item, NewItem^.Data^, fDataSize);

    CurrentItem:= GetItem(Index);
    NewItem^.Next:= CurrentItem;
    NewItem^.Prior:= CurrentItem^.Prior;
    CurrentItem^.Prior^.Next:= NewItem;

    fLastPos:= Index;
    fLastPointer:= NewItem;

    Inc(fSize);
    Result:= True;
  end;
end;

(*********  Replace node data  *************)

function TDLinkedList.Replace(Index: Integer; var Item): Boolean;
```

```
var
  Temp: TNodePointer;
begin
  Temp:= GetItem(Index);
  Result:= Temp <> nil;
  if Result then
    Move(Item, Temp^.Data^, fDataSize);
end;

(******  Get data from node referenced by position Index  *******************)

function TDLinkedList.Get(Index: Integer; var Item): Boolean;
var
  Temp: TNodePointer;
begin
  Temp:= GetItem(Index);
  Result:= Temp <> nil;
  if Result then
    Move(Temp^.Data^, Item, fDataSize);
end;

end.
```

نلاحظ أننا قمنا بإضافة آلية لتسريع إسترجاع البيانات، وهذه الألية تكمن في إضافة المؤشر fLastPointer و المتغير fLastPos والذان يؤشران إلى آخر عنصر قام بإسترجاعه المستخدم، وذلك بإفتراض أن المستخدم إذا قام بإسترجاع العنصر رقم 25 مثلاً فربما يقوم بإسترجاع عنصر آخر قريب منه، مثلاً 26 أو 24، وعندها نتحرك خطوة واحدة فقط للأمام أو للخلف لجلب العنصر المحدد. كذلك فإن آلية الإسترجاع تقوم بقياس أي مؤشر أقرب للعنصر المراد إسترجاعه، هل هو مؤشر الـ Head أم الـ Tail أم الـ Last. فسوف يبدأ البحث إبتداءً من أقرب نقطة للعنصر المراد، وذلك لتحقيق سرعة الوصول للبيانات:

```
    // determine which is the closest pointer (fHead, fLastPointer, or fTail)
    DeltaHead:= Index;
    DeltaLast:= Abs(Index - fLastPos);
    DeltaTail:= Abs(Index - fSize - 1);

    // Closer to head
    if (DeltaHead <= DeltaTail) and (DeltaHead <= DeltaLast)  then
    begin
      Delta:= DeltaHead;
      Temp:= fHead;
      TempPos:= 0;
    end
    else
    //  Closer to last
    if (DeltaLast <= DeltaHead) and (DeltaLast <= DeltaTail)  then
    begin
      Delta:= DeltaLast;
      Temp:= fLastPointer;
      TempPos:= fLastPos;
    end
    else
    // Closer to tail
```

```
begin
  Delta:= DeltaTail;
  Temp:= fTail;
  TempPos:= fSize - 1;
end;
```

المقاطع والذاكرة

تمتاز لغة أوبجكت باسكال بدعمها للمقاطع بشكل جيد، وهي لغة ذات إمكانات عالية بالنسبة لمايعرف بمعالجة المقاطع string processing. وتوجد عدة طرق للتعامل مع المقاطع، وكل طريقة تعتمد على طريقة مختلفة للتخزين في الذاكرة.

Short String

وهي المقاطع الموروثة من أيام توربو باسكال، وهي عبارة عن مصفوفة أو سلسلة من الرموز طولها لايتجاوز 255 عنصراً. والعنصر الأول (رقم صفر) يحتوي على الطول المستخدم من هذه المصفوفة.

يمكن تعريف المقاطع من هذا النوع بتحديد طول المقطع كما في المتغير MyName أو بإستخدام النوع ShortString بدون تحديد طول، وفي هذه الحالة سوف يكون طوله 255 عنصر بالإضافة للعنصر رقم صفر الذي يحتوي على الطول المستخدم:

```
var
  MyName: string[30];
  MyCountry: ShortString;
```

في المثال التالي سوف نقوم بتشريح هذا النوع من المقاطع لروئة شكله في الذاكرة:

```
var
  MyName: string[30];
  i: Integer;
begin
  Write('Please input your name: ');
  Readln(MyName);
  Writeln('Variable Size: ', SizeOf(MyName));
  Writeln('Used length (MyName[0]): ', Byte(MyName[0]));
  Writeln('Used length (Length(MyName): ', Length(MyName));
  for i:= 0 to Length(MyName) do
    Writeln('Character ', i, ' = ', MyName[i], ', as a byte: ', Byte(MyName[i]));
  Readln;
end.
```

فإذا قمنا بإدخال الإسم Mohammed سوف تكون المخرجات كالآتي:

```
Variable Size: 31
Used length (MyName[0]): 8
Used length (Length(MyName): 8
Character 0 = #, as a byte: 8
Character 1 = M, as a byte: 77
Character 2 = o, as a byte: 111
Character 3 = h, as a byte: 104
Character 4 = a, as a byte: 97
Character 5 = m, as a byte: 109
Character 6 = m, as a byte: 109
Character 7 = e, as a byte: 101
Character 8 = d, as a byte: 100
```

في المثال السابق يتم حجز 31 بايت في الذاكرة، وفي هذه الحالة قطاع البيانات، وإذا كانت داخل دالة أو إجراء يتم حجزها في المكدسة.
نستخدم هذه الطريقة عندما نريد تسجيل بيانات في ملف، مثلاً:

```
type
  TName = string[100];

var
  AName: TName;
  F: file of TName;
```

كذلك يمكن استخدامها كحقل في سجل يراد حفظه في ملف.
نجد أن أكبر عيب في هذا النوع هو عدم قدرته لتسجيل بيانات أكبر من 255 رمز، وذلك لأنه يستخدم البايت الأول فقط لتسجيل طول المقطع، وأقصى قيمة عددية للبايت هي 255.

Null terminated string

وهي النوع الذي يُستخدم في لغة C. وهي عبارة عن تعريف لمصفوفة من الرموز، وهي في هذه الحالة غير محدودة الطول بعكس سابقتها Short String. ويبدأ تخزين المقطع من أول عنصر (رقم صفر) وعند نهايتها يقوم البرنامج بوضع القيمة صفر بعد آخر عنصر مستخدم لدلالة نهاية المقطع المستخدم.

مثال لنفس البرنامج السابق:

```
var
  MyName: array [0 .. 100] of Char;
  i: Integer;
begin
  Write('Input your name: ');
  Readln(MyName);
  Writeln('Variable Size: ', SizeOf(MyName));
  Writeln('Used length StrLen(MyName): ', strlen(MyName));
  for i:= 0 to StrLen(MyName) do
    Writeln('Character ',  i, ' = ', MyName[i] , ', as a byte: ',
     Byte(MyName[i]));
  Readln;
end.
```

فإذا قمنا بإدخال الإسم Mohammed مثلاً نحصل على المخرجات التالية:

```
Variable Size: 101
Used length StrLen(MyName): 8
Character 0 = M, as a byte: 77
Character 1 = o, as a byte: 111
Character 2 = h, as a byte: 104
Character 3 = a, as a byte: 97
Character 4 = m, as a byte: 109
Character 5 = m, as a byte: 109
Character 6 = e, as a byte: 101
```

```
Character 7 = d, as a byte: 100
Character 8 = , as byte: 0
```

نلاحظ أننا استخدمنا دالة مختلفة هذه المرة لمعرفة الطول الفعلي، وهو StrLen
تُستخدم هذه الطريقة في الإتصالات بكثرة (Socket programming) عند إرادة إرسال مقاطع بين البرامج. كذلك يمكن استخدامها في قراءة وتسجيل المقاطع في الملفات.

مقارنة بين طريقة التسجيل في الذاكرة للنوعين السابقين من المقاطع:

رقم العنصر	Short String	Null terminated string
0	#8	M
1	M	o
2	o	h
3	h	a
4	a	m
5	m	m
6	m	e
7	e	d
8	d	#0
9		
10		

Ansi String

وهو نوع جديد تم استحداثه مع بداية دلفي. وهو غير محدود الطول، حيث يمكنه تخزين مقطع طوله يزيد عن جيجا بايت. وهو أقرب للمصفوفة المرنة، حيث يتم حجز مساحة للمقطع وتمديدها في الذاكرة عند الحاجة تلقائياً، كذلك يتم تحريرها من الذاكرة تلقائياً. والذاكرة المستخدمة مع هذا النوع هو ذاكرة الكومة heap.

لتعريف متغير من هذا النوع يمكن تعريفه بهذه الطريقة:

```
  MyName: string;
```

حيث لايتم تحديد الحجم المستخدم من الذاكرة أو الجم الكلي.
وعندما نقوم بوضع قيمة فيها يتم حجز موقع من الذاكرة يتناسب مع حجم المقطع المستخدم.
مثال:

```
var
  MyName: string;
  i: Integer;
begin
  Write('Write your name: ');
  Readln(MyName);
  Writeln('Total Variable Size: ', SizeOf(MyName) + Length(MyName));
  Writeln('Used length Length(MyName): ', Length(MyName));
  for i:= 1 to Length(MyName) do
    Writeln('Character ',  i, ' = ', MyName[i] , ', as a byte: ',
    Byte(MyName[i]));
  MyName:= '';
  Readln;
end.
```

عندما نقوم بوضع مقطع فارغ كما في السطر قبل الأخير، يتم تحرير الذاكرة المستخدمة من الكومة:

```
  MyName:= '';
```

وفي حالة كان هذا المتغير متغيراً محلياً داخل دالة أو إجراء، فإنه يتم تحريره تلقائياً عند الفراغ من نداء الدالة أو الإجراء حتى لو نساه المبرمج.

يتم تخزين معلومات إضافية مع هذا النوع من المتغيرات بالإضافة إلى العنوان وهي:

1. طول المقطع المستخدم: يتم تخزين طول المقطع الحالي للرجوع له بإستخدام الدالة Length
2. عداد المؤشرات: في المثال السابق فإن MyName فقط هو الذي يؤشر على المقطع، فيكون عداد المؤشرات قيمته واحد، أما إذا قمنا بكتابة العبارة التالية:

```
  YourName:= MyName;
```

وكانت YourName هي أيضاً Ansi String فلايتم حجز موقع جديد، إنما يتم فقط وضع مؤشر YourName لنفس الموقع الذي يؤشر له MyName ، ثم تتم زيادة عداد المؤشرات للمتغير MyName إلى 2

ويبقى المتغيرين السابقين يؤشران إلى نفس الموقع مالم يتم يتغيير أحدهما. فإذا تغير أحدهما مثلاً:

```
  YourName:= YourName + ' ';
```

فإن المؤشران سوف ينفصلان ويتم نسخ المقطع إلى مقطع جديد معدل يؤشر له المتغير YourName. ويصبح عداد المؤشرات يحمل القيمة واحد للمتغير YourName وكذلك القيمة واحد للمتغير MyName ، ويصبح هناك موقعان مختلفات لهذه المقاطع.

كلما يتخلى مؤشر عن مقطع ما، تقل قيمة عداد المؤشرات، إلى أن يصبح صفراً أي لايؤشر أي مؤشر إلى هذا المقطع، وفي هذه الحالة يتم تحرير هذا المقطع من الذاكرة تلقائياً، وهذه العملية تُعرف بمحرر النفايات garbage collector. وهذه الطريقة مشهورة بها لغة الجافا.
نجد أن كل هذا يحدث تلقائياً ولايحتاج المبرمج أن يتدخل فيه، لذلك أصبحت طريقة استخدام Ansi String من أفضل طرق التعامل مع المقاطع في الذاكرة بالإضافة إلى طولها الغير محدود.

لنسخ مقطع من نوع Ansi String إلى null terminated string نقوم بإستخدام الدالة StrCopy الموجودة في المكتبة SysUtils كالآتي:

```
var
  Line: string;
  Block: array [0 .. 1023] of Char;
  i: Integer;
begin
  Line:= 'My text';
  strcopy(Block, PChar(Line));
  Writeln(Block);
  Readln;
end.
```

ولنسخ مقطع من نوع null terminated string إلى Ansi String نقوم أولاً بحجز مكان في الذاكرة بحجم المقطع المطلوب بإستخدام الإجراء SetLengh والذي نستخدمه مع المصفوفة المرنة، ثم نقوم بإستخدام الدالة StrCopy كالآتي:

```
var
  Line: string;
  Block: array [0 .. 1023] of Char;
  i: Integer;
begin
  Block:= 'My text';
  SetLength(Line, StrLen(Block));
  strcopy(PChar(Line), Block);
  Writeln(Line);
  Readln;
end.
```

الكائنات والمكونات Objects and Components

بالنسبة لحجز وتحرير الكائنات أو المكونات فإن طريقتهما أشبه بطريقة المؤشر المرتبط بنوع مثل

```
  Pi: ^Integer;
```

فمثلاً في هذا البرنامج قمنا بحجز الكائن List ثم إستخدامه ثم قمنا بتحريره:

```
var
  List: TStringList;
  i: Integer;
begin
  List:= TStringList.Create; // Allocate space for List object
  List.Add('First');
  List.Add('Second');
  List.Add('Third');

  for i:= 0 to List.Count - 1 do
    Writeln(List[i]);

  List.Free; // Free memory allocated by List object
  Readln;
end.
```

نلاحظ أننا قمنا بحجز موقع في الذاكرة للكائن بواسطة الـ Constructor وهو Create. وفي النهاية قمنا بتحريره بواسطة Free . و الدوال Create, Free لاتقوم بالحجز فقط، إنما يمكنها تشغيل كود يُستخدم يقوم بكتابته المبرمج عندما يقوم بإنشاء مكون أو كائن جديد.

يُفضل إستخدام الإجراء FreeAndNil الموجودة في الوحدة SysUtils بدلاً من الدالة Free، وذلك لأن الأولى تقوم بنداء الدالة Free ثم بوضع القيمة nil في متغير الكائن List:

```
  FreeAndNil(List); // Free memory allocated by List object,
                    //and put nil in List pointer
```

المراجع (فصل الذاكرة)

Free Pascal Documentation: http://www.freepascal.org/docs.var

الفصل الثاني

إدارة الملفات

Files Management

مقدمة

في الكتاب السابق ذكرنا كيفية التعامل مع الملفات بمختلف أنواعها، وكنا نقصد بها التعامل مع محتوى الملف. أما هذه المرة فسوف نتطرق إن شاء الله لكيفية التعامل مع أسماء الملفات والمجلدات، فمثلاً كيفية إستعراض ملفات في مجلد ما، أو حذف ملف أو تغيير إسمه.

إستعراض الملفات

في المثال التالي سوف نقوم بإدخال إسم ملف ثم يقوم البرنامج بعرض معلومات الملف، مثلاً هل هو موجود أم لا، وتاريخ تعديله، وحجمه:

```
program DisplayFileInfo;

{$mode objfpc}{$H+}

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

var
  FRec: TSearchRec;
  FileName: string;
begin
  Write('Please input file name: ');
  Readln(FileName);
  if FindFirst(FileName, faAnyFile, FRec) = 0 then
  begin
    Writeln('File name: ', FRec.Name);
    Writeln('File size: ', FRec.Size, ' bytes');
    Writeln('File Time: ', DateTimeToStr(FileDateToDateTime(FRec.Time)));
  end
  else
    Writeln('File not found');
  Readln;
end.
```

يمكن كتابة إسم الملف فقط إذا كان في نفس الدليل أو المجلد الذي يعمل به البرنامج، أو كتابة الدليل مع إسم الملف مثلاً:

```
/home/motaz/test.txt
```

في البرنامج السابق استخدمنا سجل من النوع **TSearchRec** وهو سجل للتعامل مع الدالة **FindFirst** التي استخدمناها للحصول على معلومات الملف.
يمكن كذلك إستخدام علامة * عند إدخال إسم الملف، مثلاً لو أدخلنا

```
/home/motaz/*.txt
```

فسوف يقوم البرنامج بإستعراض أول ملف يتنهي إمتداده بـ .txt
ولو أدخلنا *.* سوف يقوم البرنامج بإستعراض أول ملف يتحصل عليه.

برنامج إستعراض دليل

هذا البرنامج يقوم بسؤال المستخدم بإدخال إسم دليل، ثم يقوم بإستعراض كافة الملفات والمجلدات الفرعية التي يحتويها:

```
var
  FRec: TSearchRec;
  DirName: string;
begin
  Write('Please directory name end with directory sperator (/ or \): ');
  Readln(DirName);
  if FindFirst(DirName + '*.*', faAnyFile, FRec) = 0 then
  repeat
    if faDirectory and FRec.Attr > 0 then
    begin
      Writeln('[', FRec.Name, ']    <DIR> ',
        DateTimeToStr(FileDateToDateTime(FRec.Time)));
    end
    else
    begin
      Writeln(FRec.Name, '    ', FRec.Size, ' bytes   ',
        DateTimeToStr(FileDateToDateTime(FRec.Time)));
    end;
  until FindNext(FRec) <> 0
  else
    Writeln('File not found');
  Readln;
end.
```

استخدمنا الدالة FindNext والتي تقوم بإرجاع الملف التالي في سلسلة الملفات الموجودة في دليل ما.
كذلك استخدمنا الحقل Attr لمعرفة نوع الملف هل هو دليل فرعي أم ملف:

```
    if faDirectory and FRec.Attr > 0 then
```

برنامج عدد الأسطر في برنامج باسكال

في هذا المثال سوف يقوم البرنامج بفتح كافة ملفات باسكال وحساب عدد الأسطر فيها:

```
program codelines;

{$mode objfpc}{$H+}

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

```pascal
{$IFDEF WINDOWS}{$R codelines.rc}{$ENDIF}

var
  Directory: string;
  F: TextFile;
  Line: string;
  Count: Integer;
  Rec: TSearchRec;
  FileTypes: array [0 .. 1] of string;
  i: Integer;
begin
  Write('Input directory name end with (/ or \): ');
  Readln(Directory);
  Count:= 0;
  FileTypes[0]:= '*.pas';
  FileTypes[1]:= '*.lpr';

  // search and open files
  for i:= 0 to 1 do
  if FindFirst(Directory + FileTypes[i], faAnyFile, Rec) = 0 then
  repeat
    AssignFile(F, Directory + Rec.Name);
    Writeln('Counting ', Rec.Name);
    Reset(F);
    while not Eof(F) do
    begin
      Readln(F, Line);
      Inc(Count);
    end;
    CloseFile(F);
  until FindNext(Rec) <> 0;
  FindClose(Rec);
  Writeln('Finding ', Format('%3.0n', [Count / 1]) + ' pascal code lines');
  Writeln('Press enter to close');
  Readln;
end.
```

يقوم البرنامج بفتح جميع الملفات المنتهية بالإمتداد .**pas** و .**lpr** ملف بعد ملف ويقوم بقراءة كل الأسطر لحساب عددها، ثم يقوم بحساب العدد الكلي وإظهاره للمستخدم.
وكذلك استخدمنا الإجراء **FindClose** لتحرير الذاكرة أو الموارد المستخدمة بواسطة السجل **Rec**.

برنامج حجم المجلد Directory size

البرنامج التالي يطلب من المستخدم بإختيار مجلد أو دليل، ثم يقوم بقراءة كل معلومات الملفات ثم يحسب الحجم الكلي للملفات الموجودة في هذا المجلد:

قمنا بإنشاء برنامج جديد من نوع **Application** ووضعنا فيه زر ومحرر **Memo**

ثم قمنا بكتابة الكود التالي في الوحدة الرئيسية:

```
unit main;

{$mode objfpc}{$H+}

interface

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

const

{$IFDEF UNIX}
  Slash = '/';
{$ENDIF}

{$IFDEF WINDOWS}
   Slash = '\';
{$ENDIF}

type
```

```
  { TfmMain }

  TfmMain = class(TForm)
    Button1: TButton;
    Memo1: TMemo;
    procedure Button1Click(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  fmMain: TfmMain;

implementation

{ TfmMain }

function GetDirectorySize(ADir: string): Int64;
var
  FRec: TSearchRec;
begin
  Result:= 0; // Initial size
  if FindFirst(ADir + '*', faAnyFile , FRec) = 0 then
  repeat
    if FRec.Attr and faDirectory = 0 then // Normal file
      Result:= Result + FRec.Size
    else // Directory
    begin
      if (FRec.Name <> '.') and (FRec.Name <> '..') then
        Result:= Result + GetDirectorySize(ADir + FRec.Name + Slash);

    end;
  until FindNext(FRec) <> 0;
  FindClose(FRec);
end;

procedure TfmMain.Button1Click(Sender: TObject);
var
  Dir: string;
  ASize: Int64;
begin
  if SelectDirectory('Select a directory', '', Dir) then
  begin
    if Dir[length(Dir)] <> Slash then
      Dir:= Dir + Slash;
    ASize:= GetDirectorySize(Dir);
    Memo1.Lines.Add('');
    Memo1.Lines.Add('Directory: ' + Dir);

    if ASize < 1024 then // less than kilo
      Memo1.Lines.Add('Size = ' + Format('%3.0n', [ASize / 1]) +
      ' Bytes')
    else
    if ASize < 1048576 then // less than mega
      Memo1.Lines.Add('Size = ' + Format('%3.0n', [ASize / 1024]) +
```

```
      ' Kilo Bytes')
    else
    if ASize < 1073741824 then // less than gega
      Memo1.Lines.Add('Size = ' + Format('%3.0n', [ASize / 1048576]) +
       ' Mega Bytes')
    else
      Memo1.Lines.Add('Size = ' + Format('%3.0n', [ASize / 1073741824]) +
       ' Gega Bytes');

  end;
end;

initialization
  {$I main.lrs}

end.
```

نجد من البرنامج السابق الفوائد التالية:

1. إستخدمنا الدالة **SelectDirectory** وهي مشابهة لـ **OpenDialog1.Execute** إلا أنها تتيح للمستخدم إختيار مجلد وليس ملف:

```
if SelectDirectory('Select a directory', '', Dir) then
```

2. استخدمنا مايعرف بموجهات المترجم compiler directive وهي طريقة لتوجيه المترجم بالطريقة التي يترجم بها الكود، فمثلاً استخدمناه في هذه الحالة لضمان أن هذا البرنامج يعمل في بيئة لينكس أو وندوز، فإذا كان البرنامج تتم ترجمته في بيئة لينكس فإن فاصل المجلدات يكون /، أما إذا كان في بيئة وندوز فإن الفاصل يكون \

```
{$IFDEF UNIX}
  Slash = '/';
{$ENDIF}

{$IFDEF WINDOWS}
   Slash = '\';
{$ENDIF}
```

ففي حالة لينكس أو ماكنتوش يقوم المترجم بترجمة وإعتماد الخيار الأول / ويقوم بتجاهل الخيار الثاني كأنه ليس مكتوب، والعكس بالعكس.

3. قمنا بنداء الدالة GetDirectorySize من داخلها بما يعرف بالـ recursion :

```
        Result:= Result + GetDirectorySize(ADir + FRec.Name + Slash);
```

وقد استفدنا منها بتكرار عملية البحث داخل مجلد فرعي. فمثلاً إذا قام المستخدم بإختيار مجلد ما، وكان بداخله عدة مجلدات فرعية، فإن هذه الدالة سوف تغوص في كل المجلدات الفرعية لمعرفة حجم الملفات بداخلها، وسوف يتم الدخول إلى المجلدات الفرعية ومايتفرع منها مهما كانت تحتويه من تفرعات.

برنامج النسخ الذكي

تعتمد فكرة البرنامج في أنه يُتيح للمستخدم نسخ ملفات من مجلد ما في القرص الصلب إلى فلاش أو أي مكان آخر. وبدلاً من نسخ جميع الملفات في كل مرة، يقوم برنامج النسخ الذكي بمقارنة الملفات أولاً فإذا وجد أن حجم الملف و تاريخه في المصدر والمكان المراد النسخ إليه لم يتغير قام بتخطي ذلك الملف. وتكون النتيجة بنسخ الملفات التي تم تغييرها فقط.
كذلك فإن البرنامج يقوم بحفظ جلسات النسخ، حتى لايحتاج المستخدم بإختيار دليل النسخ في كل مرة، فقط يمكنه إختيار جلسة تم حفظها من قبل ليعيد النسخ.

قمنا بإنشاء مشروع جديد وقمنا بإنزال المكونات التالية في الفورم الرئيسي:

وقد قمنا بإضافة الوحدة DList.pas ليتم إستخدامها لحفظ الجلسات. وقمنا بكتابة الكود التالي في وحدة الفورم الرئيسي:

```
unit main;

{$mode objfpc}{$H+}

interface

uses
```

```
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs,
  StdCtrls, Buttons, DList;

const

{$IFDEF UNIX}
  Slash = '/';
{$ENDIF}

{$IFDEF WINDOWS}
   Slash = '\';
{$ENDIF}

type

  TSessionRec = record
     Title: string[250];
     Source: string[255];
     Dest: string[255];
     SubDir: Boolean;
     BackupTime: TDateTime;
  end;

  { TfmMain }

  TfmMain = class(TForm)
    bbBrowseSource: TBitBtn;
    bbBrowseDest: TBitBtn;
    bbBackup: TBitBtn;
    bbNew: TBitBtn;
    cbSessions: TComboBox;
    cxRecurs: TCheckBox;
    edDest: TEdit;
    edSource: TEdit;
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    laLastBackup: TLabel;
    Memo1: TMemo;
    procedure bbBackupClick(Sender: TObject);
    procedure bbBrowseDestClick(Sender: TObject);
    procedure bbBrowseSourceClick(Sender: TObject);
    procedure bbNewClick(Sender: TObject);
    procedure cbSessionsChange(Sender: TObject);
    procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
    procedure FormCreate(Sender: TObject);
  private
    Sessinos: TDLinkedList;
    function CopyDir(ASourceDir, ADestDir: string; Recure: Boolean): Integer;
    { private declarations }
  public
    { public declarations }
  end;

var
  fmMain: TfmMain;
```

```
implementation

{ TfmMain }

function TfmMain.CopyDir(ASourceDir, ADestDir: string; Recure: Boolean): Integer;
var
  FRec, DRec: TSearchRec;
  DestDirName: string;
begin
  Result:= 0;
  // Check if ASourceDir is a directory not a file
  if DirectoryExists(ASourceDir) then
  begin
    if ASourceDir[length(ASourceDir)] <> Slash then
      ASourceDir:= ASourceDir + Slash;
    ASourceDir:= ASourceDir + '*';
  end;

  Memo1.Lines.Add('الدليل: ' + ExtractFilePath(ASourceDir));
  Application.ProcessMessages;

  if ADestDir[length(ADestDir)] <> Slash then
    ADestDir:= ADestDir + Slash;

  if FindFirst(ASourceDir, faAnyFile , FRec) = 0 then
  repeat
    if FRec.Attr and faDirectory = 0 then // Normal file
    begin
      // copy onle files that has been changed/added
      if (FindFirst(ExtractFilePath(ADestDir) + FRec.Name, faAnyFile, DRec) <> 0)
       or
       (DRec.Size <> FRec.Size) or (DRec.Time <> FRec.Time) then
      begin
        Memo1.Lines.Add('الملف: ' + FRec.Name + ', ' + IntToStr(FRec.Size));
        CopyFile(ExtractFilePath(ASourceDir) + Frec.Name,
            ExtractFilePath(ADestDir) + FRec.Name, True);
        Application.ProcessMessages;
        Result:= Result + 1;
      end;
      FindClose(DRec);
    end
    else // Directory
    if Recure then
    begin
      if (FRec.Name <> '.') and (FRec.Name <> '..') then
      begin
        DestDirName:= ExtractFilePath(ADestDir) + FRec.Name;
        if not DirectoryExists(DestDirName) then
          CreateDir(DestDirName);
        Result:= Result + CopyDir(ExtractFilePath(ASourceDir) + Frec.Name,
          DestDirName, True);
      end;
    end;
  until FindNext(FRec) <> 0;
  FindClose(FRec);
end;
```

```
procedure TfmMain.bbBackupClick(Sender: TObject);
var
  Count: Integer;
  Rec: TSessionRec;
  Index: Integer;
begin
  if (Trim(edSource.Text) = '') or (Trim(edDest.Text) = '') then
    MessageDlg('يجب إختيار دليل ملفات النسخ ودليل للنسخ إلى', mtError, [mbOk], 0)
  else
  begin
    Memo1.Lines.Add('');
    Memo1.Lines.Add('بداية النسخ : ' + DateTimeToStr(Now));
    Count:= CopyDir(edSource.Text, edDest.Text, cxRecurs.Checked);
    Memo1.Lines.Add(IntToStr(Count) + ' ملفات تم نسخها ');
    Memo1.Lines.Add('الإنتهاء عند : ' + DateTimeToStr(Now));

    // Save backup session
    if Trim(cbSessions.Text) <> '' then
    begin
      Index:= cbSessions.Items.IndexOf(cbSessions.Text);
      if Index <> -1 then
        Sessinos.Delete(Index);
      Rec.Title:= cbSessions.Text;
      Rec.Source:= edSource.Text;
      Rec.Dest:= edDest.Text;
      Rec.SubDir:= cxRecurs.Checked;
      Rec.BackupTime:= Now;
      Sessinos.Insert(0, Rec);
      if Index = -1 then
        cbSessions.Items.Insert(0, Rec.Title) // Append
      else
        cbSessions.Items.Exchange(Index, 0);
    end;
  end;
end;

procedure TfmMain.bbBrowseDestClick(Sender: TObject);
var
  Dir: string;
begin
  if SelectDirectory('الدليل الذي سوف يُنسخ له', edDest.Text, Dir) then
    edDest.Text:= Dir;
end;

procedure TfmMain.bbBrowseSourceClick(Sender: TObject);
var
  Dir: string;
begin
  if SelectDirectory('دليل الملفات المراد نسخها', edSource.Text, Dir) then
    edSource.Text:= Dir;
end;

procedure TfmMain.bbNewClick(Sender: TObject);
begin
  cbSessions.Text:= '';
  edDest.Clear;
```

```
  edSource.Clear;
  cxRecurs.Checked:= False;
  laLastBackup.Caption:= '';
end;

procedure TfmMain.cbSessionsChange(Sender: TObject);
var
  Rec: TSessionRec;
begin
  if cbSessions.ItemIndex <> -1 then
  begin
    if Sessinos.Get(cbSessions.ItemIndex, Rec) then
    begin
      edDest.Text:= Rec.Dest;
      edSource.Text:= Rec.Source;
      cxRecurs.Checked:= Rec.SubDir;
      laLastBackup.Caption:= ' آخر نسخ ' + DateTimeToStr(Rec.BackupTime);
    end;
  end;
end;

procedure TfmMain.FormClose(Sender: TObject; var CloseAction: TCloseAction);
begin
  Sessinos.Save(ChangeFileExt(ParamStr(0), '.rec'));
end;

procedure TfmMain.FormCreate(Sender: TObject);
var
  Rec: TSessionRec;
  i: Integer;
begin
  cbSessions.Clear;
  Sessinos:= TDLinkedList.Create(SizeOf(TSessionRec));
  if FileExists(ChangeFileExt(ParamStr(0), '.rec')) then
    Sessinos.Load(ChangeFileExt(ParamStr(0), '.rec'));

  for i:= 0 to Sessinos.Count - 1 do
  begin
    Sessinos.Get(i, Rec);
    cbSessions.Items.Add(Rec.Title);
  end;

end;

initialization
  {$I main.lrs}

end.
```

الفصل الثالث

قواعد البيانات العلائقية

Relational Databases

مقدمة

في الكتاب السابق استخدمنا كافة أنواع الملفات لتخزين البيانات. إلا أن هذه الطريقة كانت تصلح للبيانات القليلة (بالآلاف مثلاً) وفي حالة مستخدم واحد، أو مايعرف ببرامج سطح المكتب (Desktop Applications). أما في حالة البيانات الكثيرة (ملايين السجلات) وفي حالة الحاجة لإستخدام شبكي للنظام (عدد من المستخدمين يصلون إلى نفس البيانات في آن واحد) فإن أنسب طريقة هي استخدام قواعد بيانات علائقية. وهي عبارة عن أنظمة مخصصة لقواعد البيانات ذات تركيب معقد تسمح للمبرمج إنشاء جدوال وعلاقات بين الجداول وفهارس وغيرها من الإمكانات التي تفرضها بيئات معقدة من المؤسسات الكبيرة.
هذه المرة لايقوم المبرمج بالوصول مباشرة إلى ملفات البيانات كما كان في السابق، وبدلاً عن ذلك يقوم المبرمج بإستخدام مكتبة أو مكونات لمخاطبة محرك قاعدة البيانات (Database Engine)، والذي بدوره يقوم بالتعامل مع البيانات التي تكون في الذاكرة وفي القرص الصلب.
وكمثال لمحركات قواعد البيانات العلائقية (RDBMS) :

1. Oracle
2. MS-SQL Server
3. IBM DB-2
4. MySQL
5. FireBird
6. Interbase
7. PostgreSql
8. Sybase
9. Informix

وبعضها تجاري والبعض الآخر حر ومفتوح المصدر مثل FireBird, MySql, PostgreSql

في هذا الفصل سوف نقوم بإستخدام FireBird وذلك لأنه حر بالكامل وغير محدود، ويعمل على عدد من أنظمة التشغيل مثل لينكس، ووندوز.

قاعدة بيانات FireBird

في عام 2000 قامت شركة بورلاند بفتح المصدر لمحرك قاعدة بياناتها Interbase. فقام فريق FirdBird مباشرة بإصدار فرع من كود Inerbase 6 وقامو بتسميته FireBird. بعد ذلك قامو بإجراء تعديلات متواصلة. إلى الآن ووصل رقم إصدارة FireBird إلى 2.5 ويخططون الآن لإصدار النسخة رقم 3.
تمتاز الـ FireBird بتقنية جديدة في التحكم في الوصول المتعدد لذلك فهي تسمى Multi-Generational RDBMS. وتمتاز بأن لغة الـ SQL التي تستخدمها تعتبر قياسية أكثر من محركات قواعد البيانات نظيراتها.

إحتياجات برامج قاعدة البيانات

عندما نقوم بتصميم برنامج يستخدم قاعدة بيانات علائقية، فنحن نحتاج عند إنزال وتثبيت البرنامج إلى محرك قاعدة بيانات ومكتبات للوصول لقاعدة البيانات، أي ان إنزال البرامج أكثر تعقيداً من البرنامج

العادية التي تستخدم فقط الملفات مثلاً. لذلك يجب على المبرمج أن يقرر أنه سوف يستخدم هذا النوع من البرامج عندما يجد أن النظام كبير ويستخدم إمكانيات قواعد البيانات العلائقية.

برنامج إدارة قاعدة البيانات FireBird

نحتاج لبرنامج لإدارة قاعدة بيانات FireBird وهذا البرنامج يُمكننا من الآتي:

1. إنشاء قاعدة بيانات جديدة
2. إنشاء جداول تتكون من حقول
3. تعديل الجداول
4. إنشاء الإجراءات، والفهارس
5. عرض البيانات وتعديلها

وغيرها من إحتياجات التعامل مع قاعدة البيانات.

توجد برامج كثيرة لإدارة قواعد بيانات FireBird منها FlameRobin. و Turbo Bird وسوف نستخدم في الأمثلة القادمة برنامج Turbo Bird إن شاء الله. وهذا البرنامج تم تصميمه بواسطة لازاراس وهو برنامج حر مفتوح المصدر.

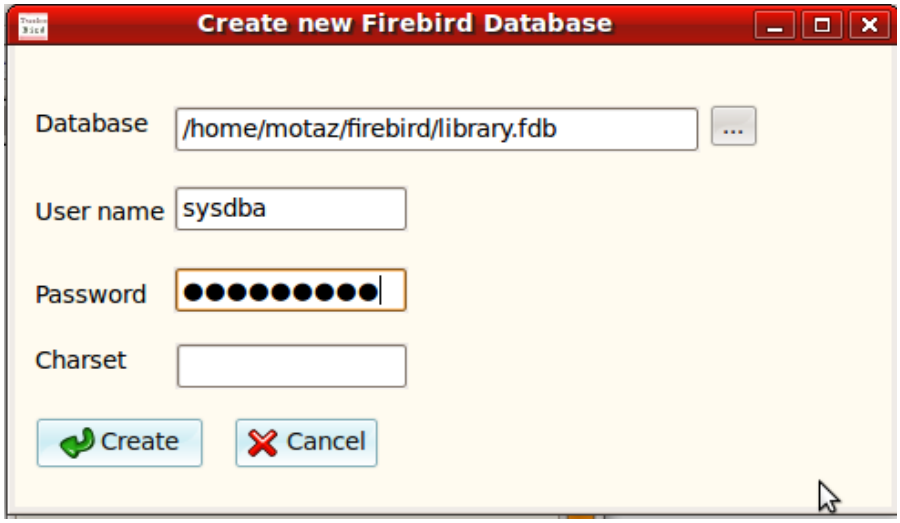
برنامج المكتبة المدرسية

سوف نقوم في هذا المثال بتصميم برنامج يصلح لمكتبة مدرسية. حيث يقوم مسؤول المكتبة بعمل الآتي:

1. إدراج الكتب الجديدة في البرنامج وتصنيفها
2. البحث عن كتاب معين
3. إعارة كتاب لطالب
4. معرفة الكتب المعارة
5. إستلام الكتب المعارة

نقوم أولاً بإنشاء قاعدة البيانات في مكان يستطيع محرك بيانات FireBird بالوصول له. فمثلاً في بيئة لينكس يستخدم هذا المحرك إسم دخول firebird ذو صلاحيات محدودة، لذلك يجب إنشاء دليل جديد وإعطاء هذا المستخدم صلاحية كاملة للوصول للملفات في هذا الدليل والتي سوف تكون قواعد البيانات، حيث تتمثل قاعدة البيانات الواحدة في ملف واحد ينتهي بالإمتداد fdb .

بعد تشغيل برنامج Turbo Bird نقوم بإختيار File/ Create New Database وندخل مدخلات كالآتي مثلاً:

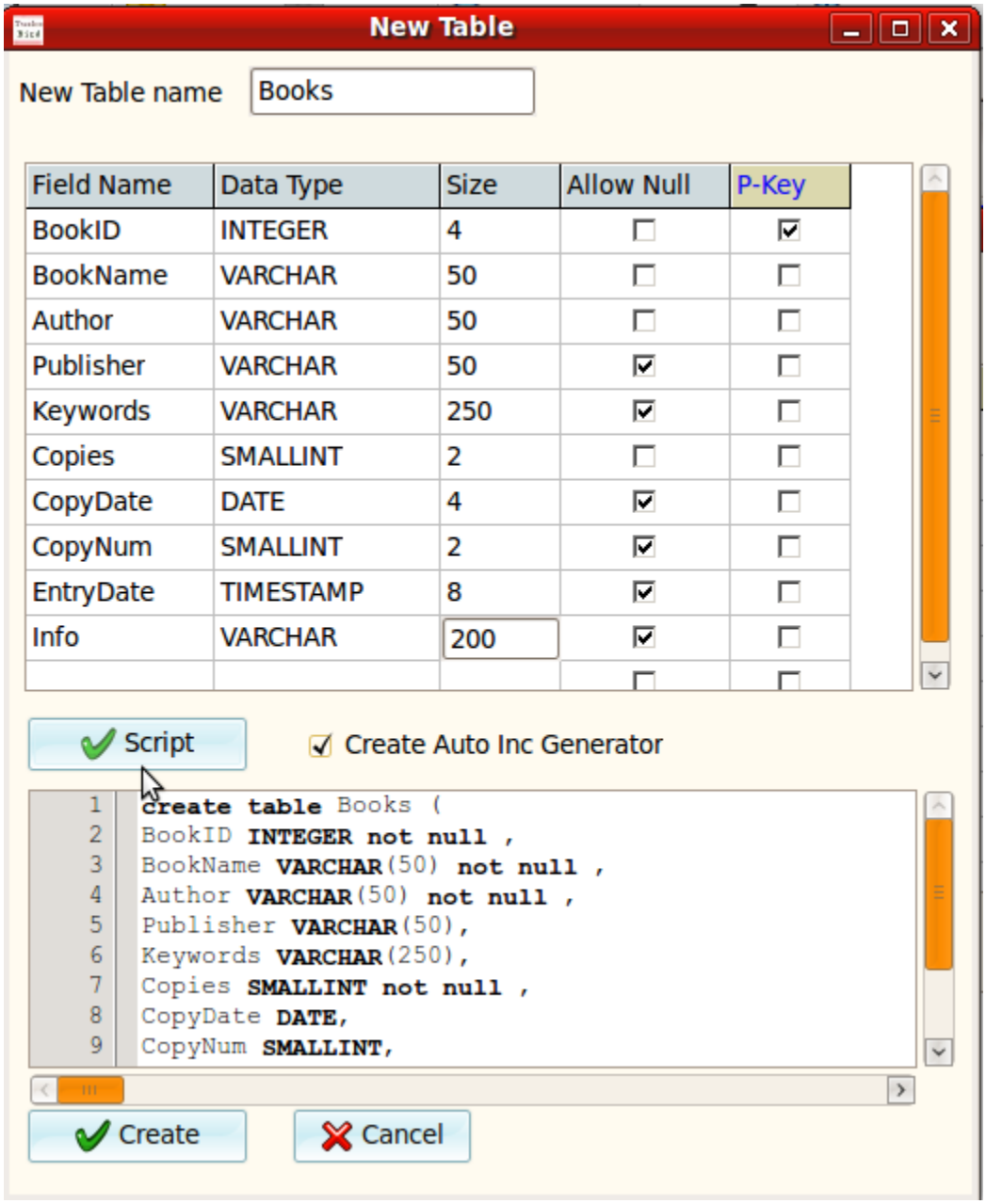


ثم نقوم بتسجيل قاعدة البيانات للوصول إليها دائما عن طريق برنامج Turbo Bird ونعطيها إسم مختصر مثل Library

الخطوة الثانية هي إنشاء الجدول أو الجداول التي سوف نحتاجها.
كبداية نريد إنشاء جدول الكتب الذي يحتوي على الحقول التالية:

الحقل	شرح
BookID	رقم تسلسلي يضاف تلقائياً للكتاب الجديد
BookName	إسم الكتاب
Author	إسم المؤلف
Publisher	إسم الناشر أو المطبعة
Keywords	كلمات مفتاحية لمحتويات الكتاب تساعد على البحث
Copies	عدد النسخ
CopyDate	تاريخ الطبعة
CopyNum	رقم الطبعة
EntryDate	تاريخ الإدخال ضمن المكتبة
Info	معلومات إضافية عن الكتاب، مثلاً موقع الكتاب

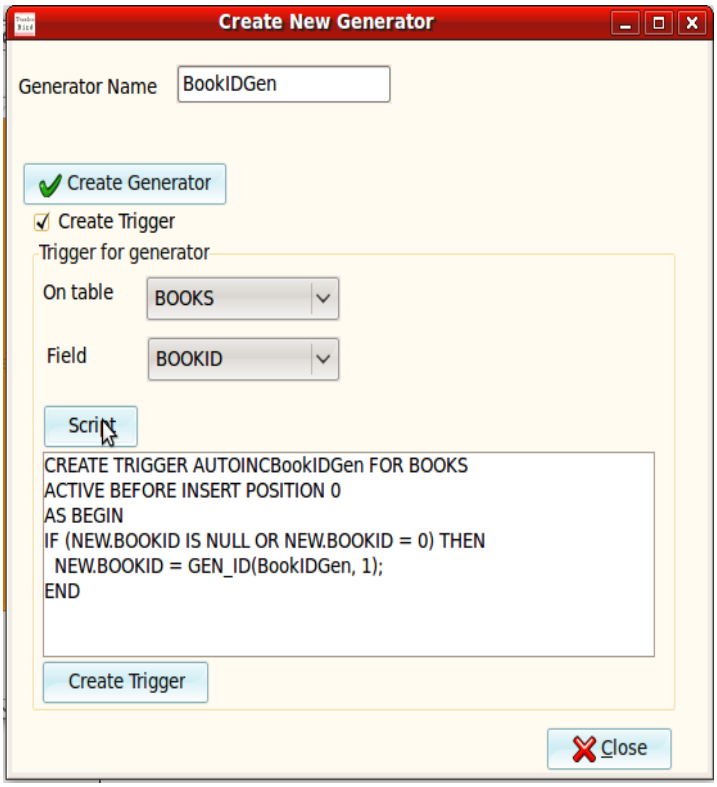
ثم نقوم إنشاء الجدول الجديد عن طريقة البرامج كالآتي:



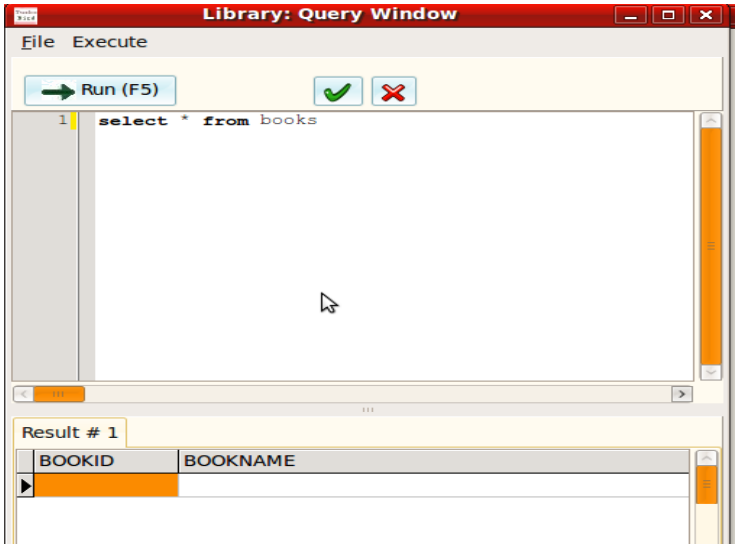
نلاحظ أننا قمنا بإختيار Create Auto Inc Generator وهو الرقم المتسلسل التلقائي الذي سوف نستخدمه مع الحقل BookID. كذلك قمنا بإختيار الحقل نفسه ليكون المفتاح الرئيسي للجدول Primary Key.
ونلاحظ كذلك أننا قمنا بتصحيح بعض الحقول بالنسبة للعمود Allow Null وهي حقول غير مهمة يمكن تجاهلها إثناء الإدخال.

بعد الضغط على الزر Script يظهر كود الـ SQL الذي عن طريقه ننشيء جدول في قاعدة بيانات FireBird. ثم نقوم بالضغط على الزر Create

بعد إنشاء الجدول بنجاح، نجد شاشة إضافة Generator :



فنقوم بإختيار إسم كالإسم السابق، ثم نقوم بالضغط على الزر Create Generator، بعد ذلك نقوم بإنشاء trigger وهو الإجراء الذي سوف يقوم بإضافة الرقم التسلسلي عند إضافة كتاب جديد تلقائياً.
نختار الحقل BookID ثم نضط الزر Script ثم Create Generator.



بعد ذلك نقوم بفتح نافذة Query Window ونكتب الكود التالي لنتأكد من أن الجدول قد تمت إضافته:

ثم نقوم بتجربة إضافة كتاب عن طريق نفس الشاشة. وذلك بكتابة كود الـ SQL التالي:

```
insert into Books (BookName, Author, Publisher, Keywords, Copies, CopyDate,
CopyNum, EntryDate)
values ('إدارة الموارد البشرية' ,'محمد الصيرفي' ,'دار الفكر الجامعي' ,'إدارة،
الموارد البشرية، الت
خطيط، القوى العاملة' ,1 ,'01-01-2007' ,1
```

```
CURRENT_TIMESTAMP);
```

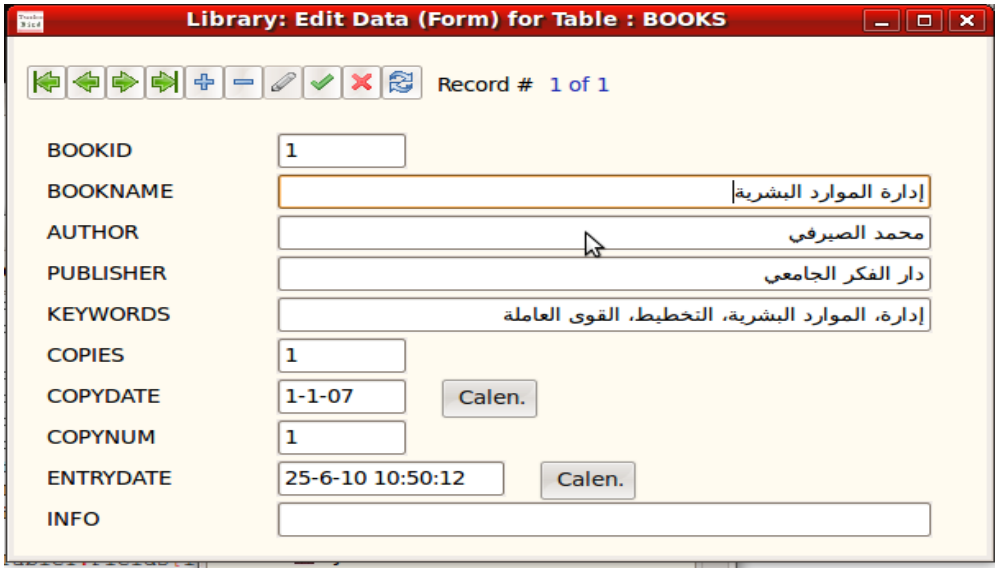
بعد تنفيذه، نقوم بالضغط على زر Commit لحفظه فعلياً في القرص الصلب بدلاً من الذاكرة. ثم نقوم مرة أخرى بعرض محتوى الجدول بواسطة:

```
select * from books
```

كذلك يمكن عرض المحتويات أو إضافتها بطريقة أسهل، وهل إختيار الجدول ثم إختيار:

Library/Tables/Books / Edit Data (Form)

وذلك بالضغط على الزر الأيمن للماوس في الجدول Books ثم Edit Form



بعد ذلك نقوم بإنشاء برنامج جديد وفي الفورم الرئيسي نضع Label نكتب فيه إضافة كتاب:

- ثم نقوم بإضافة TDataModule من File/New/Data Module وهو عبارة عن حاوية تُوضع فيها المكونات التي لاتظهر أثناء التشغيل. ثم نقوم بتسمية الوحدة Data وإسم الحاوية dmData

- ثم نضع المكونات التالية: IBConnection, SQLTransaction, SQLQuery من صفحة SQLdb

- ثم نختار قاعدة البيانات Library.fdb في خاصية DatabaseName في المكون ibconnection1. فإذا كان محرك قاعدة البيانات فير بيرد موجود محلياً (أي مع البرنامج في نفس الكمبيوتر) فيمكن أن يكون إسم قاعدة البيانات في الخاصية DataBaseName كالآتي:

```
/home/motaz/firebird/library.fdb
```

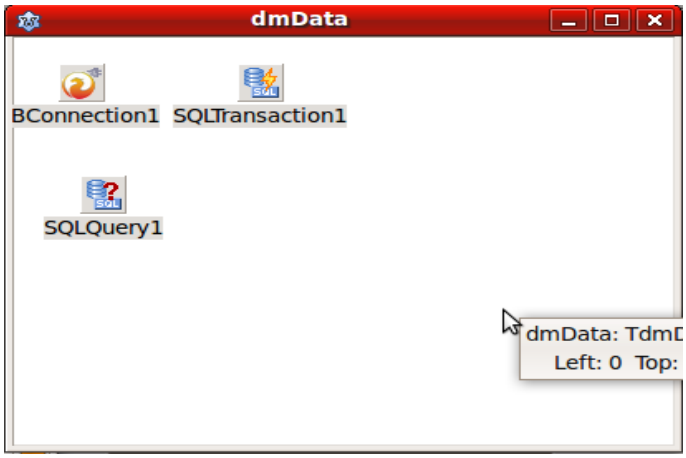
أما إذا كان محرك البيانات في جهاز آخر (مخدم) فيمكن الوصول إليه بإسم ذلك الجهاز أو عنوانه الخاص بالشبكة مثلاً:

```
192.168.1.2:/home/firebird/library.fdb
```

- ثم نُدخل إسم المستخدم لقاعدة البيانات وكلمة المرور في الخواص: UserName, Password

- بعد ذلك نقوم بإختيار SqlTransaction1 في الخاصية Transaction في IBConnection1

- في المكون SQLQuery1 نقوم بإختيار IBConnection1 و SQLTransaction1



- بعد ذلك نقوم بكتابة إجراء إضافة كتاب في وحدة حاوية البيانات dmData:

```
function TdmData.AddBook(BookName, Author, Publisher, Keywords, Info: string;
  Copies, CopyNum: Integer; CopyDate: TDate): Boolean;
begin
  try
    SQLQuery1.Close;
    SQLQuery1.SQL.Text:= 'insert into Books (BookName, Author, Publisher, ' +
      'Keywords, Copies, CopyDate,' +
      ' CopyNum, EntryDate, Info) ' +
      'values (:BookName, :Author, :Publisher, :Keywords,' +
      ' :Copies, :CopyDate, :CopyNum, :EntryDate, :Info) ';

    SQLQuery1.Params.ParamByName('BookName').Value:= BookName;
    SQLQuery1.Params.ParamByName('Author').Value:= Author;
    SQLQuery1.Params.ParamByName('Publisher').Value:= Publisher;
    SQLQuery1.Params.ParamByName('Keywords').Value:= Keywords;
```

```
    SQLQuery1.Params.ParamByName('Copies').Value:= Copies;
    SQLQuery1.Params.ParamByName('CopyDate').Value:= CopyDate;
    SQLQuery1.Params.ParamByName('CopyNum').Value:= CopyNum;
    SQLQuery1.Params.ParamByName('EntryDate').Value:= Now;
    SQLQuery1.Params.ParamByName('Info').Value:= Info;
    SQLQuery1.ExecSQL;
    SQLTransaction1.Commit;
    Result:= True;

  except
  on e: exception do
  begin
    Result:= False;
    ShowMessage(e.Message);
  end;
  end;

end;
```

نلاحظ في الإجراء السابق أننا قد قمنا بكتابة الـ SQL Code في الخاصية SQL في المكون SQLQuery1.
وقد قمنا بإضافة باراميترات أو مدخلات يتم الوصول إليها لاحقاً. مثلاً إسم الكتاب :

:BookName

نُدخله بهذه الطريقة:

```
    SQLQuery1.Params.ParamByName('BookName').Value:= BookName;
```

ويتم تنفيذ كود الـ SQL عن طريق الإجراء ExecSQL. أما الإجراء Commit فهو يقوم بحفظ التغييرات التي حدثت في قاعدة البيانات في القرص الصلب. لأن التعديلات يتم حفظها أولاً في ذاكرة مخصصة لهذا المستخدم فإما قام بحفظها عن طريق Commit أو بإلغائها عن طريق Rollback.

ملحوظة:

قمنا بتوسعة حقل BookName إلى 100 حرف بدلاً من 50، و Publisher إلى 100 والـ Author إلى 150 وذلك لإمكانية وجود أكثر من مؤلف في الكتاب الواحد.
وقمنا بذلك عن طريق برنامج Turbo Bird :

Tables-> Books-> Table Management → Fields -> Edit

- بعد ذلك نقوم بإضافة فورم جديد للإضافة به Edit Box, SpinEdit كالتالي:

- ثم نقوم بكتابة الكود التالي في زر الإضافة ، بعد إضافة الوحدة Data إلى الـ uses clause:

```
procedure TfmAddBook.BitBtn1Click(Sender: TObject);
begin
  if dmData.AddBook(edBookName.Text, edAuthor.Text, edPublisher.Text,
    edKeywords.Text, edInfo.Text,
    seCopies.Value, seCopyNum.Value, EncodeDate(seCopyDate.Value, 1, 1)) then
      ShowMessage('تمت إضافة الكتاب بنجاح');
end;
```

إجراء البحث

بالنسبة للبحث عن كتاب أو مجموعة كتب، فيمكننا البحث عن طريق اسم كتاب، اسم مؤلف، كلمة مفتاحية، اسم الناشر أو معلومات إضافية.

- نقوم بإضافة فورم جديد نحفظه بإسم Search.pas ونسمي الفورم كمكون fmSearch. ثم نقوم بإضافة الوحدة Data تحت Implementation كالآتي:

```
uses data;
```

- ثم نضع في الفورم GroupBox والتي نضع فيها حقول البحث. ثم نضع TdbGrid من صفحة Data Controls، و TDataSource من صفحة Data Access فيصبح الشكل كالآتي:

- في زر بحث، نكتب الكود التالي:

```
procedure TfmSearch.bbSearchClick(Sender: TObject);
begin
  dmData.SQLQuery1.Close;
  dmData.SQLQuery1.SQL.Text:= 'select * from Books where ';

  if Trim(edBookName.Text) <> '' then
    dmData.SQLQuery1.SQL.Add('BookName like ''%' + edBookName.Text + '%'' and ');

  if Trim(edAuthor.Text) <> '' then
    dmData.SQLQuery1.SQL.Add('Author like ''%' + edAuthor.Text + '%'' and ');

  if Trim(edPublisher.Text) <> '' then
    dmData.SQLQuery1.SQL.Add('Publisher like ''%' + edPublisher.Text +
       '%'' and ');

  if Trim(edKeyword.Text) <> '' then
    dmData.SQLQuery1.SQL.Add('Keywords like ''%' + edKeyword.Text + '%'' and ');

  if Trim(edInfo.Text) <> '' then
    dmData.SQLQuery1.SQL.Add('Info like ''%' + edInfo.Text + '%'' and ');
```

```
  dmData.SQLQuery1.SQL.Add('1 = 1');
  dmData.SQLQuery1.Open;
end;
```

نلاحظ أننا استخدمنا عبارة الشرط if وذلك حتى يتم تشكيل كود الـ SQL تلقائياً حسب ماقام بإختياره المستخدم، فمثلاً إذا قام المستخدم بكتابة كلمة إدارة في اسم الكتاب وترك باقي الحقول فإن كود الـ SQL سوف يصبح كالآتي:

```
select * from Books where
BookName like '%إدارة%' and
1 = 1
```

ونلاحظ أننا قمنا بإضافة العبارة :

```
1 = 1
```

وذلك حتى تكون تكملة لـ and حتى لاتكون العبارة ناقصة، وكذلك عندما لايقوم المستخدم بإدخال أي حقل فإنه يتحصل على كل الكُتب الموجودة في المكتبة، وفي هذه الحالة تكون العبارة كالآتي:

```
select * from Books where
1 = 1
```

- ثم نضيف وحدة Search إلى وحدة الفورم الرئيسي فتصبح عبارة uses في الوحدة الرئيسة كالآتي:

```
uses AddBook, search;
```

- بعد ذلك نقوم بإدراج Label نكتب فيه بحث عن كتب في الفورم الرئيسي ونقوم بإظهار فورم البحث كالآتي:

```
procedure TfmMain.Label2Click(Sender: TObject);
begin
  fmSearch.ShowModal;
end;
```

تعديل البيانات

بالنسبة لتعديل البيانات، قمنا بإضافتها في شاشة البحث، حيث يمكن للمستخدم أن يبحث عن كتاب ثم يقوم بتعديل معلوماته. ولإضافة هذا الإجراء قمنا بعمل بعض التعديلات في فورم ووحدة إضافة الكتاب AddBook بدلاً من عمل فورم مشابه وحتى لانقوم بتكرار، لأنه يوجد فارق بسيط بين فورم أو شاشة إضافة أو تعديل كتاب. والتعديلات هي:

- قمنا بتعديل إجراء التهيئة Init حيث قمنا بإضافة مُدخل للتفريق بين إضافة جديدة والتعديل.
- كذلك تمت إضافة مُدخل ثاني يحتوي على الرقم المفتاحي للسجل المُراد تعديله، فأصبح الإجراء كالآتي:

```
procedure TfmAddBook.Init(NewBook: Boolean; BookID: Integer = 0);
var
  Y, M, D: Word;
begin
  fNewBook:= NewBook;
  if fNewBook then
  begin
    edBookName.Clear;
    edAuthor.Clear;
    edPublisher.Clear;
    edKeywords.Clear;
    edInfo.Clear;
    DecodeDate(Date, Y, M, D);
    seCopyDate.Value:= Y;
    seCopies.Value:= 1;
    seCopyNum.Value:= 1;
    bbAdd.Caption:= 'إضافة';
  end
  else
  begin
    fBookID:= BookID;
    bbAdd.Caption:= 'تعديل';
  end;
end;
```

والمتغير fNewBook هو متغير تم تعريفه في فورم الإضافة تحت القسم private ليتم استخدامه لاحقاً مع إجراء التعديل.

- قمنا بتعديل حدث الضغط على الزر Add ليعمل حسب حالة الفورم، فإذا كان يُستخدم للإضافة تم استدعاء إجراء الإضافة، وإذا كان يُستخدم للتعديل تم نداء إجراء التعديل كالآتي:

```
procedure TfmAddBook.bbAddClick(Sender: TObject);
begin
  if fNewBook then // Add
  begin
    if dmData.AddBook(edBookName.Text, edAuthor.Text, edPublisher.Text,
         edKeywords.Text, edInfo.Text, seCopies.Value, seCopyNum.Value,
         EncodeDate(seCopyDate.Value, 1, 1)) then
```

```
          ShowMessage('تمت إضافة الكتاب بنجاح')
  end
  else   // Modify
  begin
    dmData.EditBook(fBookID, edBookName.Text, edAuthor.Text, edPublisher.Text,
      edKeywords.Text, edInfo.Text,seCopies.Value, seCopyNum.Value,
      EncodeDate(seCopyDate.Value, 1, 1));
  end;

end;
```

- قمنا بكتابة إجراء جديد في dmData وهو إجراء تعديل كتاب EditBook :

```
function TdmData.EditBook(BookID: Integer; BookName, Author, Publisher,
  Keywords, Info: string; Copies, CopyNum: Integer; CopyDate: TDate): Boolean;
begin
  try
    SQLQuery1.Close;
    SQLQuery1.SQL.Text:= 'update Books set BookName = :BookName, ' +
      'Author = :Author, Publisher = :Publisher' +
      ', Keywords = :Keywords, Copies = :Copies, CopyDate = :CopyDate,' +
      ' CopyNum = :CopyNum, EntryDate = :EntryDate, Info = :Info ' +
      'where BookID = :BookID';
    SQLQuery1.Params.ParamByName('BookID').Value:= BookID;
    SQLQuery1.Params.ParamByName('BookName').Value:= BookName;
    SQLQuery1.Params.ParamByName('Author').Value:= Author;
    SQLQuery1.Params.ParamByName('Publisher').Value:= Publisher;
    SQLQuery1.Params.ParamByName('Keywords').Value:= Keywords;
    SQLQuery1.Params.ParamByName('Copies').Value:= Copies;
    SQLQuery1.Params.ParamByName('CopyDate').Value:= CopyDate;
    SQLQuery1.Params.ParamByName('CopyNum').Value:= CopyNum;
    SQLQuery1.Params.ParamByName('EntryDate').Value:= Now;
    SQLQuery1.Params.ParamByName('Info').Value:= Info;
    SQLQuery1.ExecSQL;
    SQLTransaction1.CommitRetaining;
    Result:= True;

  except
  on e: exception do
  begin
    Result:= False;
    ShowMessage(e.Message);
  end;
  end;

end;
```

والفرق الرئيسي بينه وبين إجراء إضافة كتاب AddBook هو كود الـ SQL حيث أن الأول كان يستخدم عبارة

```
insert into TableName....
```

أما التعديل فهو يستخدم عبارة:

```
update TableName...
```

- نلاحظ كذلك أننا استخدمنا CommitRetaining والتي لاتقوم بإغلاق نتيجة البحث بخلاف Commit والتي تقوم بإغلاق كافة السجلات المفتوحة.

- كذلك قمنا بإضافة TSQLQuery جديد وسميناه sqSearch ليُستخدم في فورم البحث بدلاً من SqlQuery1 الذي يستخدم لغرض الإضافة والتعديل.

- قمنا كذلك بإضافة زر للتعديل يعمل في حالة وجود نتائج بحث. وكتبنا فيه الكود التالي :

```
procedure TfmSearch.bbEditClick(Sender: TObject);
var
  Y, M, D: Word;
begin
  with dmData.sqSearch do
  begin
    fmAddBook.Init(False, FieldByName('BookID').AsInteger);
    fmAddBook.edBookName.Text:= FieldByName('BookName').AsString;
    fmAddBook.edAuthor.Text:= FieldByName('Author').AsString;
    fmAddBook.edPublisher.Text:= FieldByName('Publisher').AsString;
    fmAddBook.edKeywords.Text:= FieldByName('Keywords').AsString;
    fmAddBook.edInfo.Text:= FieldByName('Info').AsString;
    fmAddBook.seCopies.Value:= FieldByName('Copies').AsInteger;
    fmAddBook.seCopyNum.Value:= FieldByName('CopyNum').AsInteger;
    DecodeDate(FieldByName('CopyDate').AsDateTime, Y, M, D);
    fmAddBook.seCopyDate.Value:= Y;
    fmAddBook.ShowModal;
  end;
end;
```

إستلاف وإرجاع الكتب

من ضمن خواص برامج المكتبة هي إمكانية إعارة كتب وإرجاعها. وحصر الكتب المعارة.
لإضافاة هذه الخاصية قمنا بإضافة حقل جديد إسمه IsBorrowed في الجدول Books وذلك عن طريق برنامج توربو بيرد Library/Tables/Books/Table Management/Fields /New/.
نوع المتغير الجديد هو SmallInt والقيمة الإفتراضية Default Value هي 0. وهي تعني أن الكتاب موجود في المكتبة، أما في حالة إعارة الكتاب فإننا نقوم بتحويل هذه القيمة إلى 1.

بعد ذلك قمنا بإضافة جدول جديد إسمه BookHistory به الحقول التالية:

P-Key	Field Name	Data Type	Size	Allow Null
☑	ID	INTEGER	4	☐
☐	BOOKID	INTEGER	4	☑
☐	OPERATIONTYPE	SMALLINT	2	☑
☐	OPERATIONTIME	TIMESTAMP	8	☑
☐	SUBSCRIBERNAME	VARCHAR	50	☑
☐	NOTES	VARCHAR	50	☑

كذلك قمنا بإضافة Generator و Trigger لإضافة رقم تلقائي في الحقل ID.
ثم قمنا بكتابة إجراء جديد في dmData أسميناه InsertBookOperation:

```
function TdmData.InsertBookOperation(BookID, OperationType: Integer;
  SubscriberName, Notes: string): Boolean;
begin
  try
    sqInsertBookOperation.Close;
    sqInsertBookOperation.SQL.Text:= 'insert into BOOKHISTORY ' +
      '(BOOKID, OPERATIONTYPE, OPERATIONTIME, SUBSCRIBERNAME, NOTES) ' +
      'values (:BOOKID, :OPERATIONTYPE, :OPERATIONTIME, :SUBSCRIBERNAME, :NOTES)';
    sqInsertBookOperation.Params.ParamByName('BookID').Value:= BookID;
    sqInsertBookOperation.Params.ParamByName('OperationType').Value:=
      OperationType;
    sqInsertBookOperation.Params.ParamByName('SubscriberName').Value:=
      SubscriberName;
    sqInsertBookOperation.Params.ParamByName('Notes').Value:= Notes;
    sqInsertBookOperation.Params.ParamByName('OperationTime').Value:= Now;
    sqInsertBookOperation.ExecSQL;

    // Update Borrow status
    case OperationType of
      1: SetIsBorrowed(BookID, 1); // Borrow book
      2: SetIsBorrowed(BookID, 0); // Return book
    end;

    SQLTransaction1.CommitRetaining;
    Result:= True;

  except
  on e: exception do
  begin
    Result:= False;
    ShowMessage(e.Message);
  end;
  end;

end;
```

وإجراء آخر لتعديل حالة الكتاب (مستلف، متوفر):

```
function TdmData.SetIsBorrowed(BookID, IsBorrowed: Integer): Boolean;
begin
  try
    sqSetIsBorrowed.Close;
    sqSetIsBorrowed.SQL.Text:= 'update Books set IsBorrowed = :IsBorrowed where '
      + 'BookID = :BookID';
    sqSetIsBorrowed.Params.ParamByName('BookID').Value:= BookID;
    sqSetIsBorrowed.Params.ParamByName('IsBorrowed').Value:= IsBorrowed;
    sqSetIsBorrowed.ExecSQL;
    Result:= True;

  except
  on e: exception do
```

```
  begin
    Result:= False;
    ShowMessage(e.Message);
  end;
  end;
end;
```

وقد قمنا بإضافة sqInsertBookOperation, sqSetIsBorrowed من النوع TSQLQuery

كذلك قمنا ببعض التغييرات في فورم البحث، حيث قمنا بإضافة حالة الكتاب للبحث. وقمنا بإضافة زرين للإعارة وإعادة الكتاب:

وأصبح أجراء البحث الجديد هو:

```
procedure TfmSearch.bbSearchClick(Sender: TObject);
begin
  dmData.sqSearch.Close;
  dmData.sqSearch.SQL.Text:= 'select * from Books where ';

  if Trim(edBookName.Text) <> '' then
    dmData.sqSearch.SQL.Add('BookName like ''%' + edBookName.Text + '%'' and ');

  if Trim(edAuthor.Text) <> '' then
    dmData.sqSearch.SQL.Add('Author like ''%' + edAuthor.Text + '%'' and ');
```

```
  if Trim(edPublisher.Text) <> '' then
    dmData.sqSearch.SQL.Add('Publisher like ''%' + edPublisher.Text + '%'' and ');

  if Trim(edKeyword.Text) <> '' then
    dmData.sqSearch.SQL.Add('Keywords like ''%' + edKeyword.Text + '%'' and ');

  if Trim(edInfo.Text) <> '' then
    dmData.sqSearch.SQL.Add('Info like ''%' + edInfo.Text + '%'' and ');

  // New
  bbBorrow.Enabled:= True;
  bbReturn.Enabled:= True;

  case cbBookState.ItemIndex of
    1: begin // View existed books
         dmData.sqSearch.SQL.Add('IsBorrowed = 0 and ');
         bbReturn.Enabled:= False;
       end;
    2: begin // View Borrowed books
         dmData.sqSearch.SQL.Add('IsBorrowed = 1 and ');
         bbBorrow.Enabled:= False;
       end;
  end;
  //
  dmData.sqSearch.SQL.Add('1 = 1');
  dmData.sqSearch.Open;

  bbEdit.Enabled:= dmData.sqSearch.RecordCount > 0;

  // New
  if not bbEdit.Enabled then
  begin
    bbBorrow.Enabled:= False;
    bbReturn.Enabled:= False;
  end;
  //
end;
```

بعد ذلك قمنا بإضافة فورم جديد لكتابة بيانات الإعارة أو الإعادة، وأسميناه fmBookOperation واسم الوحدة BookOperation. بالشكل التالي:

وقمنا بإضافة المتغيرات التالية في قسم Public في كود الفورم:

```
  public
    OperationType: Integer;
    BookID: Integer;
    { public declarations }
  end;
```

وكتبنا الكود التالي في زر (إجراء):

```
procedure TfmBookOperation.bbProcedureClick(Sender: TObject);
begin
  if dmData.InsertBookOperation(BookID, OperationType, edSubscribername.Text,
      edNotes.Text) then
    ModalResult:= mrOk;
end;
```

ثم كتبنا الإجرائين التاليين في فورم البحث في الأزرار (إرجاع) و (إستلاف) على التوالي:

```
procedure TfmSearch.bbReturnClick(Sender: TObject);
begin
  with dmData.sqSearch do
  if (Active) and (RecordCount > 0) then
    if (FieldByName('IsBorrowed').AsInteger = 1) then
    begin
      fmBookOperation.Caption:= 'إرجاع كتاب';
      fmBookOperation.laBookName.Caption:= FieldByName('BookName').AsString;
      fmBookOperation.BookID:= FieldByName('BookID').AsInteger;
      fmBookOperation.laBorrowLabel.Visible:= False;
      fmBookOperation.edSubscribername.Visible:= False;
      fmBookOperation.edSubscribername.Clear;
      fmBookOperation.edNotes.Clear;
      fmBookOperation.OperationType:= 2; // Return
      fmBookOperation.ShowModal;
    end
    else
      ShowMessage('هذا الكتاب غير مستلف');

end;

procedure TfmSearch.bbBorrowClick(Sender: TObject);
begin
  with dmData.sqSearch do
  if (Active) and (RecordCount > 0) then
    if (FieldByName('IsBorrowed').AsInteger = 0) then
    begin
      fmBookOperation.Caption:= 'إستلاف كتاب';
      fmBookOperation.laBookName.Caption:= FieldByName('BookName').AsString;
      fmBookOperation.BookID:= FieldByName('BookID').AsInteger;
      fmBookOperation.laBorrowLabel.Visible:= True;
      fmBookOperation.edSubscribername.Visible:= True;
      fmBookOperation.edSubscribername.Clear;
      fmBookOperation.edNotes.Clear;
      fmBookOperation.OperationType:= 1; // Borrow
      fmBookOperation.ShowModal;
```

```
    end
    else
      ShowMessage('هذا الكتاب تم إستلافه');
end;
```

طقم السجلات ثنائية الإتجاه Bi Directional Record set

استخدمنا في ماسبق إلى الآن طقم سجلات أحادية الإتجاه Uni Directional Recordset. وهو يعني أن طقم السجلات Recordset مثل الـ SQLQuery إما أن يُستخدم لإرجاع (قراءة) البيانات، أو للتعديل فقط. في بعض الأحيان نحتاج للإثنين معاً، فمثلاً تكون لطقم السجلات القدرة على عرض البيانات في Grid مثلاً وتعديلها مباشرة.
لتحويل طقم السجلات إلى ثنائية الإتجاه ماعلينا إلا مليء خاصية UpdateSql بكود الـ SQL الخاص بالتعديل.

سوف نقوم بتعديل شاشة البحث fmSearch وذلك بإضافة إمكانية تعديل البيانات في شاشة نتيجة البحث مباشرة (جدول البيانات DBGrid). الخطوات هي:

في الحزمة sqSearch نقوم بوضع كود الـ SQL التالي في الخاصية UpdateSQL:

```
update Books set BookName = :BookName, Author = :Author, Publisher = :Publisher,
      Keywords = :Keywords, Copies = :Copies, CopyDate = :CopyDate,
      CopyNum = :CopyNum,  EntryDate = :EntryDate, Info = :Info
      where BookID = :BookID
```

وفي الخاصية InsertSQL نضع الكود التالي:

```
insert into Books (BookName, Author, Publisher, Keywords, Copies, CopyDate,
CopyNum, EntryDate, Info)
      values (:BookName, :Author, :Publisher, :Keywords, :Copies, :CopyDate,
:CopyNum, :EntryDate, :Info)
```

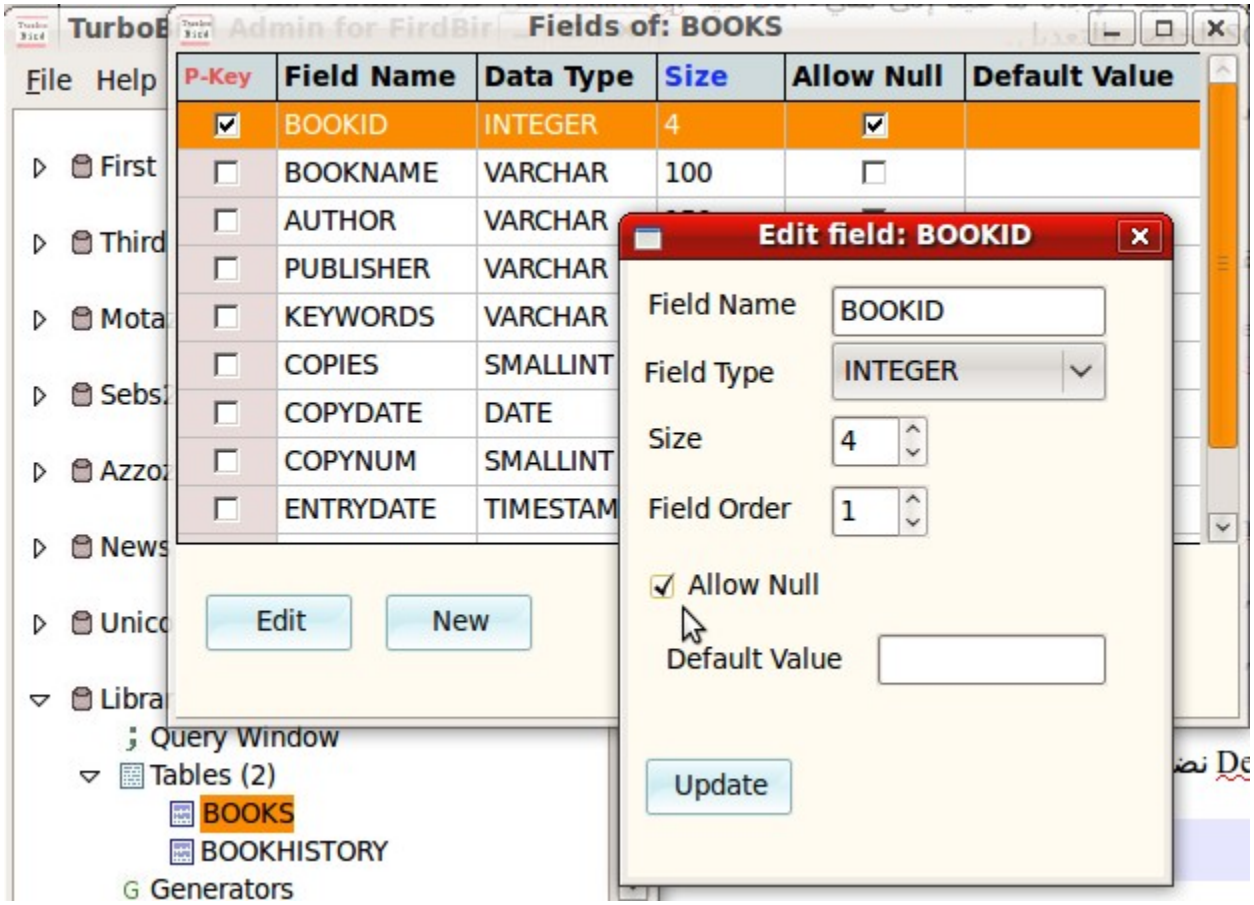
وفي الخاصية DeleteSQL نضع كود الـ SQL التالي:

```
delete from Books
where BookID = :BookID
```

- ثم نقوم بإختيار Allow Null في الحقل BookID في الجدول Books عن طريق:

Turbo Bird/Library/Tables/Books/Table Management/Fields/Edit BookID, Check Allow Null, Update, Run, Commit

كما في الشكل التالي:



وذلك للسماح بعدم إدخال رقم الكتاب BookID عند إضافة كتاب جديد، لأن قاعدة البيانات سوف تقوم بإدخال هذا الرقم تلقائياً عن طريق الـ Trigger, Generator

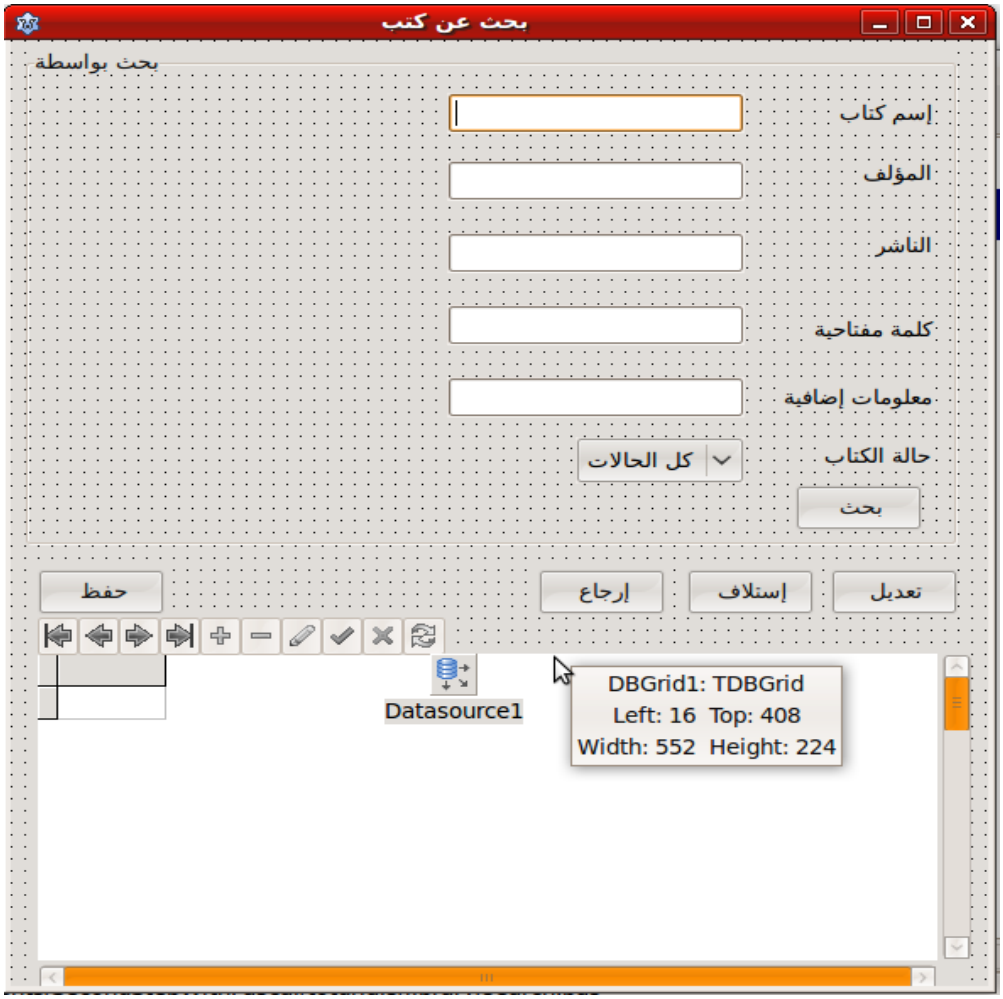
- ثم نقوم بإدراج TDBNavigator من صفحة Data Controls في لازاراس. ونقوم بإختيار DataSource1 في خاصية DataSource.

- ثم نقوم بإضافة زر لحفظ البيانات المعدلة ونضع الكود التالي فيه:

```
dmData.sqSearch.ApplyUpdates;
dmData.SQLTransaction1.CommitRetaining;
```

وبدون إجراء ApplyUpdates لايتم حفظ البيانات المعدلة.

يصبح شكل الفورم كالتالي:



عند تشغيل البرنامج نقوم بالبحث عن كتاب معين أو كل الكتب، بعد ذلك نستطيع تعديل البيانات التي تظهر أمامنا بدون الحاجة للضغط على زر تعديل كما يظهر في الشكل التالي:

وبعد الإنتهاء من التعديل أو الإضافة يجب ضغط الزر حفظ.

ملحوظة: لابد من التأكد من أن خاصية ReadOnly في الحزمة sqSearch قيمتها False وأن جدول البيانات DBGrid1 به خاصية EditOptions/dgEditing قيمتها True. كذلك ReadOnly قيمتها False في جدول البيانات.

تقرير الكتب المُستلفة

التقارير هي من المواضيع المهمة في البرامج الإدارة، حيث أن التقرير هو إظهار معلومة يمكن الإستفادة منها في إتخاذ القرار. فمثلاً، يمكن عمل تقرير بعدد الكتب في المكتبة، ويمكن إصدار تقرير بكمية الإستلاف خلال شهر معين.
وسوف نقوم إن شاء الله بعرض تقرير للكتب المستلفة.

لعمل هذا التقرير نقوم بإضافة حزمة بيانات من نوع TSQLQuery ونربطها مع IBConnection1.
وسميناها sqBorrowed.
في خاصية SQL لهذه الحزمة نكتب كود الـ SQL التالي:

```
select OperationTime as BTime, BookName as Book,
SubscriberName as BorrowedBy
from BookHistory
inner join Books on Books.BookID = BookHistory.BookID
where OperationType = 1 and IsBorrowed = 1
order by OperationTime desc
```

نلاحظ أننا احتجنا لأكثر من جدول لعرض هذه المعلومات، وهي الجداول Books, BookHistory. ومن هنا جاءت إحدى فوائد قاعدة البيانات العلائقية. فنجد أننا في جدول BookHistory لايوجد اسم الكتاب، إنما يوجد رقمه BookID والذي سوف يُستخدم كفتاح ربط (foreign Key) لإستخلاص إسم الكتاب من الجدول الأول.
طريقة الربط هذه تسمى الـ inner join ونبين فيها الجدول المراد الربط به و الحقول المستخدمة للربط بين الجدولين:

```
inner join Books on Books.BookID = BookHistory.BookID
```

كذلك فقد وضعنا شرط أن الكتاب وضعه الآن مستلف (IsBorrowed = 1) في جدول Books وأن نوع العملية في جدول الـ BookHistory قيمتها 1 وتعني إستلاف :

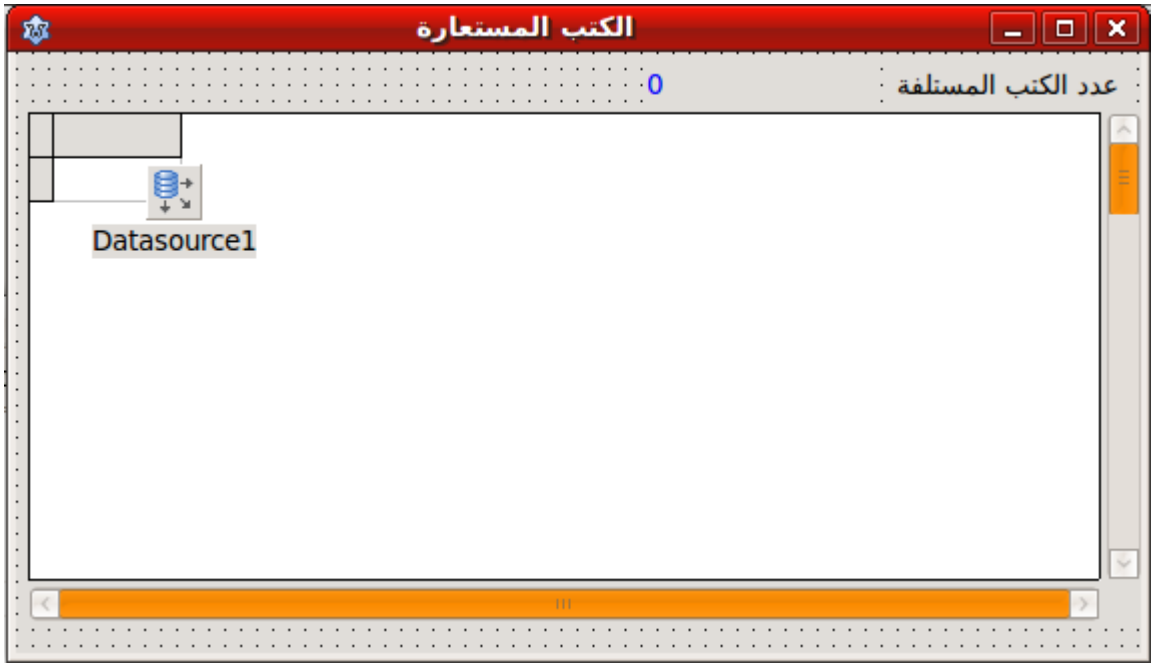
```
where OperationType = 1 and IsBorrowed = 1
```

وقد قمنا بالترتيب التصاعدي بناءً على تاريخ الإستلاف. حيث أن الكتب المستلفة أخيراً تظهر أولاً، والكتب المستلفة قديماً تظهر في نهاية الجدول:

```
order by OperationTime desc
```

بعد ذلك قمنا بإنشاء فورم جديد أسميناه fmBorrow وسمينا الوحدة Borrow. ووضعنا فيه جدول بيانات dbGrid و Label سميناه laCount لعرض عدد الكتب المستلفة.

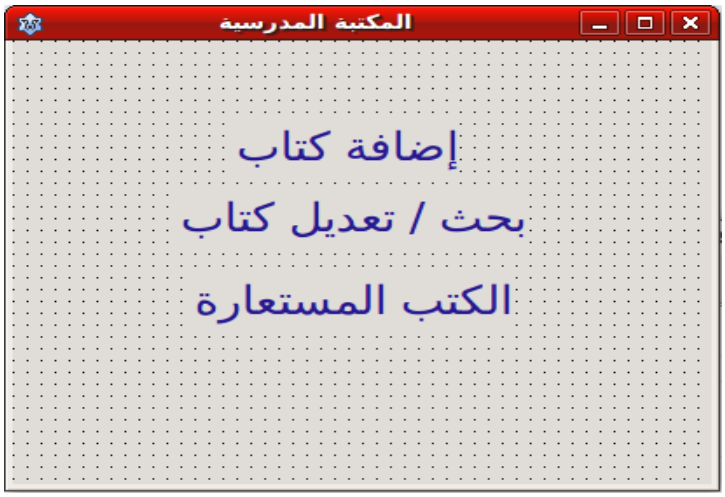
ثم وضعنا DataSource تم ربطها بالحزمة sqBorrowed. ويظهربالشكل التالي:



قمنا بكتابة إجراء في هذا الفورم وسميناه Init ووضعناه في Public حتى يمكن الوصول إليه من الفورم الرئيسي main:

```
procedure TfmBorrowed.Init;
begin
  dmData.sqBorrowed.Close;
  dmData.sqBorrowed.Open;
  dmData.sqBorrowed.Last; // get real count
  laCount.Caption:= IntToStr(dmData.sqBorrowed.RecordCount);
  dmData.sqBorrowed.First; // return back to first record
end;
```

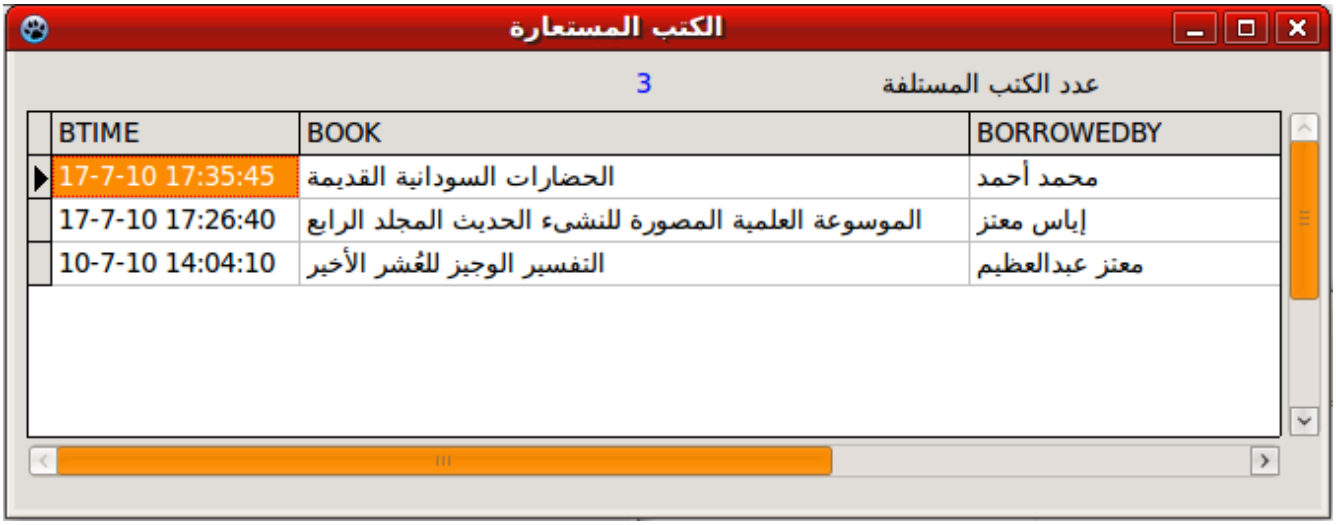
وفي الفورم الرئيسي fmMain وضعنا Label ثالث وكتبنا فيه (الكتب المستعارة)، فأصبح شكل الفورم الرئيسي كالتالي:



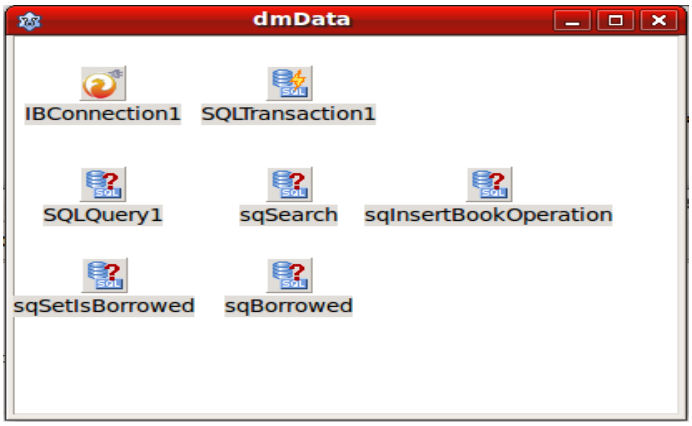
وكتبنا الكود التالي في الـ Label:

```
procedure TfmMain.Label3Click(Sender: TObject);
begin
  fmBorrowed.Init;
  fmBorrowed.ShowModal;
end;
```

عند تشغيل البرنامج يظهر شكل التقرير كالتالي:



وهذا هو الشكل النهائي لحاوية البيانات Data Module المسماة dmData والتي تحتوي على حزم البيانات وإجراءاتها المتسخدمة في البرنامج:



ملحوظة:
يمكن استخدام حقل البيانات الإضافية Info لوصف موقع الكتاب في المكتبة لسهولة الوصول إليه، مثلاً إذا أدخلنا

```
2-5
```

فيمكن أن يعني أن الكتاب موجود في القسم الثاني الرف الخامس.

وفي ختام برنامج المكتبة المدرسية، نجد أننا قمنا بإنشاء نظام لإضافة الكتب والإستلاف والإرجاع والبحث عن الكتب. فهذا النظام يمكن أن يتطور وينمو إلى أن يصبح نظام معقد به كثير من المزايا ويصلح لمعظم المكتبات المدرسية أو غير المدرسية.

برنامج مرآب السيارات

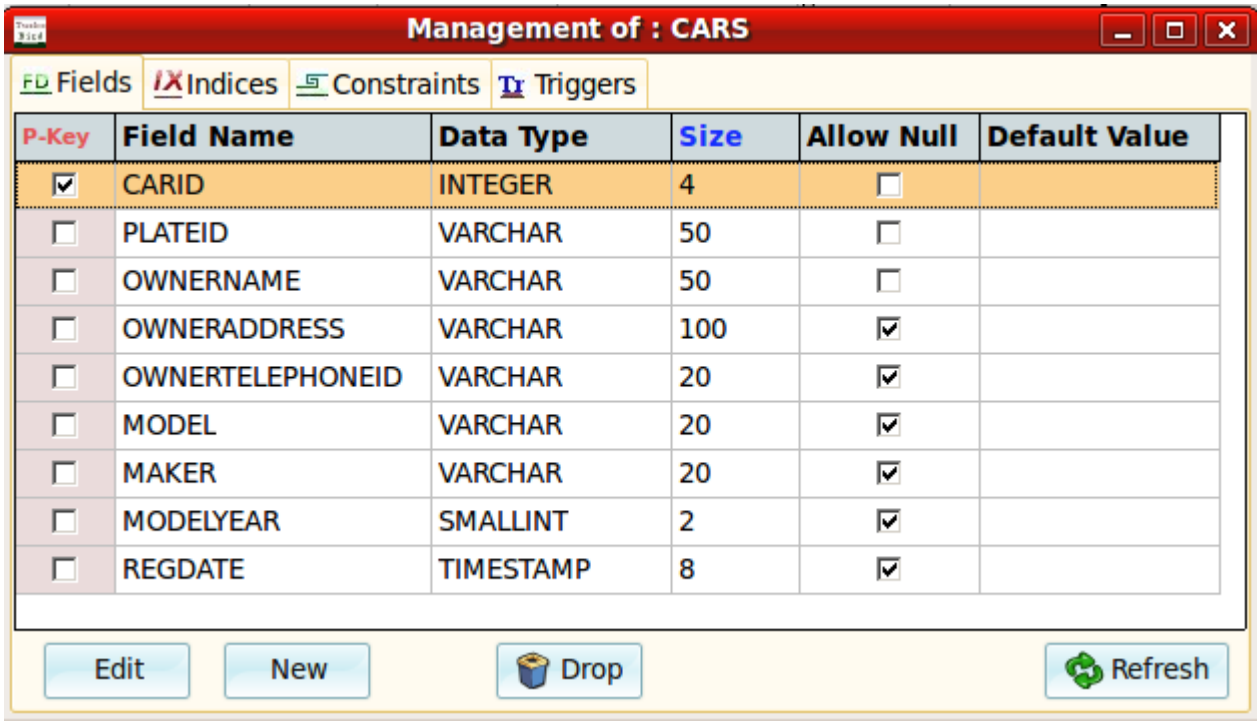
في هذا المثال نريد تصميم برنامج لتسجيل صيانة السيارات في مرآب لفحص وصيانة السيارات.
وفي البداية نقوم بتحليل البرنامج ومعرفة حاجة صاحب المرآب من البرنامج وهي على سبيل المثال:

1. إمكانية تسجيل معلومات سيارة جديدة ، مثل رقم اللوحة، الماركة، الموديل، سنة الصناعة، ومعها بيانات صاحب السيارة، مثل إسمه وعنوانه وهاتفه.

2. إمكانية تسجيل الصيانة التي تمت على هذه السيارة، مثل تغيير الزيت، تغيير قطع غيار.

3. إمكانية استخراج تقرير يمكن طباعته لتاريخ صيانة سيارة ما.

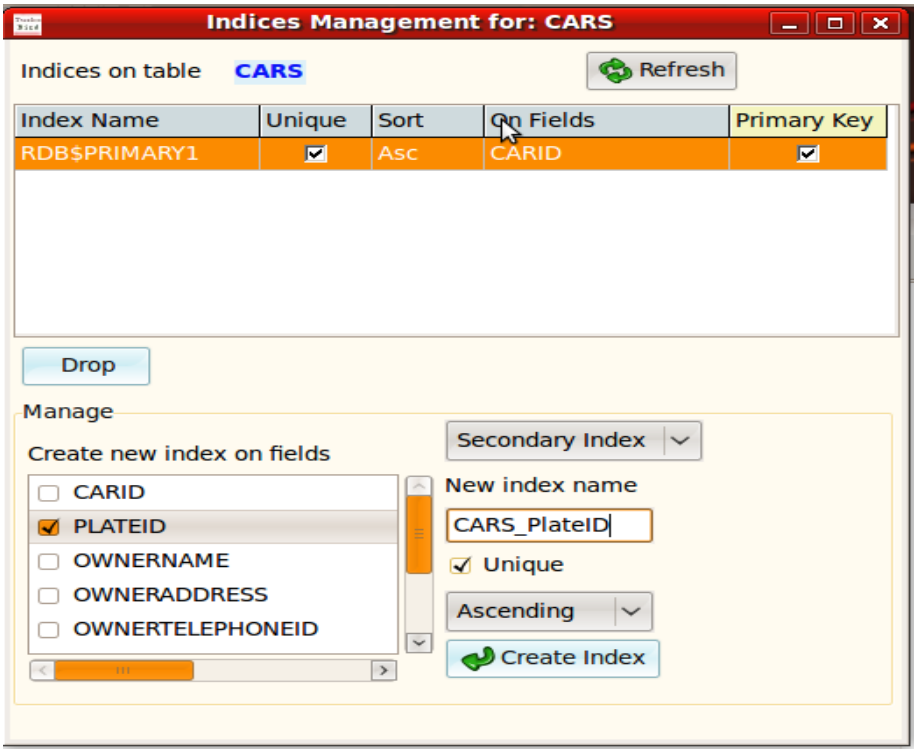
ولتحقيق ذلك، نقوم بإنشاء قاعدة بيانات جديدة نسميها Garage بواسطة برنامج Turbo Bird. ثم نقوم بإنشاء جدول جديد يحتوي على بيانات السيارات نسميه Cars ويحتوي عل الحقول التالية:



P-Key	Field Name	Data Type	Size	Allow Null	Default Value
☑	CARID	INTEGER	4	☐	
☐	PLATEID	VARCHAR	50	☐	
☐	OWNERNAME	VARCHAR	50	☐	
☐	OWNERADDRESS	VARCHAR	100	☑	
☐	OWNERTELEPHONEID	VARCHAR	20	☑	
☐	MODEL	VARCHAR	20	☑	
☐	MAKER	VARCHAR	20	☑	
☐	MODELYEAR	SMALLINT	2	☑	
☐	REGDATE	TIMESTAMP	8	☑	

ثم نقوم بإنشاء Generator مربوط بالحقل المفتاحي CarID ثم إنشاء Trigger لإضافة القيمة التسلسلية تلقائياً.

لم نقوم بجعل رقم اللوحة PlateID هو المفتاح، وذلك بسبب أن لوحة السيارة يمكن تغييرها. إلا أننا مازلنا نريدها أن لاتتكرر. يتم حل هذه المشكلة بإضافة فهرس ثانوي Secondary Index على الحقل PlateID ونعطيه خاصية الإنفراد Unique. وذلك بالوقوف على الجدول Cars ثم إختيار Table Management ثم إختيار صفحة Indices . ثم نقوم بإنشاء فهرس ثانوي على الحقل PlateID ونقوم بإختيار الميزة Unique لضمان عدم تكرار رقم اللوحة:



نقوم بإنشاء برنامج جديد نسمية Garage ونضع فيه الأزرار التالية:

بعد ذلك نقوم بإنشاء حاوية بيانات Data Module نسميها dmData ونسمي الوحدة Data. ثم نقوم بوضع المكونات التالية في حاوية البيانات:

TIBConnection, TSQLTransaction, TSQLQuery

ثم نقوم بربطها بقاعدة البيانات Garage. ونسمي طقم السجلات TSQLQuery بالإسم sqAddCar.

وفي الخاصية SQL في طقم السجلات نضع كود الـ SQL التالي:

```
insert into CARS
(PLATEID, OWNERNAME, OWNERADDRESS, OWNERTELEPHONEID, MODEL, MAKER, MODELYEAR,
REGDATE)
values (:PLATEID, :OWNERNAME, :OWNERADDRESS, :OWNERTELEPHONEID, :MODEL, :MAKER,
:MODELYEAR, Current_TimeStamp );
```

ثم نكتب إجراء إضافة سيارة في حاوية البيانات كالتالي:

```
function TdmData.AddNewCar(PLATEID, OWNERNAME, OWNERADDRESS, OWNERTELEPHONEID,
  MODEL, MAKER: string; MODELYEAR: SmallInt): Boolean;
begin
  try
    sqAddCar.Params.ParamByName('PlateID').AsString:= PlateID;
    sqAddCar.Params.ParamByName('Ownername').AsString:= OWNERNAME;
    sqAddCar.Params.ParamByName('OwnerAddress').AsString:= OWNERADDRESS;
    sqAddCar.Params.ParamByName('OwnerTelephoneID').AsString:= OWNERTELEPHONEID;
    sqAddCar.Params.ParamByName('Model').AsString:= MODEL;
    sqAddCar.Params.ParamByName('Maker').AsString:= MAKER;
    sqAddCar.Params.ParamByName('ModelYear').AsInteger:= MODELYEAR;
    sqAddCar.ExecSQL;
    SQLTransaction1.CommitRetaining;
    Result:= True;


  except
  on e: exception do
  begin
    Result:= False;
    ShowMessage(e.message);
  end;
  end;
end;
```

ثم نقوم بإضافة فورم جديد نسميه fmNewCar ونسمي وحدته NewCar.
نضع فيه مربعات نصوص Tedit و Tlabeles و TspinEdit بالشكل التالي:

ونضع الكود التالي في الزر (إضافة):

```
procedure TfmNewCar.bbAddClick(Sender: TObject);
begin
  if dmData.AddNewCar(Trim(edPlateID.Text), edOwnerName.Text, edOwnerAddress.Text,
    edOwnerTelephoneID.Text,
    edModel.Text, edMaker.Text, seModelYear.Value) then
    begin
      ShowMessage('تمت الإضافة بنجاح');
      ModalResult:= mrOK;
    end;
end;
```

ولاننسى إضافة الوحدة Data في عبارة Uses في هذا الفورم.

للبحث عن سيارة أو مجموعة سيارات نقوم بإضافة فورم جديد نسميه fmSearch ونسمي الوحدة Search. ونضع فيه المكونات التالية:

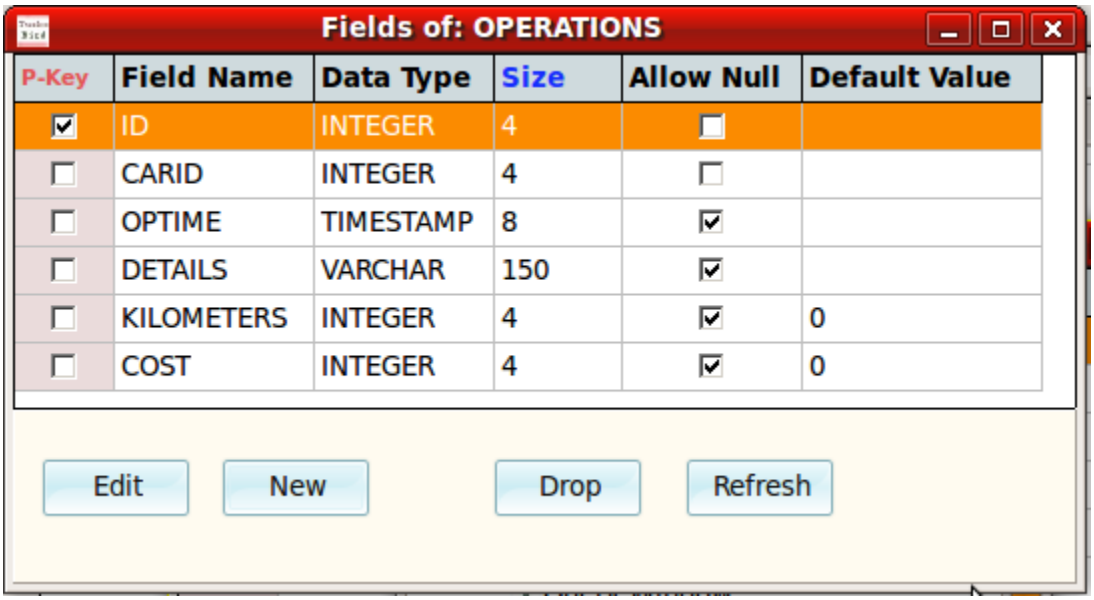
ونقوم بإضافة حزمة بيانات من النوع TSQLQuery في حاوية البيانات dmData. ونربطها بـ IBConnection1. ثم نقوم بكتابة كود البحث التالي في الزر (بحث):

```
procedure TfmSearch.bbSearchClick(Sender: TObject);
begin
  dmData.sqSearch.Close;
  dmData.sqSearch.SQL.Text:= 'select * from Cars where ';
```

```
  if Trim(edPlateID.Text) <> '' then
    dmData.sqSearch.SQL.Add('PlateID like ''%' + edPlateID.Text + '%'' and ');

  if Trim(edOwnerName.Text) <> '' then
    dmData.sqSearch.SQL.Add('OwnerName like ''%' + edOwnerName.Text + '%'' and ');

  dmData.sqSearch.SQL.Add('1 = 1');
  dmData.sqSearch.Open;
  bbMaintinance.Enabled:= dmData.sqSearch.RecordCount > 0;
  bbReport.Enabled:= bbMaintinance.Enabled;
end;
```

حيث يُمكننا هذا الكود من البحث برقم اللوحة أو جزء منها، أو اسم المالك أو جزء من اسمه. وفي حالة وجود أكثر من سيارة في نتيجة البحث تظهر في جدول البيانات dbGrid ثم يمكن إختيار إحداها لإجراء عملية صيانة على هذه السيارة أو إظهار تقرير لها.

بالنسبة للصيانة نقوم بإضافة جدول جديد في قاعدة البيانات Garage نسميه Operations ويحتوي على رقم السيارة التسلسلي والزمن الذي تمت فيه العملية، وتفاصيل تلك العملية، والكيلومترات التي قطعتها السيارة إلى تلك اللحظة، وتكلفة الصيانة. ويكون شكل الجدول كالتالي:



كذلك نقوم بإضافة Generator و Trigger للحقل ID ليكون رقم تسلسلي تلقائي.

بعد ذلك نقوم بإضافة حزمة في حاوية البيانات نسميها sqAddOperation نكتب فيها كود الـ SQL التالي:

```
insert into Operations (CarID, OPTime, Details, Kilometers, Cost)
values (:CarID, Current_TimeStamp, :Details, :Kilometers, :Cost);
```

ثم نقوم بكتابة إجراء الإضافة في هذا الجدول الجديد في حاوية البيانات:

```
function TdmData.AddOperation(CarID, Kilometers, Cost: Integer;
  OperationDetails: string): Boolean;
begin
  try
    sqAddOperation.Params.ParamByName('CarID').AsInteger:= CarID;
```

```
    sqAddOperation.Params.ParamByName('Details').AsString:= OperationDetails;
    sqAddOperation.Params.ParamByName('Kilometers').AsInteger:= Kilometers;
    sqAddOperation.Params.ParamByName('Cost').AsInteger:= Cost;
    sqAddOperation.ExecSQL;
    SQLTransaction1.CommitRetaining;
    Result:= True;
  except
  on e: exception do
  begin
    Result:= False;
    ShowMessage('حدث خطأ أثناء إضافة معلومات الصيانة ' + e.message);
  end;
  end;
end;
```

ثم نقوم بإضافة فورم جديد لكتابة معلومات الصيانة نسميه fmMaintinance بالشكل التالي:

ثم نقوم بكتابة الكود التالي في الزر (إضافة) وذلك بعد إضافة الوحدة Data إلى وحدة هذا الفورم:

```
procedure TfmMaintinance.bbAddClick(Sender: TObject);
begin
  if (Trim(edDetails.Text) = '') or (Trim(edKilometers.Text) = '') or
    (Trim(edCost.Text) = '') then
    ShowMessage('لابد من كتابة التفاصيل وعدد الكيلومترات والتكلفة')
  else
  if dmData.AddOperation(CarID, StrToInt(Trim(edKilometers.Text)),
     StrToInt(Trim(edCost.Text)), edDetails.Text) then
  begin
    ShowMessage('تمت الإضافة بنجاح');
    ModalResult:= mrOK;
  end;
end;
```

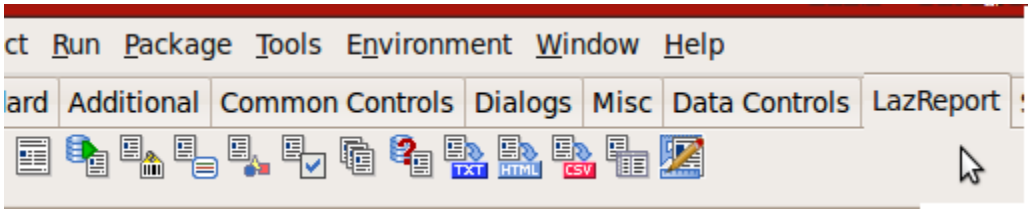
التقارير

التقارير كما سبق ذكرها فهي طريقة إدارية للوقوف على المعلومات، والنتائج، وسير العمل، وغيرها.
وبالنسبة لبرنامج مرآب السيارات، فنحن نريد إظهار تقرير يحتوي على تاريخ صيانة سيارة معينة.
هذه المرة نريد أن نقوم بتصميم تقرير يمكن طباعته بواسطة الطابعة.
سوف نستخدم حزمة مكونات خاصة بالتقارير. وبيئة لازاراس يمكنها إستيعاب أي حزمة للمكونات التي قام بكتابتها فريق لازارس، أو أي طرف ثالث قام بتطويرها. مثلاً توجد حزمة مكونات تسمى LazReport وهي موجودة ضمن ملفات لازاراس إلا أنها غير مضمنة في محرر لازاراس.
ولإضافتها نتبع الخطوات التالية:

1. إختيار Package/Open Package File (.lpk) ثم إختيار الحزمة
إذا كنت تستخدم Ubuntu فإن الحزمة توجد في هذا الدليل:
```
/usr/lib/lazarus/0.9.28.2/components/lazreport/source
```
وإذا كنت تستخدم وندوز فسوف تجده في دليل مثل
```
c:\lazarus\components\lazreport
```

2. إختيار الملف lazreport.lpk

3. نقوم بالضغط على الزر Compile ثم Install والذي يطلب منا إعادة ترجمة بيئة لازاراس نفسها.
بعد الإنتهاء من ترجمة لازاراس يظهر لنا على لوحة المكونات الصفحة التالية:



بهذا نكون قد أضفنا مكونات التقارير إلى بيئة لازاراس.

بعد ذلك نقوم بإدراج حزمة سجلات TSQLQuery في حاوية البيانات dmData ونسميها sqViewOperations ونضع فيها كود الـ SQL التالي:

```
select * from Operations
```

ثم نقوم بفتح طقم السجلات بتحويل الخاصية Active إلى True. وهو إجراء مؤقت حتى يتسنى للتقرير معرفة الحقول.

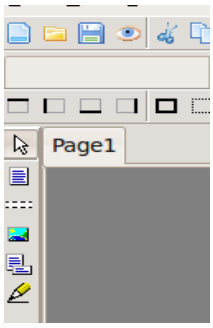
بعد ذلك نقوم بوضع مكونات التقرير TfrReport و TfrReportDataSet في الفورم fmSearch.

في خاصية DataSet في المكون TfrReportDataSet1 نقوم بإختيار sqViewOperations

وفي خاصية DataSet في المكون TfrReport1 نقوم بإختيار frDBDataSet1

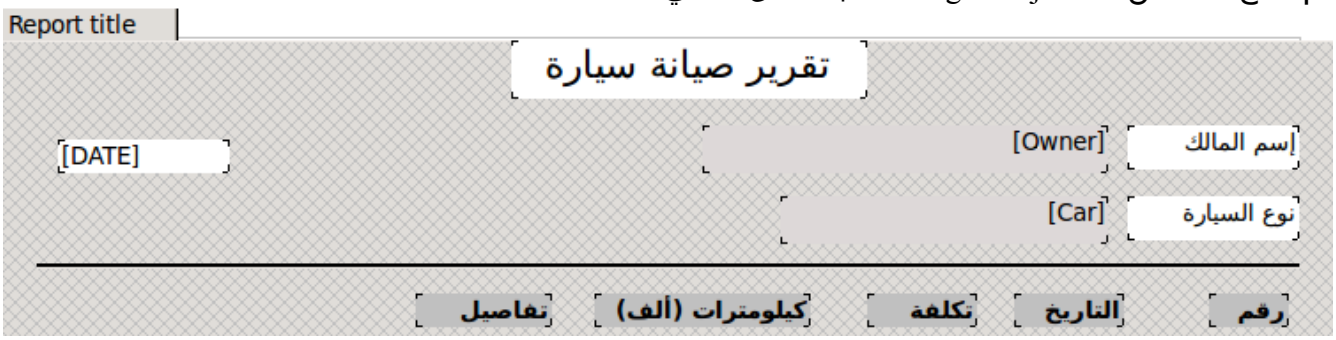
بعد ذلك نقوم بالنقر المزدوج في مكون التقرير TfrReport1 لتظهر لنا شاشة تحرير التقرير. ثم نقوم بحفظ التقرير في نفس الدليل الذي يوجد به البرنامج. نقوم بحفظه بالإسم Operations.lrf.

في الجانب الأيسر من التقرير تظهر لنا مكونات نستخدمها في تصميم التقرير وهي:Band, Rectangle Object



حيث نقوم أولاً بإدراج Band ونختار نوعها Title. ونضع فيها Rectangle Object نكتب فيه عنوان التقرير (تقرير صيانة سيارة) ثم نقوم بتكبير الخط

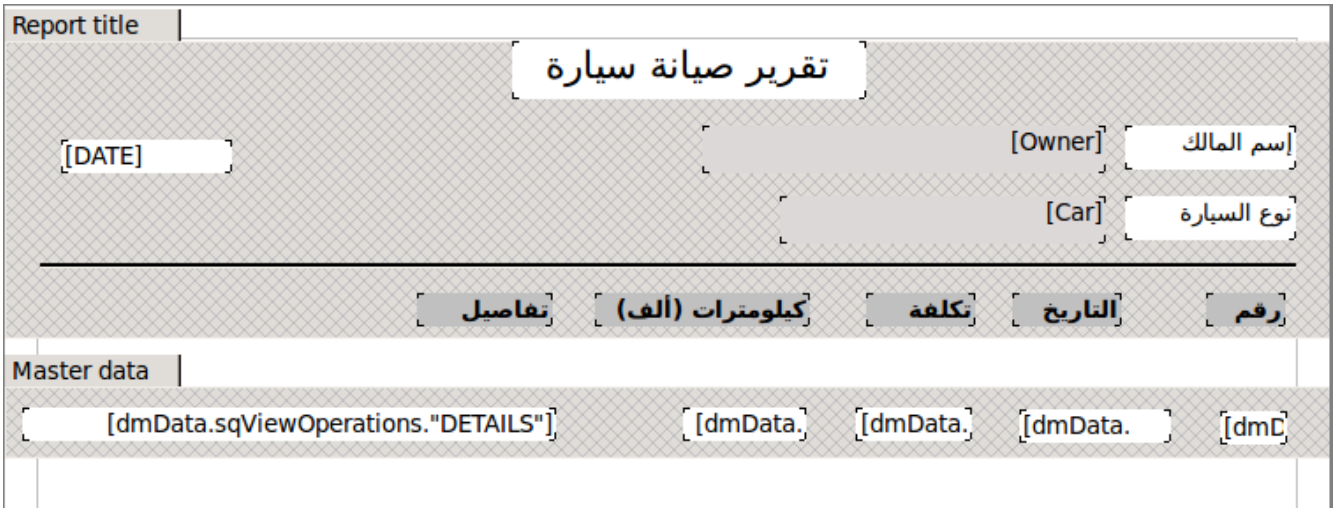
ثم نضع عدد من الـ Rectangle Object بالشكل التالي:



حيث أن [Owner] و [Car] يتم ملئها لاحقاً بواسطة الحدث OnGetValue في مكون التقرير. والكود المكتوب فيه هو:

```
procedure TfmSearch.frReport1GetValue(const ParName: String;
  var ParValue: Variant);
begin
  if ParName = 'Owner' then
    ParValue:= dmData.sqSearch.FieldByName('OwnerName').AsString
  else
    if ParName = 'Car' then
      ParValue:= dmData.sqSearch.FieldByName('Maker').AsString + ' - ' +
        dmData.sqSearch.FieldByName('Model').AsString;
end;
```

ثم نقوم بإدراج Band أخرى من نوع Master Data ثم نقوم بإختيار Tools/Tools/Insert DB Fields من القائمة الرئيسية في مصمم الفورم. ونقوم بإختيار dmData.sqViewOperations ثم نقوم بإختيار الحقول التي نريدها أن تظهر. فيصبح التقرير بالشكل التالي:

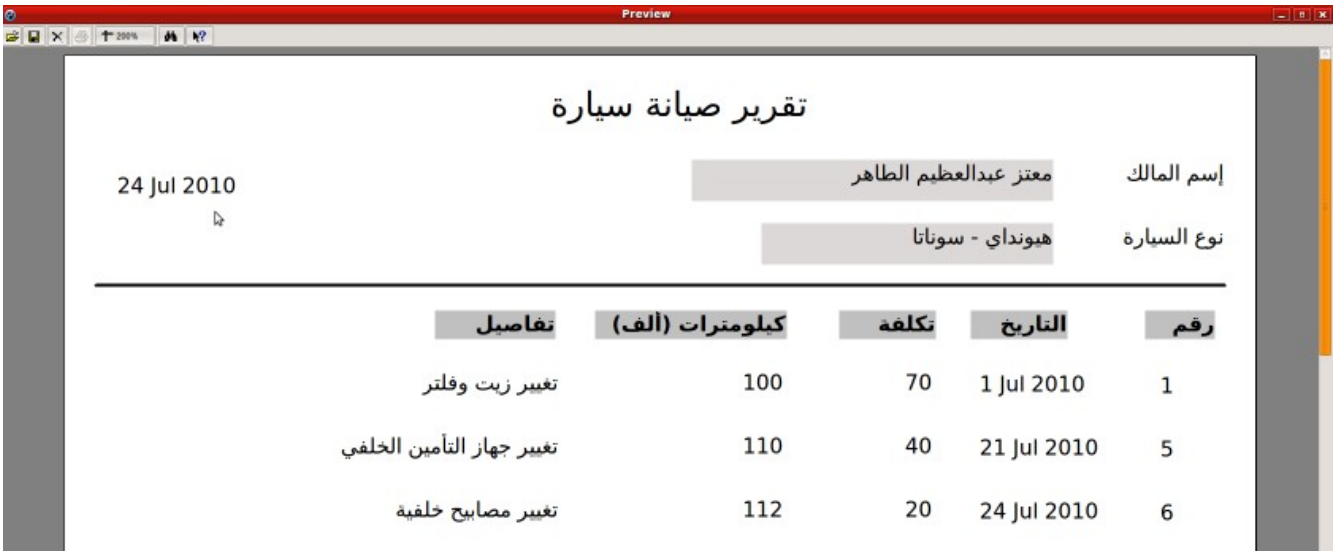


في زر (تقرير صيانة) في الفورم fmSearch نكتب الكود التالي:

```
procedure TfmSearch.bbReportClick(Sender: TObject);
begin
  with dmData do
  begin
    sqViewOperations.Close;
    sqViewOperations.SQL.Text:= 'select * from Operations where CarID = :CarID';
    sqViewOperations.Params.ParamByName('CarID').AsInteger:=
      sqSearch.FieldByName('CarID').AsInteger;
  end;
  frReport1.LoadFromFile('Operations.lrf');
  frReport1.ShowReport;
end;
```

بعد ذلك نقوم بإغلاق طقم السجلات sqViewOperations بواسطة تحويل خاصية Active إلى False ونفس العملية بالنسبة للـ IBConnection1. لأنها سوف تكون قد فُتحت.

عند تشغيل البرنامج نقوم بالبحث عن سيارة معينة ثم نعرض تقرير صيانتها فيظهر عندنا تقرير كالتالي:



الآن البرنامج جاهز ليعمل في مرآب سيارات حقيقي، وسوف يحتاج لبعض الإضافات والتعديلات التي ربما يطلبها صاحب المرآب أو المستخدمين الذين سوف يستخدمون البرنامج. فيعتبر هذا برنامج أولي عام وبسيط.
وكمثال للإضافات التي يمكن عملها هي تعديل بيانات سيارة، و إصدار فاتورة الصيانة الحالية.

حزمة التقارير Fortes

توجد حزمة ثانية للتقارير ذات إمكانات عالية وأكثر إستقراراً وسهولة من حزمة LazReport وهي Fortes وهي مضمنة مع الأمثلة (في الدليل fortes324)

ويمكن تثبيتها في لازاراس بالطريقة العادية:

Package/Open Package file..

وهي تحتاج للحزمة imagesforlazarus ، فإذا لم تكن مُثبته في لازاراس مُسبقاً، يمكن البحث عنها في دليل لازاراس ثم:

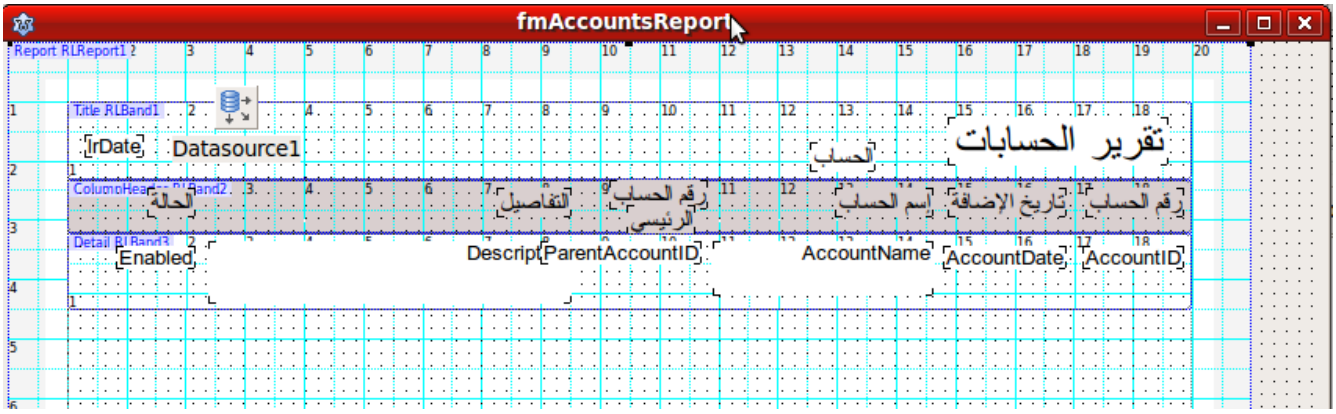
components/images/imagesforlazarus.lpk

في حالة إستخدام وندوز، يجب وضع المكتبة rlreportshared.dll في دليل الوندوز. وهذه المكتبة موجودة في الدليل rlreportdll الموجود ضمن ملفات حزمة fortes.وإلا فلن نستطيع تشغيل لازاراس بعد تثبيت هذه الحزمة.

عند توزيع برامج لازاراس في نظام التشغيل وندوز ، يجب نقل هذه المكتبة مع البرنامج حتى نتمكن من تشغيل البرامج التي تحتوي على تقارير fortes.

بعد تثبيته نجد صفحة Fortes Report ظهرت في بيئة لازاراس. لإستخدام هذه التقارير يجب أولاً إضافة فورم جديد ثم وضع المكون TRLReport. وسوف نتكلم عن هذه التقارير في إصدارة قادمة من الكتاب إن شاء الله.

شكل التقرير أثناء التصميم:



برنامج دفتر اليومية

دفتر اليومية هو عبارة عن أداة من أدوات نظام المحاسبة المالية. فهو دفتر (أو شاشة) يتم فيها تسجيل الحركات المحاسبية المختلفة، مثل دفع فاتورة الهاتف، دفع المرتبات، شراء أثاث، بيع معدات، وغيرها من العمليات المهمة المرتبطة بالمال لشركة ما.

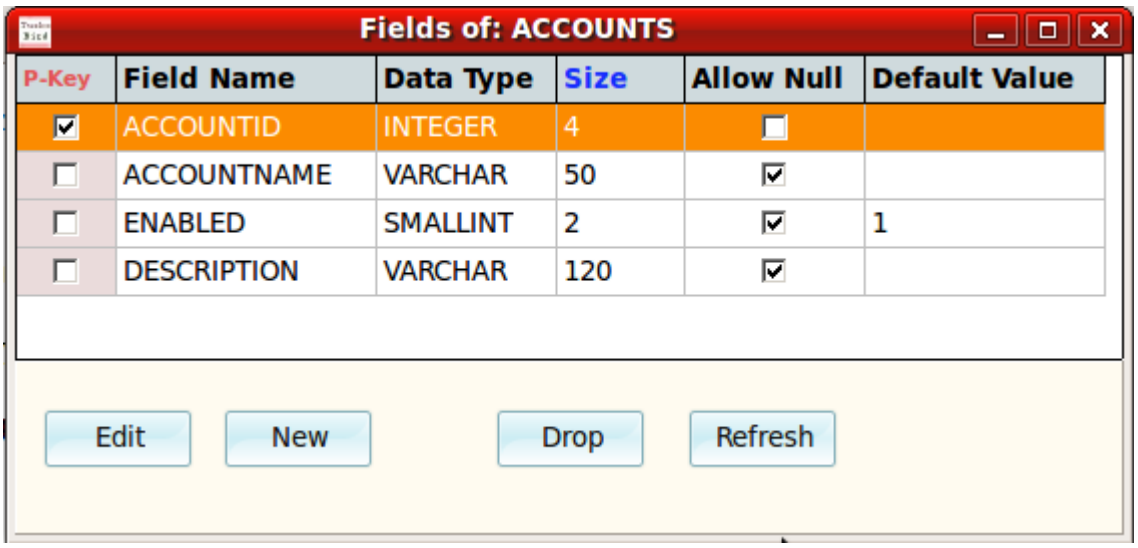
قبل البداية في تصميم وتحليل هذا البرنامج لابد للمبرمج دراسة مباديء المحاسبة المالية، حتى يستطيع فهم مايحتاجه المتسخدم أو المحاسب.

قمنا بإنشاء جدولين: جدول للحسابات Accounts وهو يحتوي على كافة حسابات الشركة، مثل حساب رأس المال، حساب المصروفات، الإيرادات وحتى حساب العملاء الذين يتعاملون بحسابات آجلة مع الشركة.
الجدول الثاني هو جدول العمليات Trans ويتم التسجيل الفعلي فيه لمدخلات دفتر اليومية في شكل قيد مزدوج. مثلاً:

200 جنيه من حساب الهاتف (المصروفات) - مدين
200 جنيه إلى حساب البنك (الأصول المتداولة) – دائن

حيث أن أي عملية تتم يقوم النظام بتسجيلها في حسابين، أولهما حساب مدين Debit وهو الحساب الذي تمت تغذيته بهذه القيمة، والحساب الآخر حساب دائن Credit وهو الحساب الذي أعطى قيمة المال. ففي المثال السابق تم دفع فاتورة الهاتف للشركة بشيك من حساب الشركة والذي يعتبر من الأصول. ففي هذه الحالة لابد أن هذه الحسابات (المصروفات، الإصول المتداولة) موجودة في جدول الحسابات Accounts وإلا تعذر للبرنامج تسجيل هذه المعاملات.

قمنا أولاً بإنشاء قاعدة بيانات جديدة بإسم Accounting. ثم قمنا بإنشاء جدول الحسابات Accounts على الشكل التالي:



وقمنا بإنشاء Generator بإسم AccountIDGen لكن قمنا بإعطائه رقم بداية كبير 1000 بدلاً من القيمة الإبتدائية صفر. وذلك للتفريق بين الحسابات الأساسية والتي سوف تكون من 1 إلى 999. والحسابات الفرعية أو المؤقتة مثل حسابات العملاء فتكون دائماً من الرقم 1000 فما فوق. قمنا بوضع القيمة 1000 في الـ Generator بواسطة كود الـ SQL التالي:

```
set generator AccountIDGen to 1000
```

ثم قمنا بتصميم جدول المعاملات Trans كالتالي:

Fields of: TRANS

P-Key	Field Name	Data Type	Size	Allow Null	Default Value
☑	TRANSID	INTEGER	4	☐	
☐	TRANSTIME	TIMESTAMP	8	☑	
☐	ACCOUNTID	INTEGER	4	☐	
☐	CREDIT	FLOAT	4	☐	
☐	DEBIT	FLOAT	4	☐	
☐	ENTRYID	INTEGER	4	☐	
☐	DETAILS	VARCHAR	120	☑	
☐	USERNAME	VARCHAR	50	☐	

Edit New Drop Refresh

بالنسبة للمعاملة الواحدة، مثلاً دفع فاتورة الهاتف يتم تسجيلها في سجلين منفصلين في هذا الجدول، كل سجل له رقم حساب مختلف AccountID وأحدهما (المدين) نقوم بتسجيل قيمته في الحقل Debit ونقوم بتسجيل القيمة صفر في الحقل Credit. والحساب الآخر (الدائن) نقوم بتسجيل قيمته في الحقل Credit ونضع صفر في الحقل Debit.
أما بالنسبة للحقل EntryID فهو يقوم بالربط بين السجلين السابقين. حيث يتم إعطائهما رقم واحد حتى نستطيع ربطها وإظهارهما سوية في التقارير، وبهذا نعني أن هذين السجلين ينتميان لمعاملة واحدة.

وقمنا بهذه العملية بإنشاء Generator مستقل أسميناه EntryIDGen. وقبل إدخال القيد المزدوج نقوم بإستخراج بإضافة الرقم واحد له ثم إرجاع قيمته بواسطة الكود التالي:

```
select GEN_ID(EntryIDGen, 1) from RDB$Database
```

ثم قمنا بملىء الحسابات الرئيسية في جدول الحسابات حسب شكل الشركة. وكمثال قمنا بإدخال البيانات التالية:

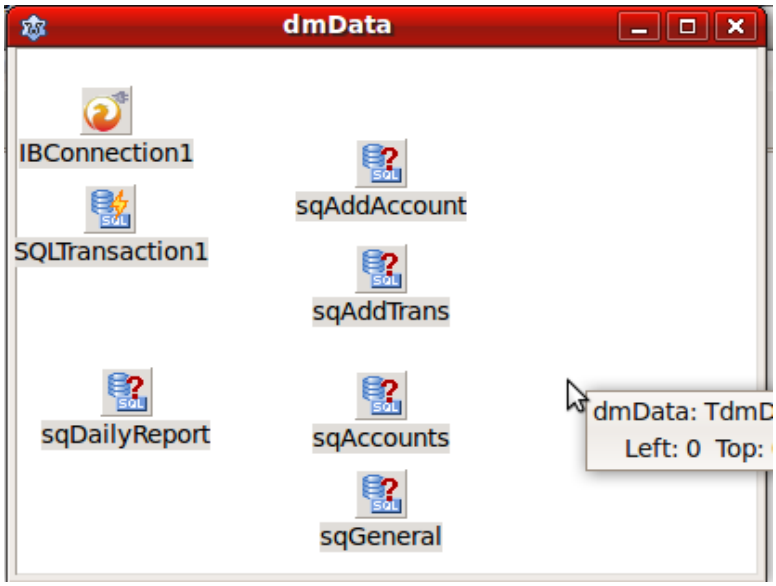
Accounting: Edit Data for Table : ACCOUNTS

Record # 6 of 6

ACCOUNT	ACCOUNTNAME	ENABLED	DESCRIPTION
1	الإصول الثابتة	1	Fixed Assets
5	المصروفات	1	Expenses
11	حقوق الملكية	1	Owners Equity
12	الخصوم	1	Liabilities
13	الإيرادات	1	Revenue
2	الإصول المتداولة	1	Assets

بعد ذلك قمنا بإنشاء برنامج جديد اسميناه Accounting وهذا هو شكل شاشته الرئيسية:

ثم نقوم بإضافة حاوية بيانات وأدرجنا بها مكونات قاعدة البيانات التالية:



والكود المصاحب لها هو:

```
unit Data;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, IBConnection, sqldb, FileUtil, LResources, Forms, Controls,
  Dialogs;

type

  TAccountIDs = array of Integer;
  { TdmData }
```

```
  TdmData = class(TDataModule)
    IBConnection1: TIBConnection;
    sqGeneral: TSQLQuery;
    sqAddAccount: TSQLQuery;
    sqAddTrans: TSQLQuery;
    sqAccounts: TSQLQuery;
    sqDailyReport: TSQLQuery;
    SQLTransaction1: TSQLTransaction;
  private
    { private declarations }
  public
    { public declarations }
    function AddNewAccount(AccountName, Description: string): Boolean;
    function InsertTrans(AccountID, EntryID: Integer; Credit, Debit: Single;
Details: string): Boolean;
    function GetEntryID: Integer;
    procedure FillAccountsBox(var Items: TStrings; var IDs: TAccountIDs);
  end;

var
  dmData: TdmData;

implementation


function TdmData.AddNewAccount(AccountName, Description: string): Boolean;
begin
  try
    sqAddAccount.Params.ParamByName('AccountName').AsString:= Trim(AccountName);
    sqAddAccount.Params.ParamByName('Description').AsString:= Description;
    sqAddAccount.ExecSQL;
    SQLTransaction1.CommitRetaining;
    Result:= True;
  except
  on e: exception do
  begin
    Result:= False;
    ShowMessage('حدث خطأ أثناء إضافة حساب ' + e.message);
  end;
  end;
end;

function TdmData.InsertTrans(AccountID, EntryID: Integer; Credit, Debit: Single;
Details: string): Boolean;
begin
  try
    sqAddTrans.Params.ParamByName('AccountID').AsInteger:= AccountID;
    sqAddTrans.Params.ParamByName('EntryID').AsInteger:= EntryID;
    sqAddTrans.Params.ParamByName('Credit').AsFloat:= Credit;
    sqAddTrans.Params.ParamByName('Debit').AsFloat:= Debit;
    sqAddTrans.Params.ParamByName('Details').AsString:= Details;
    sqAddTrans.ExecSQL;
    Result:= True;
  except
  on e: exception do
  begin
    Result:= False;
```

```
    ShowMessage('حدث خطأ أثناء إضافة قيد ' + e.message);
  end;
  end;
end;

function TdmData.GetEntryID: Integer;
begin
  try
    sqGeneral.Close;
    sqGeneral.SQL.Text:= 'select GEN_ID(EntryIDGen, 1) from RDB$Database';
    sqGeneral.Open;
    Result:= sqGeneral.Fields[0].AsInteger;
    sqGeneral.Close;

  except
  on e: exception do
  begin
    Result:= -1;
    ShowMessage('تعذر الحصول على رقم القيد: ' + e.Message);
  end;

  end;
end;

procedure TdmData.FillAccountsBox(var Items: TStrings; var IDs: TAccountIDs);
begin
  with dmData.sqAccounts do
  begin
    Close;
    Open;
    SetLength(IDs, 0);
    Items.Clear;
    while not Eof do
    begin
      Items.Add(FieldByName('AccountName').AsString);
      SetLength(IDs, Length(IDs) + 1);
      IDs[High(IDs)]:= FieldByName('AccountID').AsInteger;
      Next;
    end;
    Close;
  end;
end;

initialization
  {$I data.lrs}

end.
```

ثم قمنا بإضافة فورم للدخول على البرنامج، حيث أن برامج المحاسبة من البرامج المتطلبة لسرية معينة، فيجب أن لايدخُل على البرنامج إلا من له صلاحية (إسم دخول وكلمة مرور). كذلك فإن القيود لابد أن يتم تسجيل من قام بإدخالها. وسوف نتكلم لاحقاً عن السرية في نظام قاعدة البيانات FireBird وكيفية إضافة إسم مستخدم جديد.
أسمينا هذا الفورم fmPass وبه هذه المكونات:

وفي الزر دخول قمنا بكتابة هذا الكود الذي يسمح بعملية الدخول أو منعه حسب صحة المدخلات:

```
procedure TfmPass.bbLoginClick(Sender: TObject);
begin
  try
    dmData.IBConnection1.UserName:= edUser.Text;
    dmData.IBConnection1.Password:= edPassword.Text;
    dmData.IBConnection1.Open;
    ModalResult:= mrOk;

  except
  on e: exception do
  begin
    MessageDlg('تعذر الدخول: ' + e.Message, mtError, [mbOk], 0);
  end;

  end;
end;
```

ثم قمنا بإضافة فورم جديد لإدراج القيد المزدوج أسميناه fmAddTrans بالشكل التالي:

وهذا هو الكود المصاحب له:

```
unit AddTrans;

{$mode objfpc}{$H+}

interface

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

type

  { TfmAddTrans }

  TfmAddTrans = class(TForm)
    bbOk: TBitBtn;
    cbCreditAccount: TComboBox;
    cbDebitAccount: TComboBox;
    edCreditDetails: TEdit;
    edValue: TEdit;
    edDebitDetails: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    procedure bbOkClick(Sender: TObject);
  private
    { private declarations }
    AccountIDs: array of Integer;
    function Validate: Boolean;
  public
    { public declarations }
    procedure Init;
  end;

var
  fmAddTrans: TfmAddTrans;

implementation

uses Data;

{ TfmAddTrans }

procedure TfmAddTrans.bbOkClick(Sender: TObject);
var
  EntryID: Integer;
begin
  if Validate then
  begin
    EntryID:= dmData.GetEntryID;
    if EntryID <> -1 then
    begin
```

```
      // Insert Credit part
      if dmData.InsertTrans(AccountIDs[cbCreditAccount.ItemIndex], EntryID,
        StrToFloat(Trim(edValue.Text)), 0,  edCreditDetails.Text)
       and
      // Insert Debit part
      dmData.InsertTrans(AccountIDs[cbDebitAccount.ItemIndex], EntryID, 0,
        StrToFloat(Trim(edValue.Text)), edDebitDetails.Text) then
        begin
          // Commit both records
          dmData.SQLTransaction1.Commit;
          ShowMessage('تمت إضافة القيد المزدوج بنجاح');
          ModalResult:= mrOK;
        end;

    end;
  end;
end;


function TfmAddTrans.Validate: Boolean;
begin
  Result:= False;
  if Trim(edValue.Text) = '' then
    ShowMessage('يجب إدخال المبلغ')
  else
  if cbCreditAccount.ItemIndex = -1 then
    ShowMessage('يجب إختيار الحساب المدين')
  else
  if cbDebitAccount.ItemIndex = -1 then
    ShowMessage('يجب إختيار الحساب الدائن')
  else
  if Trim(edCreditDetails.Text) = '' then
    ShowMessage('لابد من كتابة تفصيل العملية المدينة')
  else
  if Trim(edDebitDetails.Text) = '' then
    ShowMessage('لابد من كتابة تفصيل العملية الدائنة')
  else
    Result:= True;

end;

procedure TfmAddTrans.Init;
begin
  with dmData.sqAccounts do
  begin
    Close;
    Open;
    SetLength(AccountIDs, 0);
    cbCreditAccount.Clear;
    cbDebitAccount.Clear;
    while not Eof do
    begin
      cbCreditAccount.Items.Add(FieldByName('AccountName').AsString);
      SetLength(AccountIDs, Length(AccountIDs) + 1);
      AccountIDs[High(AccountIDs)]:= FieldByName('AccountID').AsInteger;
      Next;
    end;
```

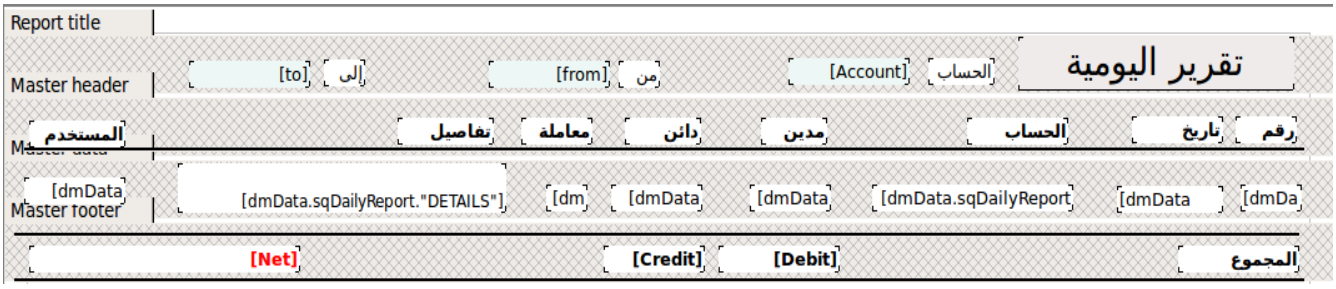
```
    Close;
    cbDebitAccount.Items.Text:= cbCreditAccount.Items.Text;
  end;
end;


initialization
  {$I addtrans.lrs}

end.
```

ثم قمنا بإدراج فورم إضافة حساب جديد أسميناه fmAddAccount به المكونات التالية:

والكود المصاحب لزر الإضافة هو:

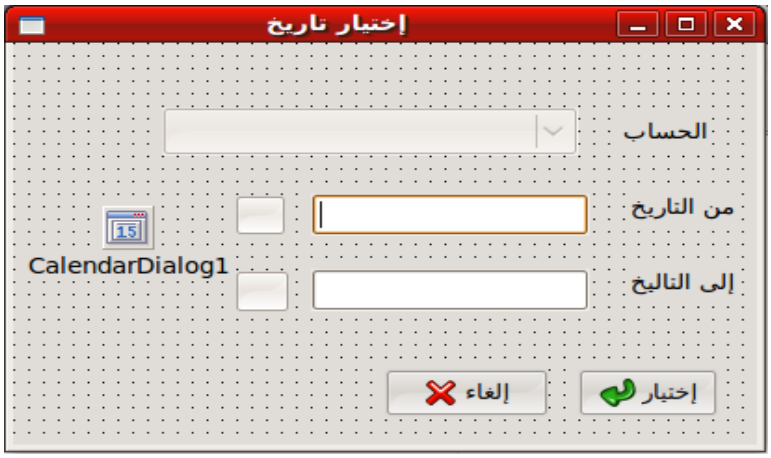
```
procedure TfmAddAccount.bbAddClick(Sender: TObject);
begin
  if (Trim(edAccountName.Text) <> '') and (Trim(edDetails.Text) <> '') then
  begin
    if dmData.AddNewAccount(edAccountName.Text, edDetails.Text) then
    begin
      ShowMessage('تمت إضافة الحساب الجديد بنجاح');
      ModalResult:= mrOK;
    end;
  end
  else
    ShowMessage('يجب إدخال كل الحقول');
end;
```

بعد ذلك قمنا بتصميم تقرير اليومية بالشكل التالي:



وقد صممناه بصورة مستعرضة Landscape حتي نتمكن من إضافة تفاصيل أكثر للسطر الواحد.

ثم قمنا بإدراج فورم لإختيار الحساب الذي نريد عرض معاملاته في التقرير وتاريخ المعاملات كالتالي:



وهذا هو الكود المصاحب للفورم:

```
unit SelDate;

{$mode objfpc}{$H+}

interface

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

type

  { TfmSelDate }

  TfmSelDate = class(TForm)
    bbSelect: TBitBtn;
    bbCancel: TBitBtn;
    CalendarDialog1: TCalendarDialog;
    cbAccount: TComboBox;
    edFromDate: TEdit;
    edTo: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
  private
    { private declarations }
  public
```

```
    { public declarations }
  end;

var
  fmSelDate: TfmSelDate;

implementation

{ TfmSelDate }

procedure TfmSelDate.FormCreate(Sender: TObject);
begin
  edFromDate.Text:= DateToStr(Date);
  edTo.Text:= DateToStr(Date);
end;

procedure TfmSelDate.SpeedButton1Click(Sender: TObject);
begin
  if CalendarDialog1.Execute then
    edFromDate.Text:= DateToStr(CalendarDialog1.Date);
end;

procedure TfmSelDate.SpeedButton2Click(Sender: TObject);
begin
  if CalendarDialog1.Execute then
    edTo.Text:= DateToStr(CalendarDialog1.Date);
end;

initialization
  {$I seldate.lrs}

end.
```

وفي الفورم الرئيسي fmMain قمنا بكتابة الكود الذي عن طريقه ننادي جميع شاشات البرنامج المختلفة والتقرير. وهذا هو الكود المصاحب للفورم الرئيسي:

```
unit main;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs,
  Buttons, LR_Class, LR_DBSet;

type

  { TfmMain }

  TfmMain = class(TForm)
    bbDailyReport: TBitBtn;
    bbAddEntry: TBitBtn;
    bbAddAccount: TBitBtn;
    frDBDataSet1: TfrDBDataSet;
    frReport1: TfrReport;
```

```
    procedure bbAddAccountClick(Sender: TObject);
    procedure bbAddEntryClick(Sender: TObject);
    procedure bbDailyReportClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure frReport1GetValue(const ParName: String; var ParValue: Variant);
  private
    Activated: Boolean;
    CreditTotal, DebitTotal: Double;
    { private declarations }
  public
    { public declarations }
  end;

var
  fmMain: TfmMain;

implementation

uses Pass, AddTrans, AddAccount, SelDate, Data;

{ TfmMain }

procedure TfmMain.FormActivate(Sender: TObject);
begin
  if not Activated then
  begin
    if fmPass.ShowModal <> mrOk then
      Close;
    Activated:= True;
  end;
end;

procedure TfmMain.bbAddEntryClick(Sender: TObject);
begin
  fmAddTrans.Init;
  fmAddTrans.ShowModal;
end;

procedure TfmMain.bbDailyReportClick(Sender: TObject);
var
  IDs: array of Integer;
begin
  dmData.FillAccountsBox(fmSelDate.cbAccount.Items, IDs);
  fmSelDate.cbAccount.Items.Insert(0, 'كل الحسابات');
  fmSelDate.cbAccount.ItemIndex:= 0;
  if fmSelDate.ShowModal = mrOK then
  with dmData.sqDailyReport do
  begin
    Close;
    Sql.Text:= 'select TransID, TransTime, AccountName, Credit, Debit, ' +
      'EntryID, Details, UserName ' +
      ' from Trans ' +
      'inner join accounts on Accounts.AccountID = Trans.AccountID ' +
      'where ';
    if fmSelDate.cbAccount.ItemIndex <> 0 then // Sepecific Account
     SQL.Text:= SQL.Text + 'Trans.AccountID = :AccountID and';
```

```
    SQL.Text:= SQL.Text + 'TransTime >= :FromDate and TransTime < :ToDate ' +
      'order by EntryID, Debit ';
    Params.ParamByName('FromDate').AsDate:= StrToDate(fmSelDate.edFromDate.Text);
    Params.ParamByName('ToDate').AsDate:= StrToDate(fmSelDate.edTo.Text) + 1;
    if fmSelDate.cbAccount.ItemIndex <> 0 then
      Params.ParamByName('AccountID').Value:=
        IDs[fmSelDate.cbAccount.ItemIndex - 1];
    Open;
    CreditTotal:= 0;
    DebitTotal:= 0;
    while not Eof do
    begin
      CreditTotal:= CreditTotal + FieldByName('Credit').AsFloat;
      DebitTotal:= DebitTotal + FieldByName('Debit').AsFloat;
      Next;
    end;
    First;
    frReport1.LoadFromFile('daily.lrf');
    frReport1.ShowReport;
    Close;
  end;
end;

procedure TfmMain.bbAddAccountClick(Sender: TObject);
begin
  fmAddAccount.ShowModal;
end;

procedure TfmMain.FormCreate(Sender: TObject);
begin
  Activated:= False;
end;

procedure TfmMain.frReport1GetValue(const ParName: String; var ParValue: Variant
  );
var
  Net: Double;
  ALabel: string;
begin
  if ParName = 'Credit' then
    ParValue:= Format('%3.0n', [CreditTotal])
  else
  if ParName = 'Debit' then
    ParValue:= Format('%3.0n', [DebitTotal])
  else
  if ParName = 'from' then
    ParValue:= fmSelDate.edFromDate.Text
  else
  if ParName = 'to' then
    ParValue:= fmSelDate.edTo.Text
  else
  if ParName = 'Account' then
    ParValue:= fmSelDate.cbAccount.Text
  else
  if ParName = 'Net' then
  begin
```

```
    Net:= DebitTotal - CreditTotal;
    if Net = 0 then
      ALabel:= ''
    else
    if Net > 0 then
      ALabel:= 'مدين'
    else
      ALabel:= 'دائن';
    ParValue:= Format('%3.0n %s', [Abs(Net), ALabel]);
  end;
end;

initialization
  {$I main.lrs}

end.
```

مثال للتقرير اليومي لشركة حاسوب:

تقرير اليومية

الحساب كل الحسابات من 08/27/2010 إلى 08/28/2010

رقم	تاريخ	الحساب	مدين	دائن	معاملة	تفاصيل	المستخدم
9	08/27/2010	الإصول المتداولة	1500	0	5	إستلام مبلغ بيع جهاز حاسوب في الخزينة	AHMED
10	08/27/2010	الإيرادات	0	1500	5	بيع جهاز حاسوب	AHMED
15	08/27/2010	شركة الخيال	1200	0	8	شراء شاشة وملحقات حاسوب بالآجل	AHMED
16	08/27/2010	الإيرادات	0	1200	8	بيع شاشة وملحقات حاسوب	AHMED
17	08/27/2010	الإصول المتداولة	700	0	9	إستلام في الخزنة	AHMED
18	08/27/2010	شركة الخيال	0	700	9	دفع جزء من فاتورة شراء شاشة وملحقات	AHMED
19	08/27/2010	المصروفات	100	0	10	دفع فاتورة الكهرباء	AHMED
20	08/27/2010	الإصول المتداولة	0	100	10	صرف من الخزينة للكهرباء	AHMED
21	08/27/2010	المصروفات	250	0	11	دفع فاتورة الهاتف	AHMED
22	08/27/2010	الإصول المتداولة	0	250	11	صرف شيك فاتورة هاتف	AHMED
23	08/27/2010	الإصول الثابتة	1000	0	12	شراء مكتب	AHMED
24	08/27/2010	الإصول المتداولة	0	1000	12	شراء مكتب بشيك من البنك	AHMED
25	08/27/2010	الإصول المتداولة	120	0	13	إستلام مبلغ بيع قرص صلب	KHALID
26	08/27/2010	الإيرادات	0	120	13	بيع قرص صلب	KHALID
المجموع			4,870	4,870		0	

السرية في قاعدة البيانات FireBird

لاحظنا في البرنامج السابق أننا استخدمنا إسم دخول وكلمة مرور، وإسم الدخول هذا هو إسم الدخول المستخدم في قاعدة البيانات FireBird حيث يمكننا إستخدام إسم الدخول SYSDBA، لكن من الأفضل إنشاء إسم لكل مستخدم على حده وإعطائه صلاحية محدودة، فقط ليستطيع إدخال معاملات وإضافة حسابات بالنسبة لبرنامج دفتر اليومية.

سوف نقوم بعرض كيفية إضافة إسم مستخدم جديد في نظام لينكس توزيعة أوبونتو:

إذا كانت قاعدة بيانات فيربيرد رقم نسختها 2.5 أو أكبر، يمكن إضافة إسم المستخدم بكل سهولة عن طريق برنامج توربو بيرد، وذلك بإختيار Users بالزر اليمين ثم إختبار Create New User.
وبعد كتابة إسم المستخدم وكلمة المرور يجب إختيار Role، وهي ACCRole، لكن على أن نكون قد قمنا بإنشاء هذه الـ Role بالخطوات 4 و 5 التالية:

أما إذا كانت نسخة قديمة، مثلاً 2.1 فيجب إتباع الخطوات التالية

1. نقوم بالدخول في الدليل :

```
/usr/lib/firebird/2.1/bin
```

2. ثم نقوم بتنفيذ الأمر التالي:

```
/usr/lib/firebird/2.1/bin$ ./gsec -user sysdba -password masterkey
```

حيث أن المستخدم sysdba له صلاحية إضافة مستخدمين.
وعند ظهور المحث

```
GSEC>
```

3. نقوم بإضافة المستخدم الجديد بواسطة الأمر:

```
add khalid -pw 0091
```

بهذا نكون قد أضفنا مستخدم جديد إسمه khalid وكلمة مروره 0091

4. بعد ذلك نستخدم برنامج Turbo Bird لإنشاء Role إسمها ACCROLE نعطيها لكل مستخدم جديد. بإستخدام Query Window:

```
CREATE ROLE ACCROLE;
```

5. ثم نعطي هذه الـ Role صلاحية على الجداول كالآتي:

```
GRANT INSERT, REFERENCES, SELECT
 ON Accounts TO  ACCROLE;

GRANT INSERT, REFERENCES, SELECT
 ON Trans TO  ACCROLE;
```

6. بعد ذلك نعطي المستخدم khalid صلاحية على الـ Role بهذه الطريقة:

```
grant ACCROLE to khalid
```

كان يمكن أن نعطي المستخدم khalid هذه الصلاحيات مباشرة دون الحاجة لوسيط ACCROLE إلا أن إستخدام الـ Role يعطينا فرصة لِتكون الصلاحيات مركزية. فمثلاً إذا كان عندنا مائة مستخدم وأضفنا جدول جديد فماعلينا إلا تعديل صلاحيات ACCROLE فنكون قد غيرنا صلاحية كافة المستخدمين المائة الذين يستخدمون هذه الـ ROLE بخطوة واحدة فقط.

ولاننسى كتابة إسم الـ Role في مكون قاعدة البيانات المستخدم IBCONNECTION1 في خاصية Role.

في المرات التالية عند إضافة مستخدم جديد لانحتاج للخطوات 4 و 5 لأن الـ Role تكون موجودة مسبقاً ولها الصلاحية المطلوبة.

نظم المحاسبة المالية

البرنامج السابق (دفتر اليومية) على بساطته إلا أنه يصلح ليكون نواة وحجر أساس لنظام محاسبي كبير يمكن إستخدامه في عدد من الشركات والمؤسسات بعد تطويره وزيادة التقارير وإمكانية إدخال للقيود بطريقة أسهل. فمثلاً يمكن إضافة شاشة لإدراج المصروفات، وأخرى للمبيعات، وهكذا، بدلاً من شاشة واحدة عامة عيبها أن المستخدم يمكن أن يُخطيء فيها فيقوم بإدخال مصروفات على أنها أرباح مثلاً. وذلك لأننا في هذه الشاشة العامة أعطيناه صلاحية إختيار الحسابات المراد إدراج المعاملة تحتها. أما الشاشات الخاصة بنوع معين فيمكن أن تكون نوع الحسابات مضمنة داخلها، أو تظهر للإطلاع فقط دون التغيير.

تختلف نظم المحاسبة المالية من الشركات إلى المصانع إلى المحاسبة الحكومية، فلكل خواصه وإحتياجاته، فعلى المبرمج أن يقوم بدراسة حاجة المؤسسة المعنية من التقارير والوظائف ليقوم بتطبيقها في هذا النظام حتى يكون الإستخدام مباشر وأكثر عملية بالنسبة لهذه المؤسسة.

بعد نشر هذا الكتاب قمت بتطوير برنامج دفتر اليومية ليصبح برنامجاً يُمكن إستخدامه في الشركات الصغيرة، حيث قمت بتطوير التقارير وطريقة الإدخال، فيمكن للمستخدم إضافة قوالب للإدخال والتقارير، مثلاً يمكن إضافة قالب إدخال المصروفات، حيث يقوم المستخدم فقط بكتابة قيمة المصروفات وتفاصيلها بدون أن يقوم بإختيار الحساب المدين والدائن. كذلك يمكن عمل قالب تقرير للمصروفات، وآخر لحساب البنك، إلخ. وكذلك تمت إضافة الشيكات بكامل دوراتها.

وسوف أقوم بنشر هذا البرنامج الجديد في موقع code.sd قريباً إن شاء الله.

الفصل الرابع

برامج الويب

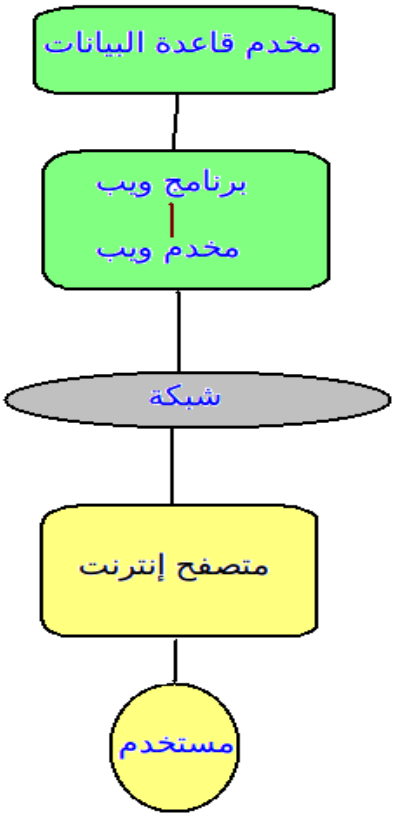
Web Applications

مقدمة

برامج الويب هي عبارة عن برامج تستخدم متصفحات الإنترنت كواجهة للمستخدم، ومخدمات الويب (Web Servers) كوسيط لتشغيل هذه البرامج. فبدلاً من تصميم برنامج وإنزاله إلى كافة المستخدمين يكفينا فقط تعريفهم بوصلة الويب URL التي فيها البرنامج أو عنوان الموقع التفاعلي الذي يمثل برامج الويب.
نقلت تفنية برامج الويب الإنترنت نقلة نوعية، فبعدما كانت الأخيرة تعتمد على صفحات ثابتة Static pages أصبح من الممكن عمل صفحات متفاعلة مع المستخدم. حيث أصبح من الممكن تنفيذ إجراءات على هذه المخدمات مثل البحث عن معلومة، عمل منتديات، وشراء عن طريق الإنترنت وغيرها من المواقع والخدمات التفاعلية.

تتميز برامج الويب عن البرامج العادية بالآتي:

1. لايحتاج المستخدم إلى إنزال برنامج، فقط يمكنه إستخدام أي متصفح للإنترنت للتعامل مع البرنامج
2. عند تحديث أو عمل إضافة في برنامج الويب، يتم تحديثه فقط في مخدم الويب، ولانحتاج لعمل أي تغيير من جانب المستخدم
3. لايهم نظام التشغيل أو المنصة الموجودة في جهاز المستخدم، فيمكن أن يكون نظام لينكس ،أو وندوز، أو ماكنتوش أو I-Phone، كذلك يمكن أن يُستخدم أي متصفح مثل فيرفوكس، كروم، أو أوبرا.
4. يتم عزل المستخدمين عن قاعدة البيانات، فبما أن برامج المخدم العميل تحتاج لأن يتصل جهاز المتسخدم بمحرك قاعدة البيانات مباشرة عن طريق الشبكة، فإن برامج الويب تتطلب فقط أن يتصل مخدم الويب بمخدم قاعدة البيانات، أما المستخدمين فيجب أن يتصلوا فقط بمخدم الويب.



وهذا هو مثال لهيكل نظام ويب متكامل. وأجزائه هي عبارة عن برامج وليس أجهزة، فمثلاً مخدم قاعدة البيانات هو برنامج محرك قاعدة البيانات، يمكن أن يوجد مع مخدم الويب في نفس الجهاز والذي هو بدوره برنامج آخر، ويمكن أن تكون قاعدة البيانات توجد في جهاز آخر.
كذلك فإن متصفح الإنترنت هو برنامج. ويمكن أن تكون هذه البرامج في أجهزة منفصلة تربط بينهما شبكة إتصال أو يمكن أن تكون في جهاز واحد، فمثلاً أثناء تصميم وتطوير برنامج ويب يمكن أن تكون كل هذه الأجزاء في جهاز المبرمج حتى تسهل عليه عملية البرمجة والإختبار.

مخدم الويب Web Server

أول خطوة في بداية رحلتنا مع برامج الويب هو الحصول على برنامج مخدم ويب، وهو الوسيط بين المستخدم الذي يتعامل مع المتصفح وبرنامج الويب الذي يلبي طلبات المستخدمين الطرفيين.
وأشهر برنامج يعمل كمخدم ويب هو الـ Apache Web Server وهو برنامج حُر يمكن الحصول عليه من الموقع التالي www.apache.org أو كتابة الأمر التالي في نظام أوبونتو:

```
sudo apt-get install apache2
```

وفي نظام فيدورا يمكن الحصول عليه أثناء التثبيت أو قرص الـ DVD أو بإستخدام الأمر

```
yum install apache2
```

بعد التثبيت نتأكد من أن البرنامج يعمل

```
service apache2 status
```

فإذا تحصلنا على النتيجة التالية فهي تعني أن البرنامج يعمل:

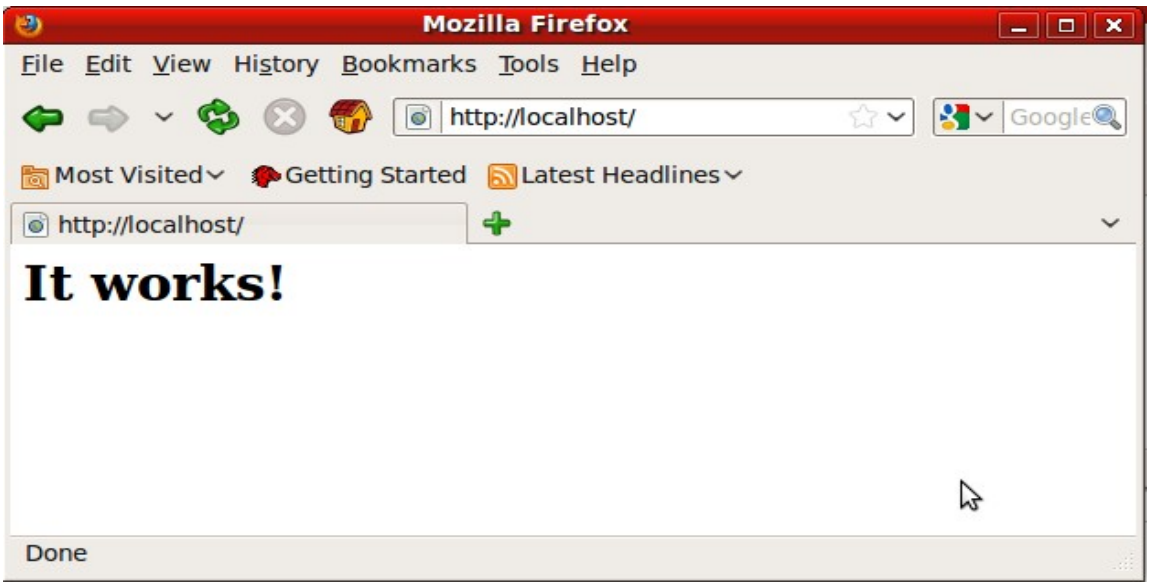
```
 * Apache is running (pid 2119)
```

وإلا قمنا بتشغيله عن طريق الأمر التالي:

```
sudo /etc/init.d/apache2 start
```

بالنسبة لنظام وندوز فيجب إنزال البرنامج من الموقع المذكور، وسوف نجد أيقونة تُمككنا من تشغيل أو إيقاف البرنامج.

ثم نقوم بفتح متصفح الإنترنت ونكتب فيه العنوان http://localhost أو http://127.0.0.1 لنجد العبارة التالية:



بعد ذلك نتأكد من أن الدليل cgi-bin موجود ومهيء بصورة صحيحة. ففي نظام وندوز نجده على هذا الدليل مثلاً:

```
c:\program files\Apache Group\...\cgi-bin
```

وفي نظام أوبونتو نجده على الدليل:

```
/usr/lib/cgi-bin
```

وفي نظام فيدورا نجده على الدليل:

```
/var/www/cgi-bin
```

ثم نقوم بفتح ملف التهيئة apache2.conf الموجود في نظام أوبونتو في هذا الدليل:

```
/etc/apache2
```

وفي وندوز إسم الملف httpd.conf وموجود في الدليل:

```
c:\program files\Apache Group\...\conf
```

ثم نبحث عن cgi-bin ونتأكد من أنه مهيء بهذه الطريقة مثلاً (في نظام أوبونتو):

```
ScriptAlias /cgi-bin/ "/usr/lib/cgi-bin/"


<Directory "/usr/lib/cgi-bin/">
  AllowOverride None
  Options ExecCGI
  Order allow,deny
  Allow from all
</Directory>
```

بعد ذلك نقوم بعمل أول برنامج ويب بسيط بإستخدام لازاراس:

برنامج الويب الأول

نقوم بإنشاء برنامج جديد نوعه program ونسميه مثلاً firstweb نكتب فيه الكود التالي:

```
program firstweb;

{$mode objfpc}{$H+}

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

```
{$IFDEF WINDOWS}{$R firstweb.rc}{$ENDIF}

begin
  Writeln('CONTENT-TYPE: TEXT/HTML');
  Writeln;
  Writeln('Hello, this is my first <b>Web Application</b>');
  Writeln('<br/> Server time is: <font color=green>' +
    DateTimeToStr(Now) + '</font>');
end.
```

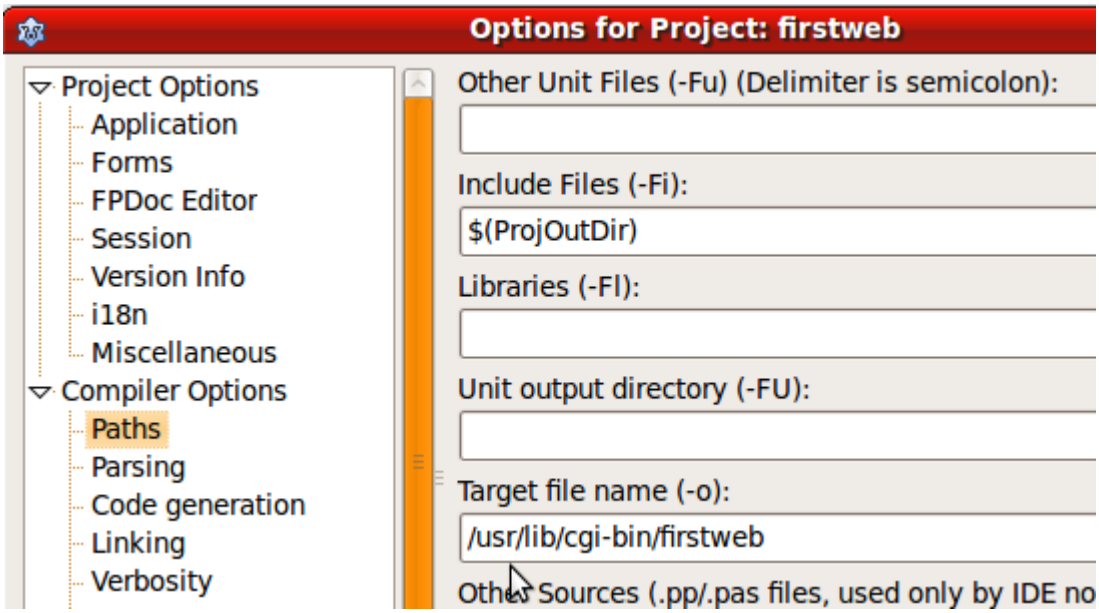
في حالة نظامي لينكس وماكنتوش نقوم بتغيير مسار مخرجات الترجمة إلى

```
/usr/lib/cgi-bin/firstweb
```

وفي حالة وندوز نكتب:

```
/usr/lib/cgi-bin/firstweb.exe
```

وذلك عن طريق Project/Project Options/Compiler Options/Paths/Target File Name/ كما في الشكل التالي:



ثم نقوم بترجمة البرنامج Ctrl-F9، ثم نقوم بكتابة الرابط التالي في المتصفح:

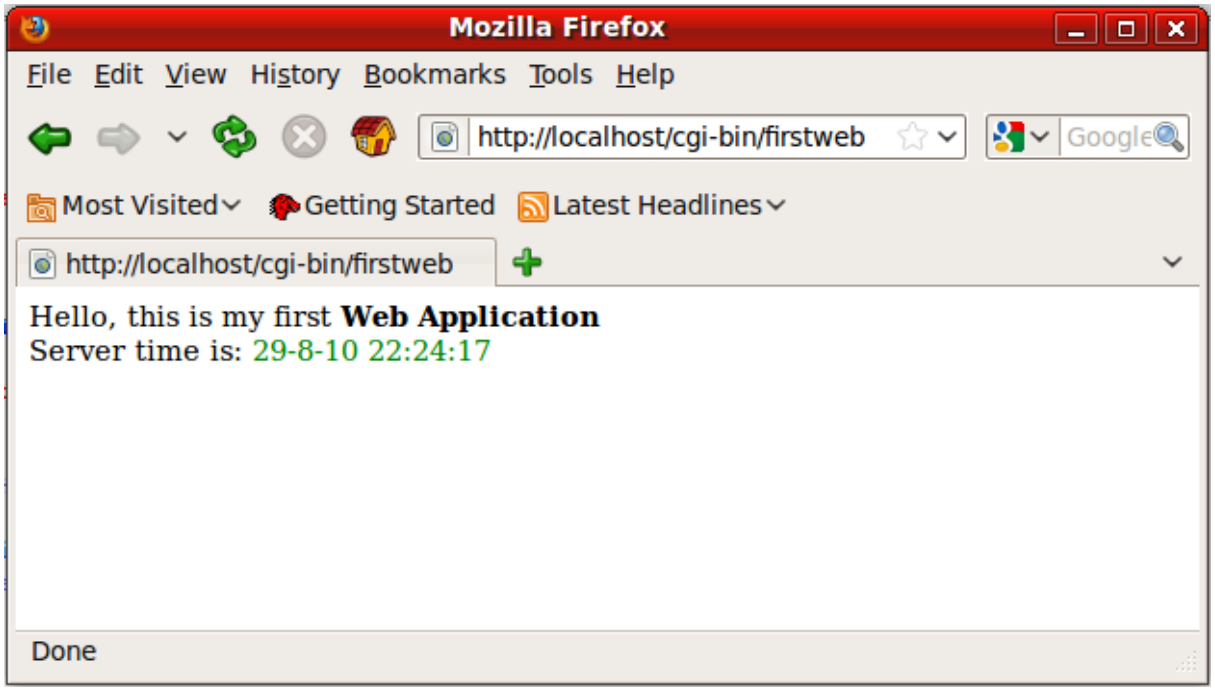
http://localhost/cgi-bin/firstweb

أو :

http://localhost/cgi-bin/firstweb.exe

حسب نظام التشغيل في المخدم. فإذا كان المخدم لينكس مثلاً نكتب الرابط الأول، وإذا كان وندوز نكتب الرابط الثاني. أما عند تشغيل البرنامج في شبكة فإن نظام التشغيل في جهاز العميل (الطرفية) لايؤثر، فقط العبرة في نظام تشغيل المخدم.

فيظهر عندنا ناتج أول برنامج ويب بالشكل التالي:



نلاحظ أننا قمنا بكتابة المخرجات في شكل ترميز HTML وهو الترميز المستخدم لبرامج الويب، أو هو الكود الذي يفهمه المتصفح لعرض المعلومات.
هذا العرض هو عبارة عن ناتج تنفيذ برنامج وليس صفحة ثابتة، والدليل أننا كلما قمنا بالضغط على المفتاح F5 أي إعادة طلب العنوان مرة أخرى فإن المحتويات تتغير (يتغير الوقت). وفي كل مرة نقوم فيها بطلب هذا العنوان يقوم المتصفح بإرسال هذا الطلب إلى مخدم الويب الذي بدوره يقوم بتشغيل البرنامج firstweb وقراءة مخراجاته ثم إرسالها إلى المتصفح. ويمكن أن تكون نتيجة تشغيل البرنامج هي معلومة من قاعدة بيانات، قراءة من ملف، أو تعديل بيانات أو أي من الإجراءات التي يمكن تنفيذها بواسطة أوبجكت باسكال. وهذا هو ماترتكز عليه برامج الويب.

برتوكول الـ CGI

البرتوكول الذي استخدمناه في البرنامج السابق هو برتوكول CGI وهو إختصار Common Gateway Interface وهي أول تقنية دعمت برامج الويب. وهي مدعومة بواسطة عدد كبير من لغات البرمجة مثل C, C++, Perl, Pascal, وغيرها، حيث أنها تمتاز بالبساطة وأنها مدعومة بواسطة عدد كبير من برامج مخدمات الويب، وهذه التقنية ليست مُحتكرة لجهة ما.

حزمة Free Spider

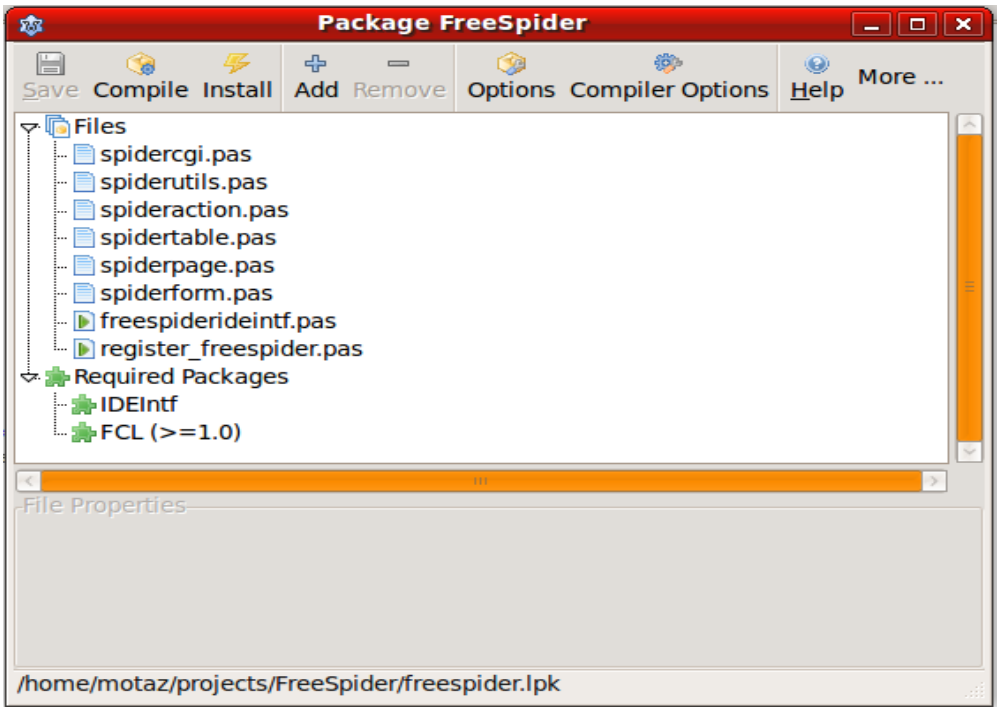
لإنتاج برامج ويب متطورة ومعقدة وبصورة سريعة لانستخدم الطريقة السابقة التي تتعامل مع برتوكول الـ CGI مباشرة، بل يجب إستخدام منصة تطوير ومكتبات متقدمة. لذلك يجب علينا إختيار تقنية تستخدم إحدى برتوكولات برامج الويب. وسوف نستخدم في هذا الكتاب حزمة Free Spider التي

يمكن الحصول عليها من موقع www.code.sd (وهي أيضاً موجودة مع الأمثلة) ثم نثبتها بالطريقة التالية في بيئة لازاراس:

- بعد إنزال FreeSpider وفكها في القرص الصلب، نقوم بتشغيل لازاراس ثم إختيار:

```
Package/Open Package File (*.lpk)
```

- ثم نختار الملف FreeSpider.lpk فتظهر لنا الشاشة التالية:



- ثم نقوم بترجمة الحزمة Compile ، ثم تثبيتها Install

هذه الخطوة تتطلب إعادة ترجمة وربط بيئة لازاراس. بعد ذلك تكون حزمة FreeSpider جاهزة للإستخدام.

برنامج FreeSpider الأول

- من بيئة لازاراس نختار

```
Project/New Project/FreeSpider CGI Web Application
```

- ثم نقوم بحفظ المشروع في الدليل MySpider: الوحدة الرئيسية نسميها main.pas والمشروع نسميه myspider

- بعد ذلك نقوم بإدراج الكائن TSpiderCGI من صفحة FreeSpider في حاوية البيانات DataModule1

- ثم نقوم بالنقر المزدوج على حاوية البيانات أو إختيار الحدث OnCreate ثم كتابة الكود التالي:

```
procedure TDataModule1.DataModuleCreate(Sender: TObject);
begin
  SpiderCGI1.Execute;
end;
```

- ثم نقوم بالنقر المزدوج على الكائن SpiderCGI1 أو إختيار الحدث OnRequest وكتابة الكود التالي:

```
procedure TDataModule1.SpiderCGI1Request(Sender: TObject;
  Request: TSpiderRequest; var Response: TSpiderResponse);
begin
  Response.Add('Hello from <b>MySpider</b> web application <br/>');
  Response.Add('Time in server is: ' + DateTimeToStr(Now));
end;
```

- ثم نقوم بتغيير مسار ترجمة وربط البرنامج إلى :

```
/usr/lib/cgi-bin/myspider
```

إذا كنا نستخدم أوبونتو، أو

```
/var/www/cgi-bin
```

إذا كنا نستخدم فيدورا.

وذلك بتغييره في Project Options كما أسلفنا.

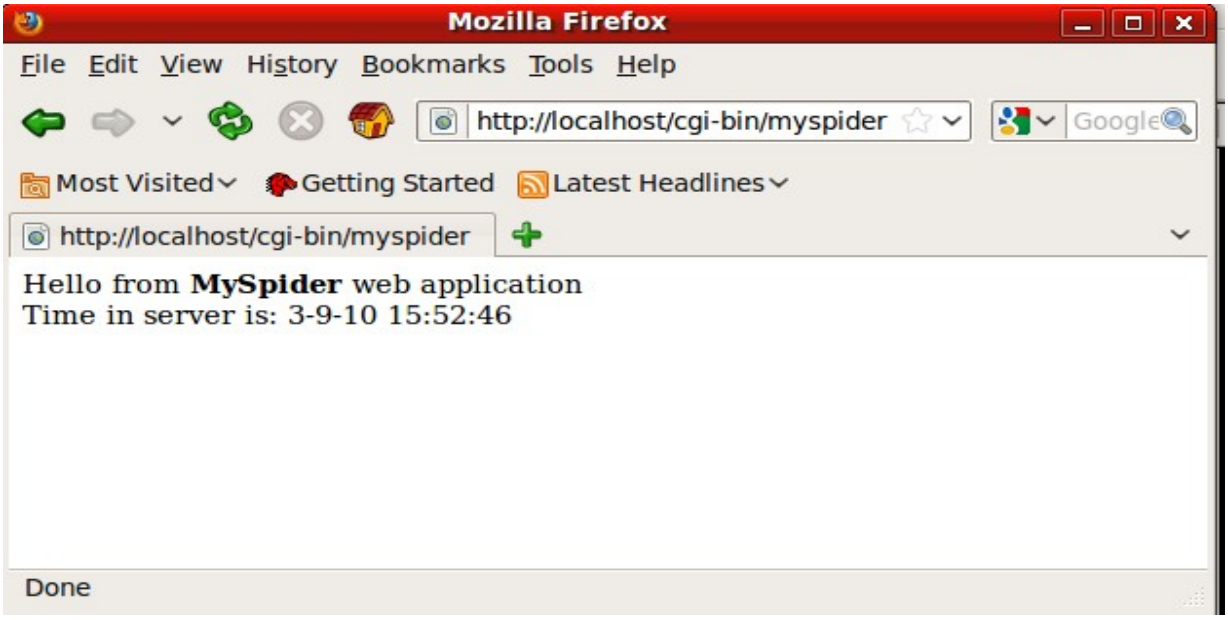
- بعد ذلك نقوم بترجمة البرنامج، ثم فتح المتصفح وكتابة العنوان التالي في نظامي لينكس وماكنتوش :

```
http://localhost/cgi-bin/myspider
```

وفي نظام وندوز نكتب الرابط التالي:

```
http://localhost/cgi-bin/myspider.exe
```

لنحصل على الرد التالي:



إستخدام المُدخلات

من الأشياء المهمة التي تُبين الصفحات الثابتة من الصفحات الديناميكية أو برامج الويب، هي إمكانية إرسال مدخلات لبرامج الويب. فكما أن برامج الأوامر النصية يمكن إدخال مدخلات لها بواسطة براميترات مثل:

```
ls -lh /etc
```

فكذلك يمكن مناداة برنامج الويب السابق عن طريق الـ URL مثل:

```
http://localhost/cgi-bin/myspider?name=Motaz
```

وفي كود البرنامج نستخدم الكائن Request والذي يحتوي على معلومات مستخدم برنامج الويب بما فيها المدخلات التي أرسلها. فمثلاً يمكن الوصول إلى المتغير Name المرسل عن طريق المتصفح بالكود التالي:

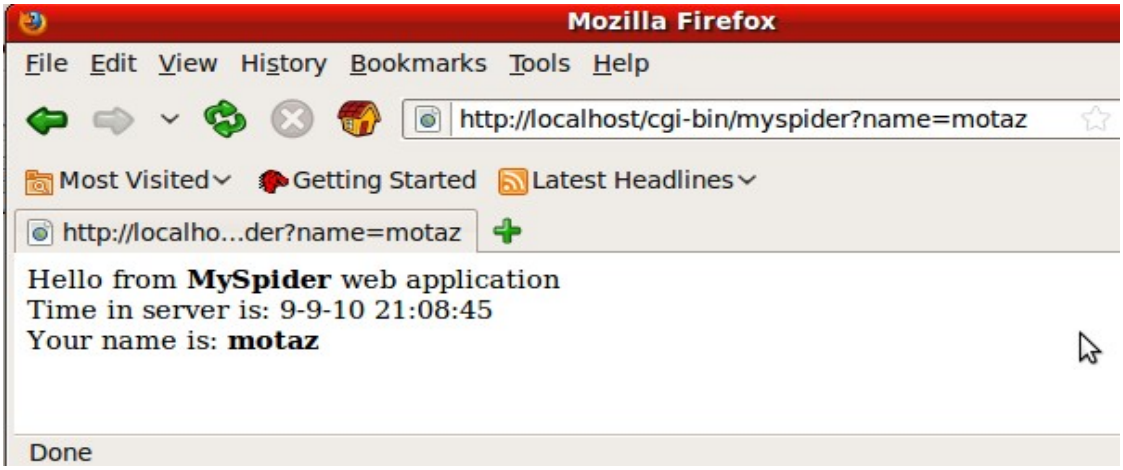
```
Request.Query('name');
```

فيصبح الكود في الحدث OnRequest بالنسبة للـ SpiderCGI1 كالتالي:

```
procedure TDataModule1.SpiderCGI1Request(Sender: TObject;
  Request: TSpiderRequest; var Response: TSpiderResponse);
begin
  Response.Add('Hello from <b>MySpider</b> web application <br/>');
  Response.Add('Time in server is: ' + DateTimeToStr(Now));
  Response.Add('<br/>Your name is: <b>' + Request.Query('name') + '</b>');
```

```
end;
```

وتكون النتيجة كالتالي في المتصفح:



كذلك يمكن إرسال أكثر من مُدخل بالفصل بينها بالرمز & مثلاً:

```
http://localhost/cgi-bin/myspider?name=Motaz&Address=Khartoum
```

ونقوم بتعديل الكود كالتالي:

```
  Response.Add('<br/>Your name is: <b>' + Request.Query('name') + '</b> ');
  Response.Add('I live in : <b>' + Request.Query('address') + '</b>');
```

فتكون النتيجة كمايلي:

Hello from **MySpider** web application
Time in server is: 9-9-10 20:55:35
Your name is: **Motaz** I live in : **Khartoum**

إستخدام صفحة ثابتة

يمكن كذلك تضمين هذه المدخلات داخل صفحة ويب ثابتة **HTML**، فمثلاً نقوم بإنشاء صفحة جديدة بواسطة أي محرر نصوص وإذا كنا نستخدم أوبونتو نحفظه بإسم **list.htm** في الدليل:

```
/var/www
```

أو عند إستخدام فيدورا نضعه في الدليل:

```
/var/www/html
```

وعند إستخدام وندوز نضعه في الدليل:

```
c:\program files...apache..\htdocs
```

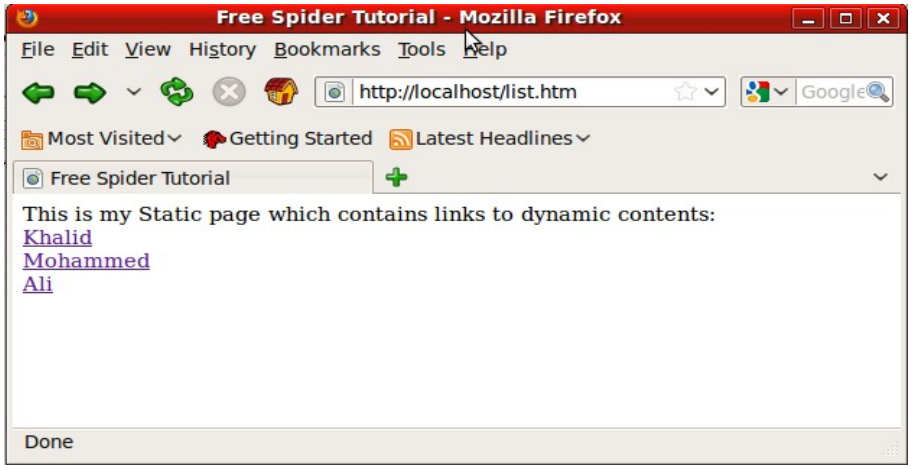
ونص الصفحة هو:

```
<html>
<head><title>Free Spider Tutorial</title></head>
<body>
This is my Static page which contains links to dynamic contents:
<br/>
<a href="/cgi-bin/myspider?name=Khalid&address=Bahri">Khalid</a><br/>
<a href="/cgi-bin/myspider?name=Mohammed&address=Omdurman">Mohammed</a><br/>
<a href="/cgi-bin/myspider?name=Ali&address=Port Sudan">Ali</a><br/>
</body>
</html>
```

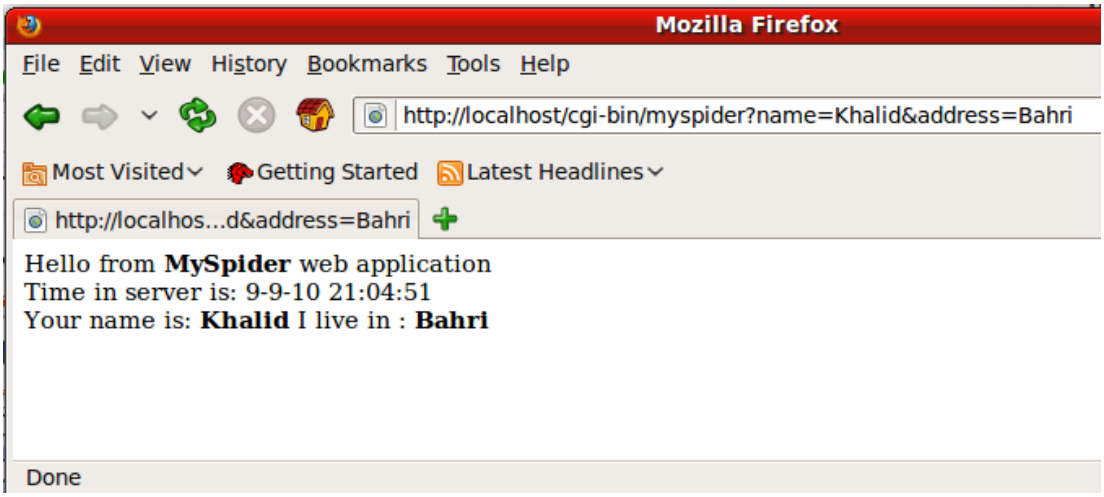
ونقوم بعرض هذه الصفحة في المتصفح بواسطة العنوان:

```
http://localhost/list.htm
```

فنحصل على الشكل التالي في المتصفح:



وعند الضغط على الإسم الأول نحصل على :



نلاحظ أن العنوان ظهر فيه المدخلات التي تم إرسالها من الصفحة السابقة **list.htm**

المكون Action

في المثال السابق إستخدمنا فقط إجراء واحد للحدث **OnRequest** في المكون **TSpiderCGI**. مع العلم أن برنامج **FreeSpider** يحتوي فقط على مكون **SpiderCGI** واحد، لذلك يتعذر إستخدام أكثر من حدث **OnRequest** بإستخدام هذا المكون.

لإستخدام أكثر من حدث للتعامل مع طلبات الويب، يجب إستخدام المكون **TSpiderAction** .
نقوم بإدراج المكون **TspiderAction** في حاوية البيانات. وفي الخاصية **Action** نكتب القيمة:

```
/firstaction
```

ثم نكتب الكود التالي في الحدث **OnRequest** بالنسبة لـ **SpiderAction1**:

```
procedure TDataModule1.SpiderAction1Request(Sender: TObject;
  Request: TSpiderRequest; var Response: TSpiderResponse);
begin
  Response.Add('This is response from <b>SpiderAction</b> component');
end;
```

ثم نقوم بترجمة البرنامج.

ولمناداة هذا الحدث **Action** من المتصفح نكتب العنوان بالشكل التالي:

```
http://localhost/cgi-bin/myspider/firstaction
```

فتظهر لنا النتيجة التالية في المتصفح:

This is response from **SpiderAction** component

يمكن كذلك إستخدام مدخلات في هذا الحدث وتكون صيغتها في العنوان كالمثال التالي:

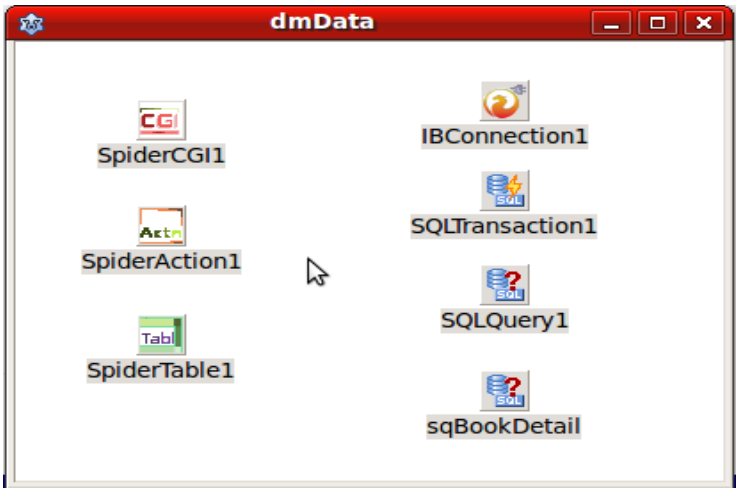
```
http://localhost/cgi-bin/myspider/firstaction?param1=value1&param2=value2
```

برنامج مستعرض الكتب

في هذا البرنامج سوف نقوم بإستعراض أسماء الكتب من قاعدة البيانات السابقة **Library** من الفصل السابق (قواعد البيانات العلائقية). وسوف نقوم إن شاء الله بعرض قائمة بأسماء الكتب بطريقة مشابهة لصفحة **list.htm** لكن هذه المرة الصفحة سوف تكون ديناميكية بدلا من كونها ثابتة، لأن أسماء الكتب غير محددة وهي موجودة في جدول **Books** في قاعدة البيانات.

نقوم بإنشاء برنامج جديد من نوع **FreeSpider Web Application** ونسميه **libweb**، وإسم الوحدة (حاوية البيانات) **main.pas**

نقوم بإدراج **TSpiderCGI**، ومكونات قواعد البيانات وإختيار **library.fdb** كما في الشكل التالي:



ثم نكتب الكود التالي في في المكون **SpiderCGI1**:

```
procedure TdmData.SpiderCGI1Request(Sender: TObject; Request: TSpiderRequest;
  var Response: TSpiderResponse);
begin
  Response.Add('<html><head><title>المكتبة</title>');
  Response.Add('<meta http-equiv="Content-Type" ' +
    'content="text/html; charset=utf-8" />');
  Response.Add('</head><body dir=rtl>');
  Response.Add('<h2>قائمة الكتب</h2>');
  SQLQuery1.SQL.Text:= 'select * from books';
  SQLQuery1.Open;
  with SQLQuery1 do
  while not EOF do
  begin
    Response.Add('<a href="/cgi-bin/libweb/viewbook?id=' +
      FieldByName('BookID').AsString + '">' + FieldByName('BookName').AsString +
      '</a><br/>');
    Next;
  end;
  SQLQuery1.Close;
  Response.Add('</body></html>');
end;
```

ثم نُدرج حزمة بيانات **TSQLQuery** ونسميها **sbBookDetail**. ونضع الكود التالي في خاصية **SQL**:

```
select * from Books
where BookID = :BookID
```

ثم نقوم بإدراج **TSpiderTable**. وفي الخاصية **Border** نضع القيمة 1، وفي الخاصية **DataSet** نختار **sqBookDetail**

ثم نقوم بإدراج مكون **TSpiderAction**. وفي الخاصية **Path** في هذا لمكون نضع القيمة التالي:

```
/viewbook
```

ونكتب الكود التالي في حدث **SpiderAction1**:

```
procedure TdmData.SpiderAction1Request(Sender: TObject;
  Request: TSpiderRequest; var Response: TSpiderResponse);
begin
  Response.Add('<html><head><title>المكتبة|</title>');
  Response.Add('<meta http-equiv="Content-Type" content="text/html; ' +
    'charset=utf-8" />');
  Response.Add('</head><body dir=rtl>');
  Response.Add('<h2>تفاصيل كتاب</h2>');

  sqBookDetail.Params.ParamByName('BookID').AsInteger:=
    StrToInt(Request.Query('id'));
  sqBookDetail.Open;
  Response.Add(SpiderTable1.Contents);
  sqBookDetail.Close;
  Response.Add('</body></html>');
end;
```

ثم نقوم بترجمة البرنامج ولاننسى تحويل مسار الترجمة إلى
/usr/lib/cgi-bin

عند تشغيل البرنامج يظهر لنا الشكل التالي في المتصفح، والذي هو عبارة عن نتيجة الحدث للمكون **SpiderCGI1**:

وعند الضغط على رابط إحدى الكتب تظهر لنا التفاصيل التالية، والتي هي عبارة عن نتيجة لتشغيل الحدث المربوط بـ **SpiderAction1**:

ORROWED	INFO	ENTRYDATE	COPYNUM	COPYDATE	COPIES	KEYWORDS	PUBLISHER	AUTHOR	BOOKNAME	BOOKID
0	كتاب أحكام التجويد	3-7-10 16:57:30	1	1-1-09	1	تجويد، القرآن	دار الإيمان - الإسكندرية	كريمة أوبزيد	أحكام التجويد	3

طريقة Get method

الطريقة التي إستخدمناها في الأمثلة السابقة لإرسال البيانات تُعرف بطريقة **GET**. وهي بإرسال المُدخلات مع العنوان **URL**، ثم قراءتها بواسطة

```
Request.Query('ParamName')
```

وهي طريقة سهلة الإستخدام، إلا أن بها بعض نواحي القصور، وهي أن طولها محدود، وهو حوالي 240 حرف، وسوف أتأكد من هذا الرقم لاحقاً إن شاء الله.

والمشكلة الثانية هي ظهور هذه المتغيرات في العنوان، فمثلاً لو كان أحد هذه المدخلات هي كلمة مرور فإنها سوف تظهر في عنوان المتصفح.

وهذه الطريقة مستخدمة بكثرة في حالة المدخلات المحدودة ، وتُستخدم أيضاً مع الروابط في برامج الويب.

إستخدام الفورم

يمكن إستخدام فورم الويب مع طريقة Get لإرسال البيانات بدلاً من إرسالها في العنوان. وذلك بإنشاء ملف HMTL به النص التالي:

```
<form method=GET action="/cgi-bin/myspider">
Your name <input type=text name="name"><br/>
Your address <input type=text name="address"><br/>
<input type=submit value="Send">
</form>
```

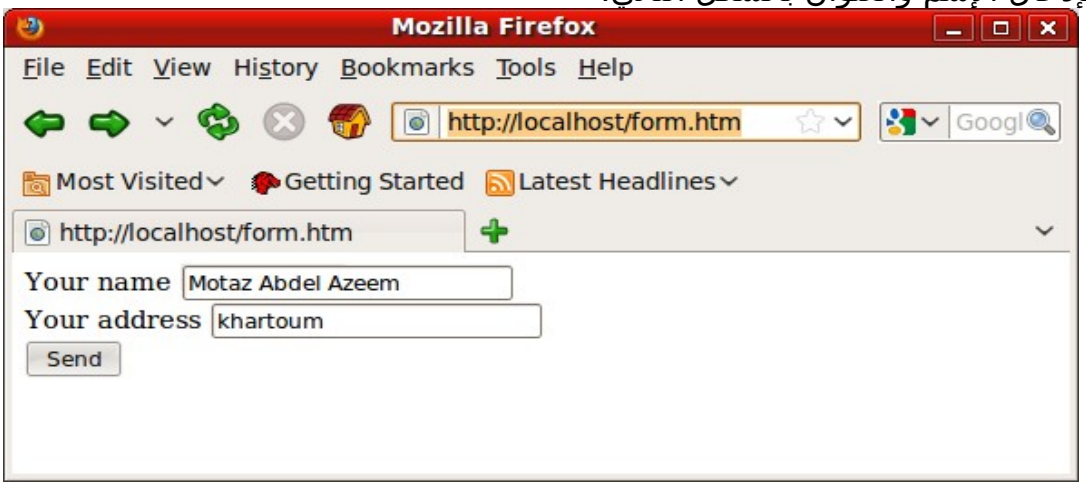
ونقوم بحفظه بإسم form.html في دليل الويب:

```
/var/www
```

ومن المتصفح نكتب العنوان :

```
http://localhost/form.htm
```

ثم نقوم بإدخال الإسم والعنوان بالشكل التالي:



وعند الضغط على الزر Send نحصل على النتيجة التالية:

Hello from **MySpider** web application
Time in server is: 11-9-10 11:59:41
Your name is: **Motaz Abdel Azeem** I live in : **khartoum**

طريقة Post method

هذه الطريقة تُستخدم بواسطة الفورم، حيث يجب توفر فورم حتي يمكننا إرسال بيانات إلى برنامج الويب. وهي غير محدودة بالنسبة لعدد المدخلات أو حجمها. حيث يمكننا إرسال مدخلات بسيطة، نصوص، ملفات ،صور، أو حتى برامج مضغوطة مثلاً عن طريق Post.

لإستخدام هذه الطريقة ماعلينا إلا تغير الصفحة السابقة Form.htm ، التغيير هو كلمة GET تُحولها إلى POST لتصبح الصفحة كالتالي:

```
<form method=POST action="/cgi-bin/myspider">
Your name <input type=text name="name"><br/>
Your address <input type=text name="address"><br/>
<input type=submit value="Send">
</form>
```

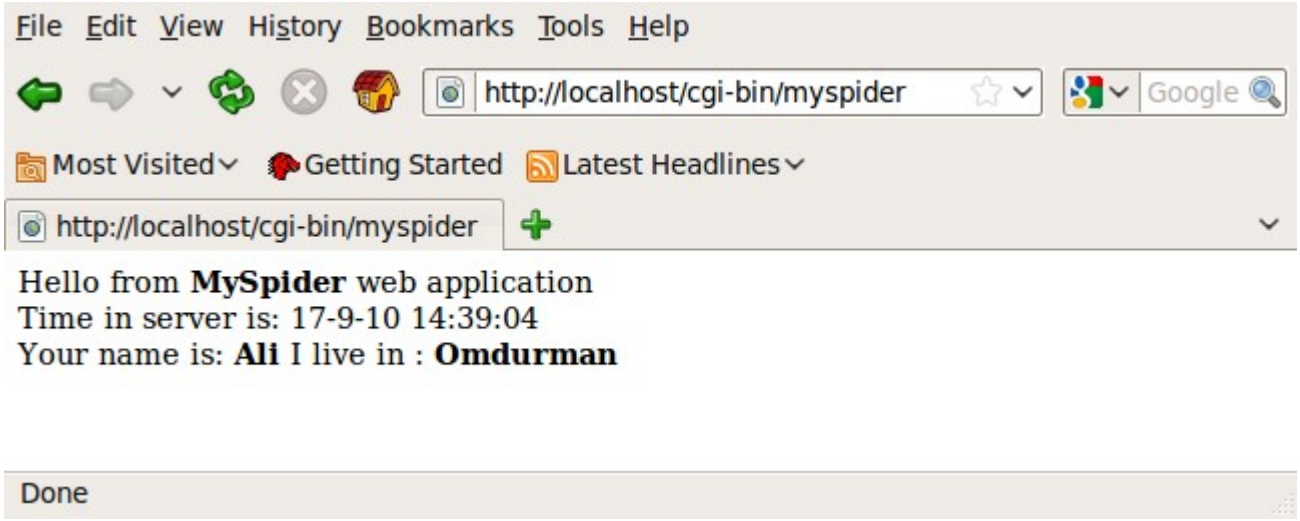
ثم نقوم بتغيير عبارات Request.Query إلى Request.Form في برنامج myspider لتصبح كالآتي:

```
procedure TDataModule1.SpiderCGI1Request(Sender: TObject;
  Request: TSpiderRequest; var Response: TSpiderResponse);
begin
  Response.Add('Hello from <b>MySpider</b> web application <br/>');
  Response.Add('Time in server is: ' + DateTimeToStr(Now));
  Response.Add('<br/>Your name is: <b>' + Request.Form('name') + '</b> ');
  Response.Add('I live in : <b>' + Request.Form('address') + '</b>');
end;
```

عند تنفيذ العنوان

http://localhost/form.htm

نلاحظ أن المتغيرات (المدخلات name, address) لاتظهر هذه المرة عند إرسال بيانات الفورم، أما في طريقة Get فكانت تظهر حتى بإستخدام الفورم:



فإذا قمنا بإرسال كلمة مرور مثلاً فإنها لاتظهر على العنوان.

فورم إسبايدر Spider Form

يمكننا تصميم فورم كما في المثال السابق عن طريق أي محرر للنصوص، أو عن طريق برامج تصميم الويب للحصول على فورم به حقول يتم إرسالها لبرنامج ويب.
كذلك يمكننا إنتاج فورم عن طريق المكون SpiderForm. في هذه الحالة يكون الفورم ديناميكي، حيث يمكن حذف حقول منه حسب حالة معينة.

لتجربة عمل فورم نقوم بإدراج TSpiderForm في برنامج MySpider، ونضع TspiderAction نسميه saForm وفي خاصية path نضع القيمة :

```
/form
```

وفي المكون SpiderForm1 نضع في الخاصية Action القيمة التالية:

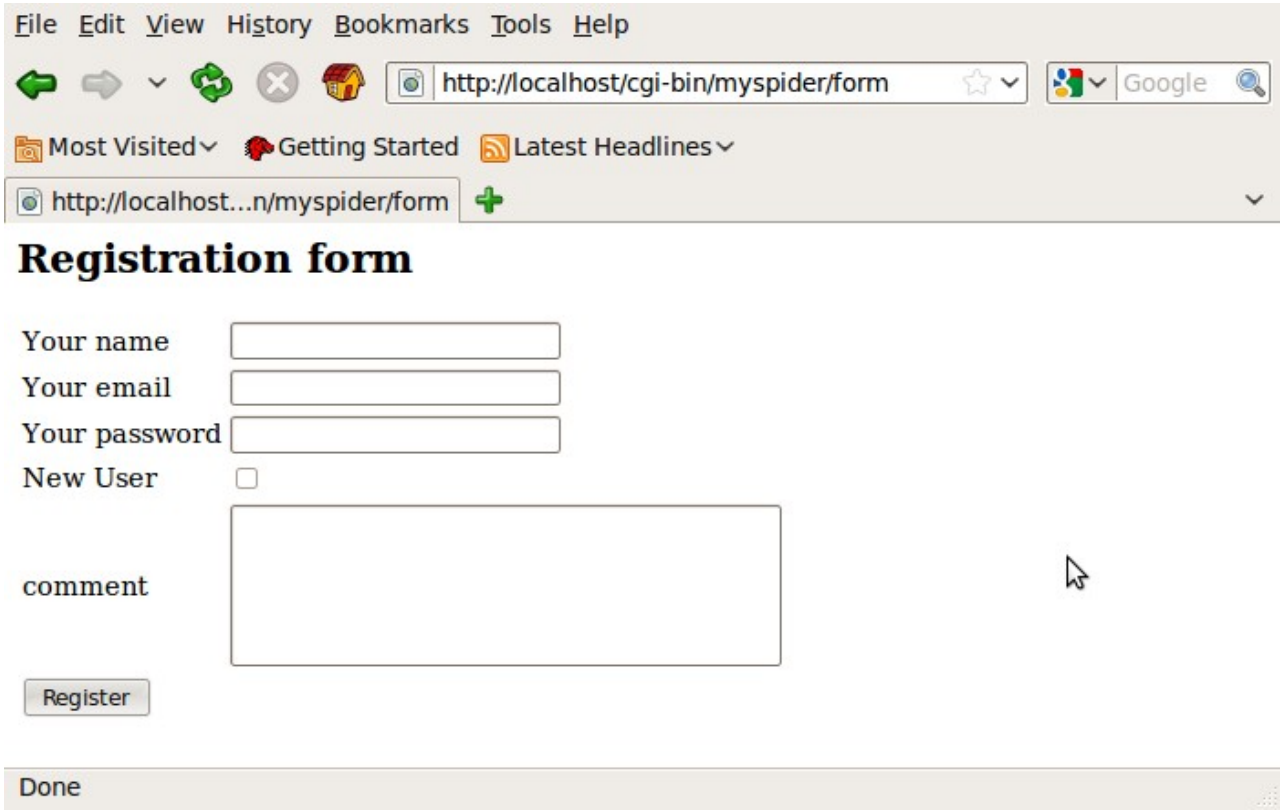
```
/cgi-bin/myspider/form
```

ثم نقوم بكتابة الكود التالي في الحدث OnCrequest للمكون saForm:

```
procedure TDataModule1.saFormRequest(Sender: TObject; Request: TSpiderRequest;
  var Response: TSpiderResponse);
begin
  Response.Add('<h2>Registration form</h2>');

  SpiderForm1.AddText('Your name');
  SpiderForm1.AddInput(itText, 'name');

  SpiderForm1.AddText('Your email');
  SpiderForm1.AddInput(itText, 'email');

  SpiderForm1.AddText('Your password');
  SpiderForm1.AddInput(itPassword, 'pass');

  SpiderForm1.AddText('New User');
  SpiderForm1.AddInput(itCheckbox, 'newuser');

  SpiderForm1.AddText('comment');
  SpiderForm1.AddInput(itTextArea, 'comment', '', 'rows=5 cols=40');

  SpiderForm1.AddInput(itSubmit, 'reg', 'Register');

  Response.Add(SpiderForm1.Contents);
end;
```

بعد الترجمة، يمكننا إستدعاء الفورم بالعنوان التالي:
http://localhost/cgi-bin/myspider/form

فيظهر لنا الفورم التالي:



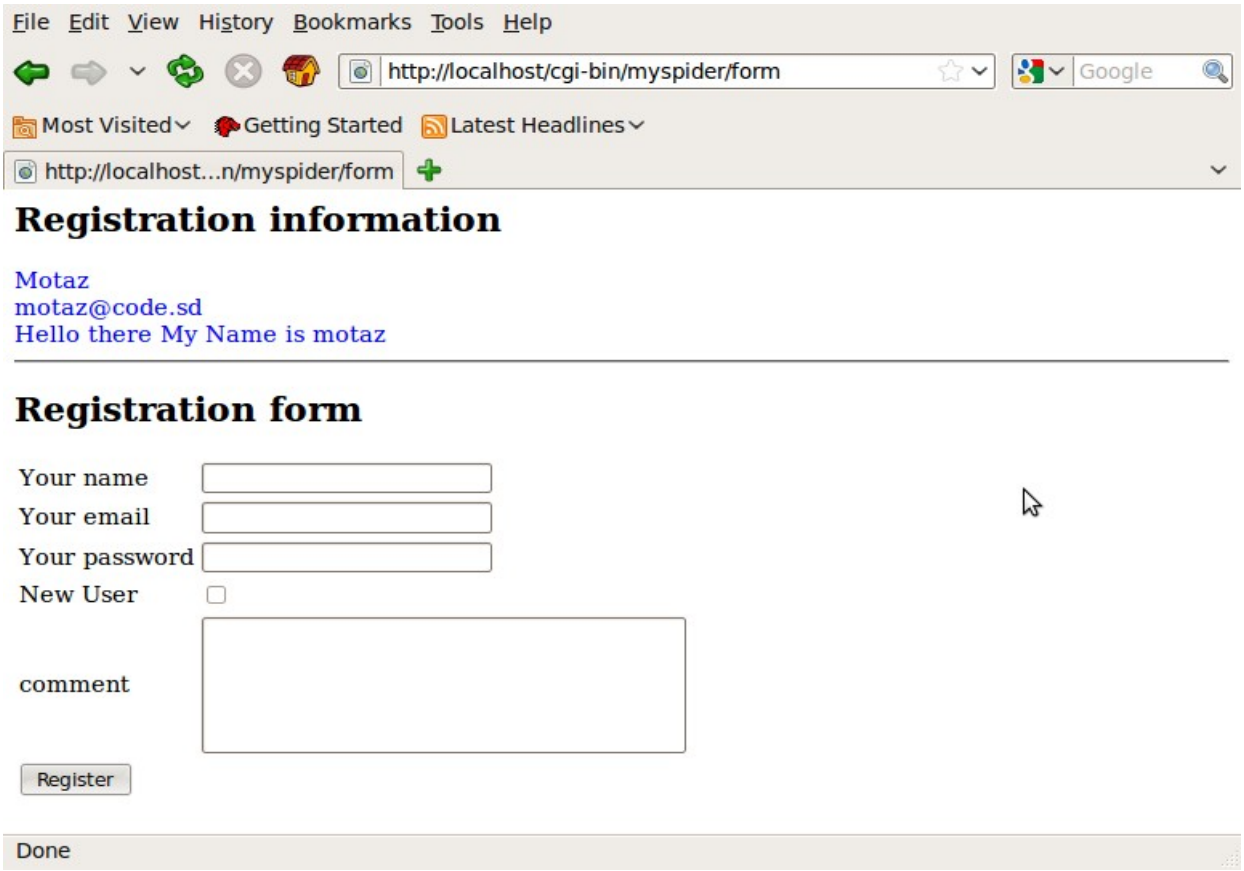
أما كود إستقبال البيانات فلم نقم بكتابته بعد.
وقد قمنا بتعديل الكود السابق لحدث المكون saForm لنضيف فيه إستقبال البيانات من الفورم ليصبح كالتالي:

```
procedure TDataModule1.saFormRequest(Sender: TObject; Request: TSpiderRequest;
  var Response: TSpiderResponse);
begin
  if Request.Form('reg') <> '' then // receive registration form
  begin
    Response.Add('<h2>Registration information</h2>');
    Response.Add('<font color=blue>');
    Response.Add(Request.Form('name') + '<br/>');
    Response.Add(Request.Form('email') + '<br/>');
    Response.Add(Request.Form('comment') + '<br/>');
    Response.Add('</font><hr>');
  end;

  Response.Add('<h2>Registration form</h2>');

  SpiderForm1.AddText('Your name');
  SpiderForm1.AddInput(itText, 'name');

  SpiderForm1.AddText('Your email');
  SpiderForm1.AddInput(itText, 'email');
```

```
  SpiderForm1.AddText('Your password');
  SpiderForm1.AddInput(itPassword, 'pass');

  SpiderForm1.AddText('New User');
  SpiderForm1.AddInput(itCheckbox, 'newuser');

  SpiderForm1.AddText('comment');
  SpiderForm1.AddInput(itTextArea, 'comment', '', 'rows=5 cols=40');

  SpiderForm1.AddInput(itSubmit, 'reg', 'Register');

  Response.Add(SpiderForm1.Contents);
end;
```

بعد ذلك يمكن للمبرمج تسجيل هذه البيانات في جدول قاعدة بيانات بعد التحقق من أن المستخدم قام بإدخال معلومات التسجيل بصورة صحيحة.

وهذا مثال لبيانات تم إرسالها:



دورة حياة برنامج الـ CGI

تمتاز برامج CGI بدورة حياة قصيرة جداً، حيث يتم إستدعاء البرنامج وقت وصول الطلب من مخدم الويب، عندها يتم تشغيل البرنامج وتحميله في الذاكرة، وعند تلبية الطلب وإرسال الرد Response إلى مخدم الويب الذي بدوره يقوم بإرسال هذا الرد إلى المتصفح فإن البرنامج يتم إغلاقه ويُمحى من الذاكرة، لذلك فإن هذه التقنية تعتبر stateless. وهي ميزة لها حسنات و بها عيوب.

أما حسناتها فهي أن حجز الموارد مثل الذاكرة والإتصال بقواعد البيانات والملفات تكون لفترة قصيرة. والميزة الثانية هي أنه لايحدث تسرب في الذاكرة memory leak مع طول الإستخدام. فإذا نسي المبرمج تحرير بعض المكونات، أو إغلاق جدول قاعدة بيانات، فإنها سوف يتم تحريرها وإغلاقها تلقائياً لأن البرنامج نفسه سوف يتم إغلاقه ويتلاشي من الذاكرة، وبالتالي فإن نظام التشغيل يقوم بتحرير ذاكرة الكومة Memory والمكدسة Stack التي تم حجزها بواسطة البرنامج المعني.

أما عيبها فهو عدم الربط بين طلبات المتسخدم. فلو أن المستخدم قام بالدخول على البرنامج بإستخدام إسم دخول، ثم قام بالضغط على رابط آخر، فإن هذه الطلبات تتم خدمتها بواسطة إستدعائين منفصيلن للبرنامج، وليس بينهما رابط، حيث لايمكن لبرنامج الويب الذي يخدم عدد كبير من الطلبات من أجهزة مختلفة التمييز بينها. وتتم علاج هذه المشكلة بإستخدام مايعرف بالكوكيز cookies.

الكوكيز Cookies

وهي عبارة عن بيانات يقوم بإرسالها برنامج الويب إلى المتصفح، والذي بدوره يقوم بحفظها لفترة معينة ويقوم بإرسالها تلقائياً مع كل طلب قادم لنفس برنامج الويب أو نفس مخدم الويب. وبهذه الطريقة يتحصل برنامج الويب على هذه البيانات في كل مرة من نفس المتصفح، وبذلك يتم الربط بين الطلبات والتي تعني بأنها تأتي من مستخدم واحد.

فإذا قام المستخدم بإدخال كلمة مرور مثلاً في المرة الأولى، فلايحتاج لإرسالها مرة أخرى مع الطلبات اللاحقة ونعتبر أنه قد قام بفتح جلسة مستمرة session.

نقوم بإرسال الكوكيز للمتصفح بإستخدام الإجراء:

```
Response.SetCookie
```

ونقوم بقراءتها منه بإستخدام الدالة:

```
Request.GetCookie
```

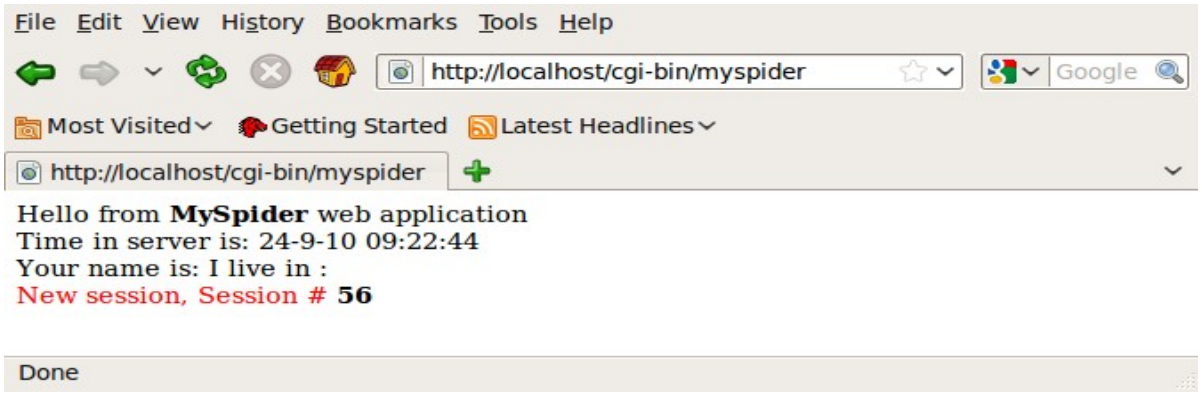
قمنا بتعديل البرنامج myspider بإضافة كود للحدث OnRequest للمكون SpiderCGI ليصبح:

```
procedure TDataModule1.SpiderCGI1Request(Sender: TObject;
  Request: TSpiderRequest; var Response: TSpiderResponse);
var
  SessionID: Integer;
begin
  Response.Add('Hello from <b>MySpider</b> web application <br/>');
  Response.Add('Time in server is: ' + DateTimeToStr(Now));
  Response.Add('<br/>Your name is: <b>' + Request.Form('name') + '</b> ');
  Response.Add('I live in : <b>' + Request.Form('address') + '</b><br/>');
```

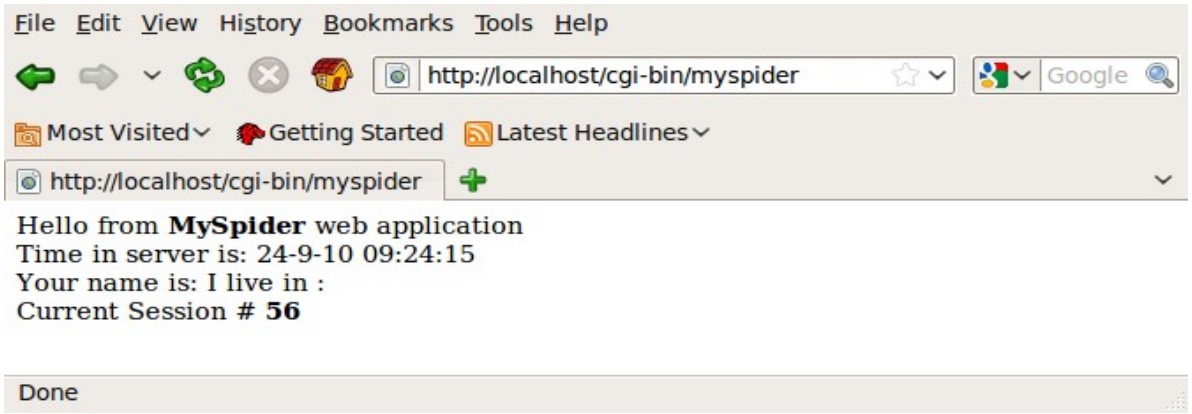
```
  // Cookies
  if Request.Cookies.IndexOfName('session') <> -1 then // Old session
    Response.Add('Current Session # <b>' + Request.GetCookie('session') + '</b>')
  else // New session
  begin
    Randomize;
    SessionID:= Random(1000);
    Response.SetCookie('session', IntToStr(SessionID), '');
    Response.Add('<font color=red>New session, Session # </font><b>' +
      IntToStr(SessionID) + '</b>');
  end;
end;
```

حيث يقوم هذا الكود بفحص المتغير session في الكوكيز الموجودة في المتصفح. فإذا لم تكون موجودة فهذا يعني أن هذا هو أول طلب للمستخدم، فيقوم بتحديد قيمة عشوائية لهذا المتغير وإرسالها للمتصفح، وفي المرات التالية من طلبات المستخدم (يمكن إستخدام F5 لتكرار نفس الطلب) يقوم البرنامج بإظهار هذه القيمة. ونلاحظ أنها ثابتة مادام المتصفح مفتوح. أما عند غلق المتصفح فإن هذه الكوكيز سوف تزول، ويتم إعطاء قيمة لجلسة جديدة.

عند فتح الرابط أول مرة نتحصل على:



وعند إعادة طلب الصفحة عن طريق F5 مثلاً نتحصل على:



وعند طلب الرابط من أجهزة مختلفة فإننا نتحصل على أرقام مختلفة للمتغير Session. لكن كل متصفح في جهاز منفصل يكون له نفس الرقم دائماً وذلك يعني أن برنامج الويب إستطاع التمييز بين طلبات الأجهزة المختلفة، وحافظ على جلسة كل مستخدم على حده.

فمثلاً إذا تم دخول ثلاث أجهزة على مخدم الويب على هذه الصفحة وقام البرنامج بإعطاء المتصفح في الجهاز الأول الرقم 512 والرقم 700 للجهاز الثاني و 210 للجهاز الثالث، فإن هذه الأرقام لاتختلط ببعضها بل يحافظ على متصفح على رقمه. وبهذه الطريقة يتعرف برنامج الويب على المستخدمين بإستخدام هذه الكوكيز.

برنامج الأخبار

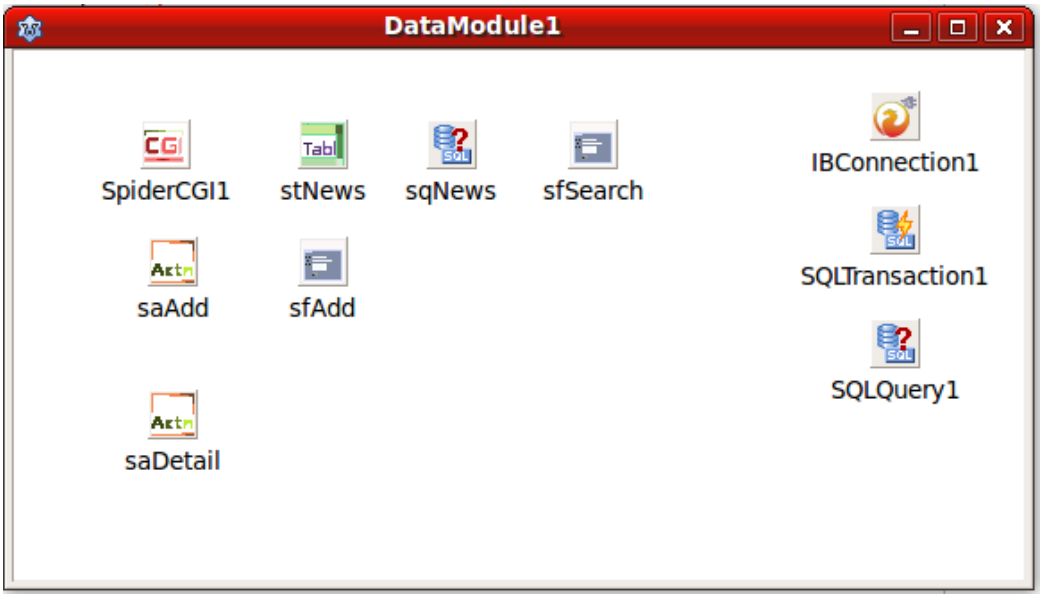
هذا البرنامج يُمكن المستخدمين من إدخال الأخبار وإستعراضها في شبكة محلية، أو الإنترنت.
لعمل هذا البرنامج يجب البداية بإنشاء قاعدة بيانات إسمها News تحتوي على جدول واحد هو News.
وهذه هي حقوله:

Management of : NEWS

Fields | Indices | Constraints | Triggers

P-Key	Field Name	Data Type	Size	Allow Null	Default Value
☑	ID	INTEGER	4	☐	
☐	TITLE	VARCHAR	50	☑	
☐	TEXT	BLOB	8	☑	
☐	USERNAME	VARCHAR	20	☑	
☐	NEWSTIME	TIMESTAMP	8	☑	
☐	CATEGORY	SMALLINT	2	☑	
☐	READERS	INTEGER	4	☑	
☐	USERADDRESS	VARCHAR	20	☑	

ثم نقوم بإنشاء برنامج فري إسبايدر جديد أسميناه كذلك News. وقمنا بإدراج المكونات التالية في حاوية بياناته DataModule1:



في الحدث OnCreate بالنسبة لحاوية البيانات نكتب الكود التالي:

```
procedure TDataModule1.DataModuleCreate(Sender: TObject);
begin
  SpiderCGI1.Execute;
end;
```

و في الحدث OnRequest بالنسبة للمكون SpiderCGI1 قمنا بكتابة الكود التالي:

```
procedure TDataModule1.SpiderCGI1Request(Sender: TObject;
  Request: TSpiderRequest; var Response: TSpiderResponse);
var
  SearchText: string;
begin
  SearchText:= Trim(Request.Form('search'));
  sfSearch.Action:= ExePath;
  sfSearch.AddText('Search');
  Response.Add('<html><header><title>News Center</title></header>');
  Response.Add('<body>');
  Response.Add('<h3>News center</h3>');
  Response.Add('<table width=100%><tr><td>');
  Response.Add('<a href="' + ExePath +'">Refresh</a></td><td>');

  // Search form
  sfSearch.AddInput(itText, 'search', SearchText, '', False);
  sfSearch.AddInput(itSubmit, '', 'Search', '', False);
  Response.Add(sfSearch.Contents);
  Response.Add('</td></tr></table>');
  Response.Add('<a href="' + ExePath + '/add">Add</a>');

  // News table
  if SearchText <> '' then  // Specify search criteria
    sqNews.SQL.Text:= 'select ID, NewsTime, Title, UserName, Readers from ' +
     'News where Lower(Title) like ''%' +
     LowerCase(SearchText) + '%'' or Lower(Text) like ''%' +
     LowerCase(SearchText) + '%'' order by ID desc'
   else // All data
     sqNews.SQL.Text:= 'select ID, NewsTime, Title, UserName, Readers from News '
      + 'order by ID desc';
  sqNews.Open;
  sqNews.FieldByName('NewsTime').DisplayLabel:= 'Time';
  sqNews.FieldByName('Title').DisplayLabel:= 'Title';
  sqNews.FieldByName('Readers').DisplayLabel:= 'Viewed';
  sqNews.FieldByName('UserName').DisplayLabel:= 'By';
  Response.Add(stNews.Contents);
  sqNews.Close;
  Response.Add('<hr>');
  Response.Add('<font color=gray>Written by Motaz Abdel Azeem');
  Response.Add('<a href="http://motaz.freevar.com">');
  Response.Add('motaz.freevar.com</a></font>');
  Response.Add('</body></html>');
end;
```

وفي الحدث OnRequest للمكون saAdd نكتب الكود التالي:

```
procedure TDataModule1.saAddRequest(Sender: TObject; Request: TSpiderRequest;
  var Response: TSpiderResponse);
var
  UserName: string;
begin
  if Request.Form('submit') = '' then  // View form
  begin
```

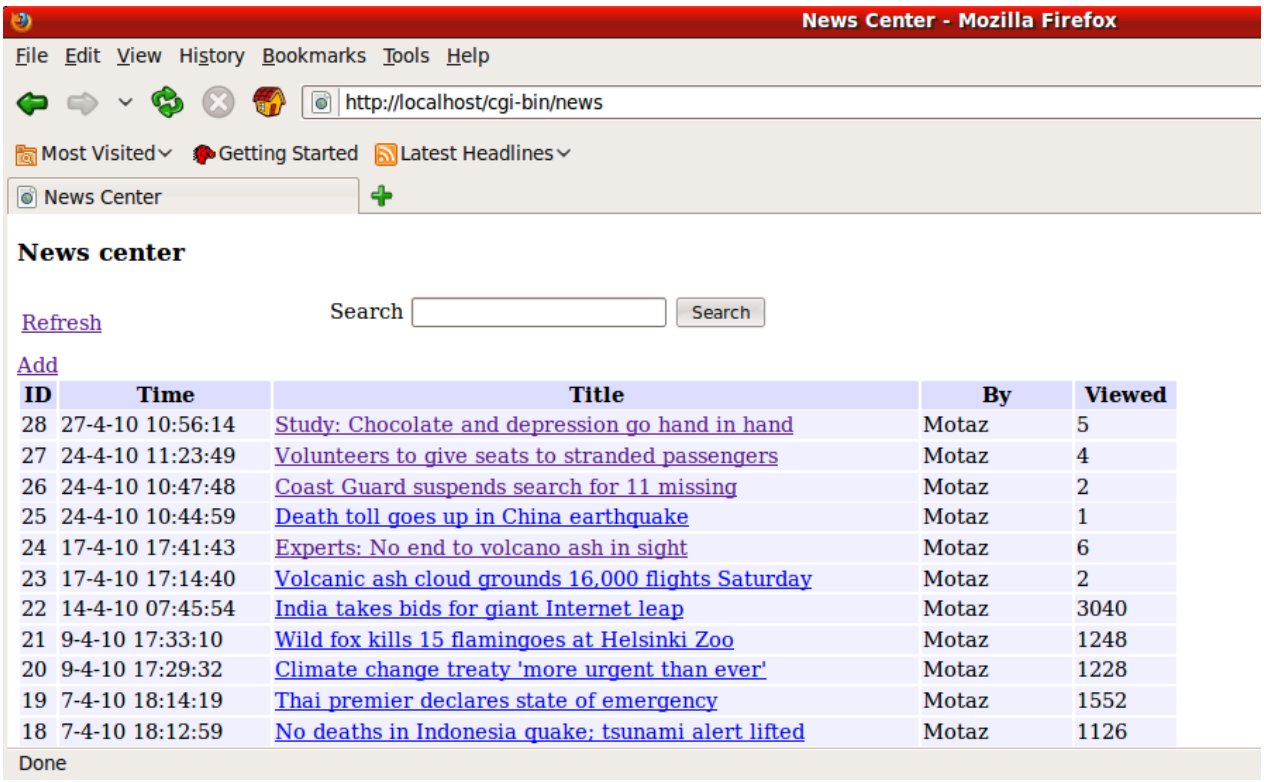
```
    sfAdd.Action:= ExePath + '/add';
    sfAdd.AddText('Title');
    sfAdd.AddInput(itText, 'title', '', 'size=50');

    UserName:= Request.GetCookie('username');
    sfAdd.AddText('User');
    sfAdd.AddInput(itText, 'user', UserName);

    sfAdd.AddText('Text');
    sfAdd.AddInput(itTextArea, 'text', '', 'rows=20, cols=60');

    sfAdd.AddInput(itSubmit, 'submit', 'Submit');

    Response.Add(sfAdd.Contents);
  end
  else // Check fields
  if (Trim(Request.Form('title')) = '') or (Trim(Request.Form('user')) = '') or
     (Trim(Request.Form('text')) = '') then
       Response.Add('You should fill all fields. Press browser''s back button' +
    ' to return to page')
  else
  begin  // Add new record
    SQLQuery1.SQL.Text:= 'insert into news (Title, Text, UserName, NewsTime,' +
      ' Category, Readers, UserAddress) ' +
      ' values (:Title, :Text, :UserName, CURRENT_TIMESTAMP, 0, 0, :UserAddress)';
    SQLQuery1.Params.ParamByName('Title').AsString:= Request.Form('title');
    SQLQuery1.Params.ParamByName('UserName').AsString:= Request.Form('user');
    SQLQuery1.Params.ParamByName('Text').AsString:= Request.Form('text');
    SQLQuery1.Params.ParamByName('UserAddress').AsString:= Request.RemoteAddress;
    SQLQuery1.ExecSQL;
    SQLTransaction1.Commit; // Save changes in database

    // Save username in cookies
    Response.SetCookie('username', Request.Form('user'), '/', Now + 10);

    Response.Add('A new record has been added<br/>');
    Response.Add('Click <a href="' + ExePath +
     '">here</a> to go to main news page');
  end;
end;
```

وفي الحدث OnRequest للمكون saDetail نكتب الكود التالي:

```
procedure TDataModule1.saDetailRequest(Sender: TObject;
  Request: TSpiderRequest; var Response: TSpiderResponse);
var
  NewsText: string;
  ID: Integer;
  RandomCheck: Boolean;
begin
  ID:= StrToInt(Request.Query('ID'));
  RandomCheck:= ID < 0;
  if RandomCheck then
  begin
    Randomize;
    ID:= Random(Abs(ID)) + 1;
  end;
```

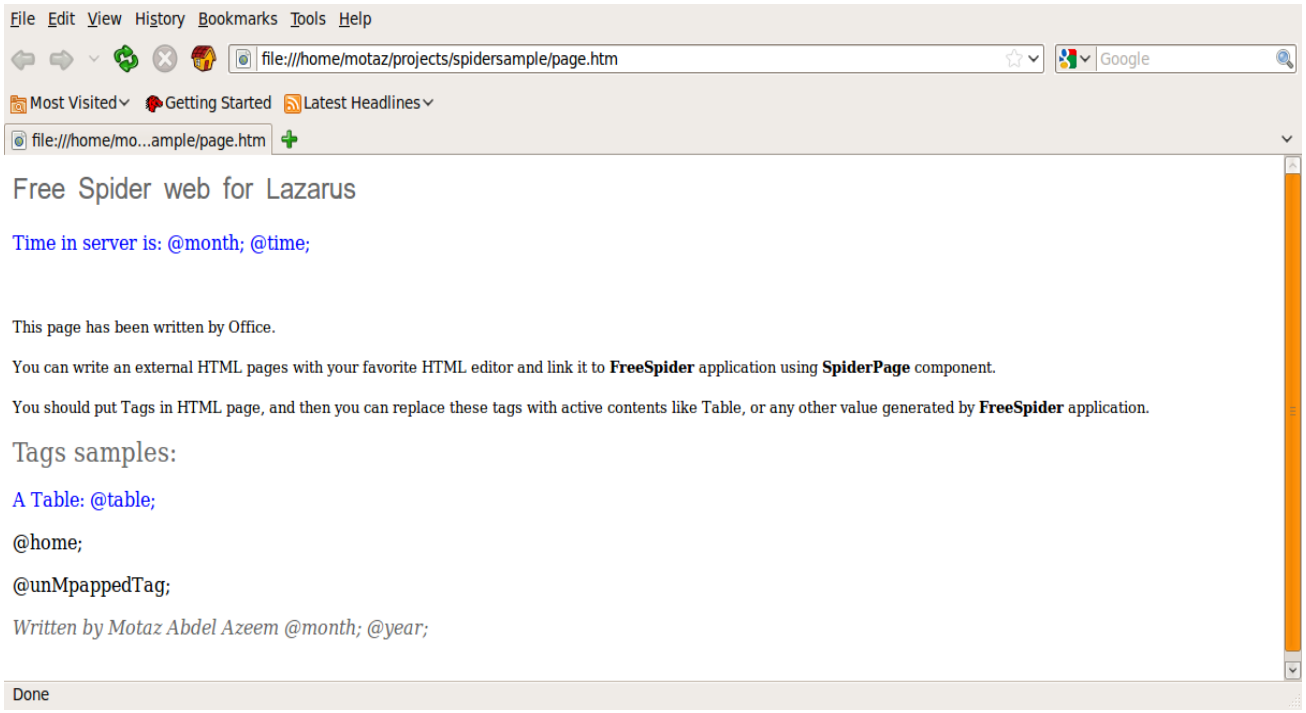
```
  SQLQuery1.SQL.Text:= 'select * from news where ID = :ID';
  SQLQuery1.Params.ParamByName('ID').AsInteger:= ID;
  SQLQuery1.Open;

  with SQLQuery1 do
  begin
    Response.Add('<h3>' + FieldByName('Title').AsString + '</h3>');
    Response.Add('Time: ' + FieldByName('NewsTime').AsString + '<br/><br/>');
    NewsText:= FieldByName('Text').AsString;
    NewsText:= StringReplace(NewsText, #10, '<br/>', [rfReplaceAll]);
    Response.Add(NewsText + '<br/><br/>');
    Response.Add('By: <b>' + FieldByName('UserName').AsString + '<br/><br/>');
  end;
  SQLQuery1.Close;
  Response.Add('<a href="' + ExePath + '">Main</a>');

  // Increase readers count
  if not RandomCheck then
  begin
    SQLQuery1.SQL.Text:= 'update News set Readers = Readers + 1 where ID = :ID';
    SQLQuery1.Params.ParamByName('ID').AsInteger:= ID;
    SQLQuery1.ExecSQL;
    SQLTransaction1.Commit;
  end;
end;
```

بعد ترجمة البرنامج وتشغيله ثم إدخال بعض الأخبار، تظهر لنا الشاشة الرئيسية التالية في المتصفح:



فصل التصميم من البرنامج

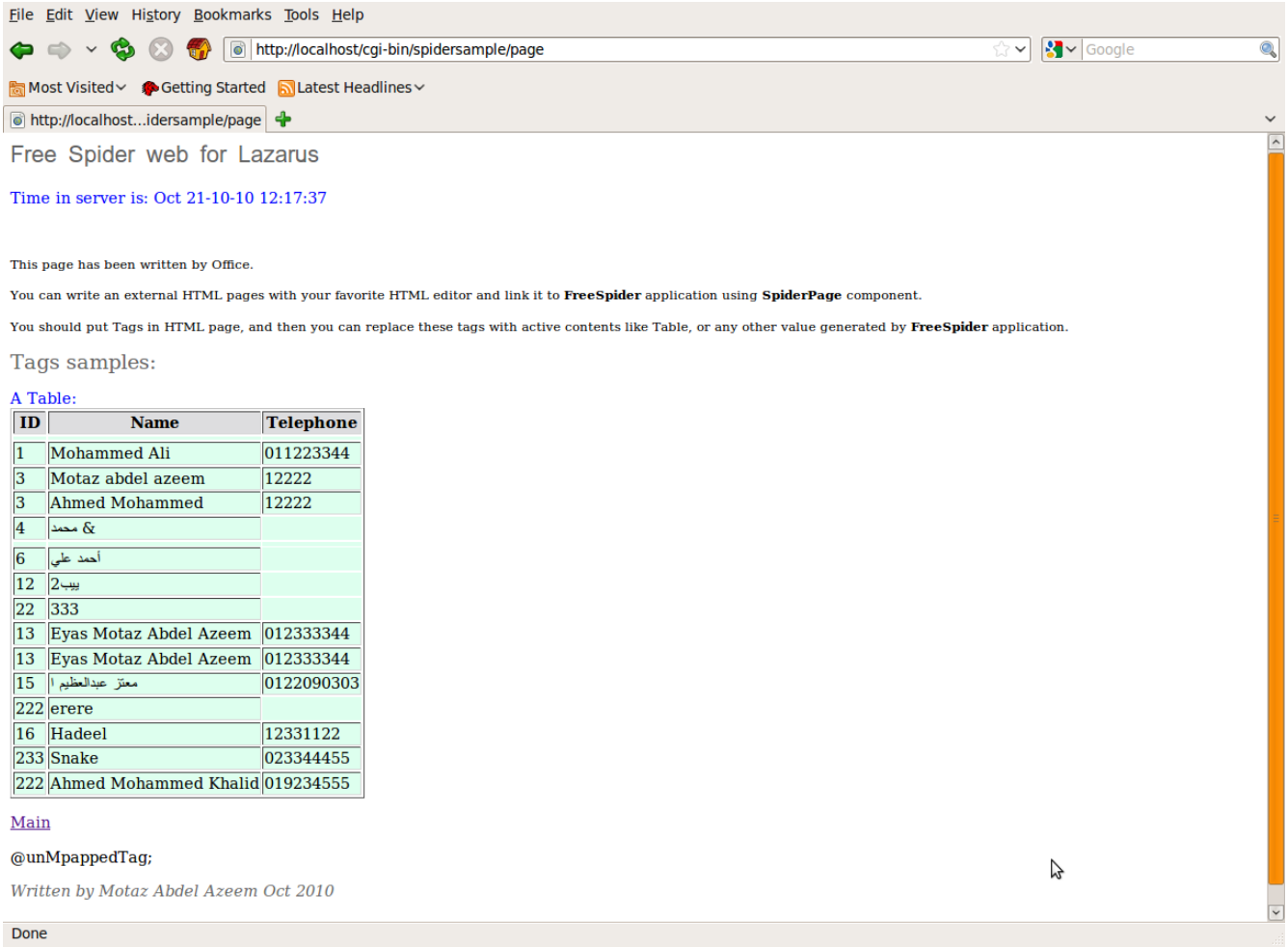
لاحظنا أن برامج الويب السابقة تم تصميم صفحاتها الديناميكية داخل كود الباسكال، فإذا إحتجنا أن نقوم بتغيير بعض محتويات الصفحات (HTML) فيجب تعديل كود البرنامج ثم إعادة ترجمته. لكن توجد طريقة يتم فيها فصل التصميم خارجياً في شكل ملفات .html عادية، ثم بإستخدام المكون TSpiderPage وبإستخدام الـ Tags فإنه يمكن جعل أجزاء من هذه الصفحات تكون ديناميكية ، فمثلاً يمكن أن يكون هناك صفحة ثابتة وداخلها جدول متغيير مثلاً :



نجد أن هذه الصفحة تم تصميمها بواسطة محرر HTML وبها بعض الـ Tags التي سوف يتم إستبدالها لاحقاً بمتغيرات عند تشغيل البرنامج. مثلاً

@month; @time; @home; @table;

عند تشغيل البرنامج يتم إستبدالها كالتالي:



والفكرة بسيطة، وهي ربط الملف page.htm بالمكون SpiderPage وعند الحدث OnTag نقوم بإستبدال هذه الـ Tags كالتالي:

```
procedure TDataModule1.SpiderPage1Tag(Sender: TObject; ATag: string;
  var TagReplacement: string);
begin
  if ATag = 'time' then
    TagReplacement:= DateTimeToStr(Now)
  else
  if ATag = 'table' then
  begin
    if FileExists('sub.dat') then
      MemDataset1.LoadFromFile('sub.dat');
    MemDataset1.Open;
    MemDataset1.FieldByName('SubName').DisplayLabel:= 'Name';
    MemDataset1.FieldByName('Address').Visible:= False;
    TagReplacement:= SpiderTable2.Contents;
  end
  else
  if ATag = 'home' then
    TagReplacement:= '<a href="./">Main</a>'
  else
  if ATag = 'month' then
    TagReplacement:= FormatDateTime('mmm', Now)
```

```
  else
  if ATag = 'year' then
    TagReplacement:= FormatDateTime('yyyy', Now);
end;
```

ويجب وضع الملف page.htm في الدليل cgi-bin حيث أن البرنامج هو من يستخدمه، ولايمكن الوصول إلى هذا الملف مباشرة عن طريق المتصفح بكتابة إسمه مباشرة، فهو لايمثل صفحة ثابته، ولو كان كذلك لوضعناه في دليل الصفحات الثابتة htdocs مثلاً.
بهذه الطريقة يمكننا إعادة تصميم الملف page.htm بدون الحاجة لفتح البرنامج وإعادة ترجمته، كذلك يمكن الإستعانة بمصمم ليس لديه خبرة في البرمجة لتصميم صفحات أو قوالب تتحول إلى صفحات ديناميكية في النهاية. بهذه الطريقة يكون تركيز المبرمج على الكود وقاعدة البيانات، أما المصمم فله الحرية في تعديل تصميم الصفحات وإضافة المحسنات مثل CSS.

الفصل الخامس

برمجة إتصالات الشبكات

Socket programming

مقدمة

برمجة إتصالات الشبكات Socket programming مقصود بها إمكانية إتصال برامج في أجهزة حاسوب مختلفة مع بعضها البعض في شبكة مبنية على برتكول TCP/IP وهو البرتوكول المُستخدم في الإنترنت وشبكات المؤسسات.
نجد أن هناك برامج كثيرة لاتُحصى تستخدم هذا الإسلوب في ربط البرامج والبيانات، فقواعد البيانات العلائقية تستخدم هذه الطريقة للإتصال بين المخدم والعميل client/server حيث يمكن أن يكون البرنامج الطرفي أو العميل في جهاز من أجهزة الشبكة، ومخدم قاعدة البيانات في جهاز آخر في نفس الشبكة أو شبكة بعيدة، على شرط إمكانية وصول جهاز العميل لجهاز المخدم بواسطة إسمه أوعنوانه الشبكي IP Address.
كذلك فإن من الأمثلة الشهيرة لإستخدام إتصالات الشبكات هو الإتصال بين متصفحات الإنترنت ومخدماتها، حيث أن المتصفحات لابد أن تكون متصلة في شبكة تُمكنها من الإتصال بمخدمات الإنترنت بواسطة عنوان المخدم أو رقمه الشبكي.

يعتمد الإتصال الشبكي في بروتوكول TCP/IP على شيئين رئيسيين هما:

1. **العنوان الشبكي** للمخدم والعميل IP Address: مثلاً يريد الجهاز 192.168.0.4 الإتصال بالمخدم 192.168.0.1 أو يمكن الإستعاضة عنها بأسماء الأجهزة في حالة وجود DNS مثلاً، user1computer و المخدم main_server

2. **رقم البورت** Port للبرنامج المخدم: حيث أن البرنامج المخدم server application لابد أن يُنصت للإتصالات من البرامج الطرفية في رقم بورت معين، مثلاً برنامج مخدم الويب apache يُنصت دائماً لطلبات المتصفحات في البورت رقم 80

ملحوظة:
يمكن أن يكون البرنامج العميل والمخدم يوجدان في نفس الجهاز، وفي هذه الحالة يمكننا إستخدام عنوان الشبكة أو إسم الجهاز للمخدم والعميل مثلاً إذا كان عنوان الجهاز هو 192.168.0.1 فإن المخدم يكون 192.168.0.1 والعميل يكون 192.168.0.1 . ويمكن كذلك إستخدام الإسم localhost للدلالة على إسم الجهاز الإفتراضي، أو العنوان الإفتراضي 127.0.0.1 الذي يعني نفس الجهاز.
كذلك يمكن أن يكون في جهاز واحد عدد كبير من البرامج المخدمة، لكن يجب أن ينفرد كل برنامج برقم بورت مختلف. مثلاً مخدم الويب يستخدم رقم البورت 80 ومخدم قاعدة بيانات FireBird يستخدم البورت 3050، فإذا حاول برنامج مخدم آخر إستخدام نفس البورت أرجع له نظام التشغيل خطأ يخبره بأن هذا البورت محجوز مثلاً "Port is in use"

حزمة LNet

لعمل برامج إتصال شبكي يُفضل إستخدام أحد مكونات TCP/IP الجاهزة، وفي الأمثلة التالية سوف نستخدم حزمة LNet التي بها مكونات للشبكات. وهذه الحزمة سهلة الإستخدام.

يتم تثبيت الحزمة بالطريقة العادية في بيئة لازاراس بعد الحصول على هذه الحزمة من الإنترنت، أو يمكن الحصول عليها مع الأمثلة المصاحبة للكتاب:

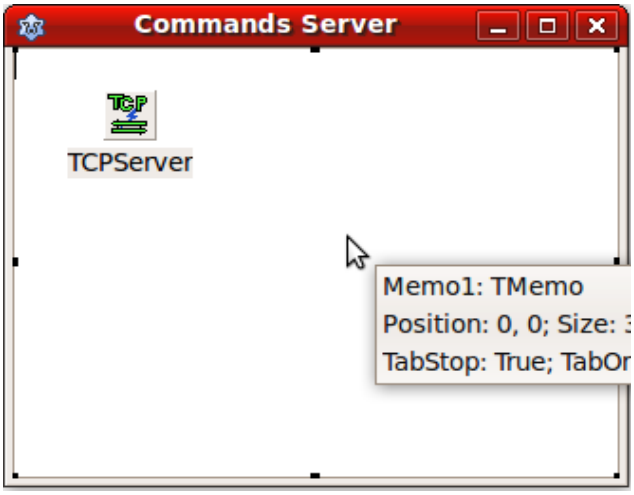
```
Package/Open Package File (*.lpk), select lnetvisual.lpk, then Install
```

برنامج الأوامر Commands

في هذا المثال سوف نقوم بعمل برنامجين، أحدهما مخدم أوامر Commands Server والآخر طرفية الأوامر Commands Client بحيث يقوم البرنامج الأول بالإستماع إلى البورت رقم 2010 في جهاز المخدم ليستقبل الإتصال من أي عدد من البرامج الطرفية.
والبرنامج الأخر Commands Client يقوم بالإتصال بالمخدم ويدع المستخدم ليقوم بإرسال أوامر، مثل time, date ليقوم المخدم بالرد عليها بإرسال التاريخ والوقت مثلاً إلى الجهاز الطرفي الذي قام بإرسال هذا الأمر. فإذا كان هناك عدد من البرامج الطرفية clients فإن المخدم يقوم بالرد على كل طلب لحده ولايحدث خلط في الأوامر أو النتائج.

البرنامج المُخدم CommandServer

نقوم بإنشاء برنامج جديد نسميه CommandServer ونضع في فورمه الرئيسي Memo و مكون الإتصالات TLTCPComponent من صفحة LNet في بيئة لازاراس ونسميها TCPServer ويكون الفورم بالشكل التالي:



ثم نقوم بجعل البرنامج يُنصت للإتصالات في البورت رقم 2010 وذلك عند بداية تشغيل البرنامج، بوضع هذا الكود في OnCreate بالنسبة للفورم الرئيسي:

```
procedure TfmMain.FormCreate(Sender: TObject);
begin
  TCPServer.Port:= 2010;
  if TCPServer.Listen then
    Memo1.Lines.Add('Listening on port: ' + IntToStr(TCPServer.Port));
end;
```

ثم قمنا بكتابة دالة تقوم بإستقبال الأوامر ثم إرجاع رد مناسب لهذه الأوامر:

```
function TfmMain.ProcessCommand(ACommand: string): string;
begin
  ACommand:= Trim(LowerCase(ACommand));
  if Pos('addnews:', ACommand) = 1 then
  begin
    Result:= 'Old news was: ' + News;
```

```
    News:= Copy(ACommand, Pos(':', ACommand) + 1, Length(ACommand));
  end
  else
  if ACommand = 'hello' then
    Result:= 'Hello there'
  else
  if ACommand = 'السلام عليكم' then
    Result:= 'وعليكم السلام'
  else
  if ACommand = 'time' then
    Result:= TimeToStr(Now)
  else
  if ACommand = 'date' then
    Result:= DateToStr(Now)
  else
  if ACommand = 'datetime' then
    Result:= DateTimeToStr(Now)
  else
  if ACommand = 'count' then
    Result:= IntToStr(TCPServer.Count - 1) + ' clients connected'
  else
  if ACommand = 'news' then
    Result:= 'News:' + News
  else
     Result:= 'Unknown command: ' + ACommand;

end;
```

نلاحظ أننا إستخدمنا متغير معرف في الفورم الرئيسي News نوعه مقطعي string وذلك لتخزين الخبر الذي قام أعد العملاء إرساله بهذه الصورة:

```
addnews:Ubuntu version 10.10 has been released on 10 Oct 2010
```

ويستطيع باقي العملاء قراءة هذا الخبر بإستخدام الأمر news

ثم قمنا بكتابة الكود التالي في الحدث OnReceive بالنسبة للمكون TCPServer:

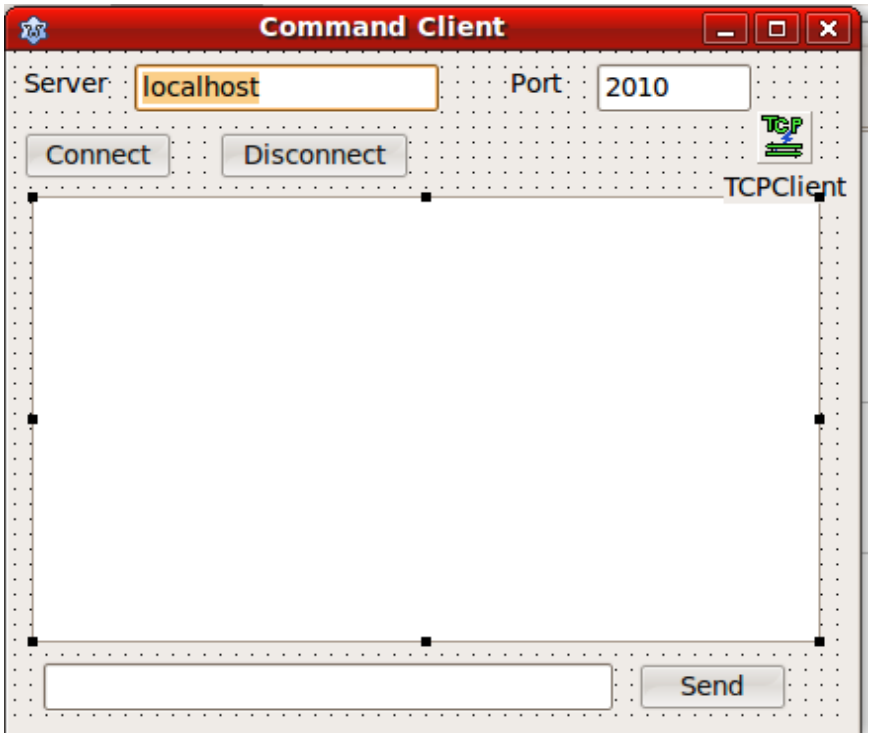
```
procedure TfmMain.TCPServerReceive(aSocket: TLSocket);
var
  Line: string;
  Res: string;
begin
  Memo1.Lines.Add('Receiving from: ' + aSocket.PeerAddress);
  aSocket.GetMessage(Line);
  Memo1.Lines.Add(Line);
  Res:= ProcessCommand(Line);
  aSocket.SendMessage(Res);
end;
```

نلاحظ أننا إستخدمنا الكائن aSocket والذي يمثل إتصال العميل الذي قام بإرسال نص (أمر) إلى المخدم. واستخدمنا الدالة GetMessage لإستقبال الرسالة النصية التي أتت من العميل، ثم الدالة SendMessage لإرسال الرد على الرسالة إلى العميل.

يمكن الرجوع للبرنامج كاملاً ضمن الأمثلة المصاحبة للكتاب.

البرنامج الطرفي CommandClient

نقوم بإنشاء برنامج جديد نسميه CommandClient نضع المكونات التالية في فورمه الرئيسي:

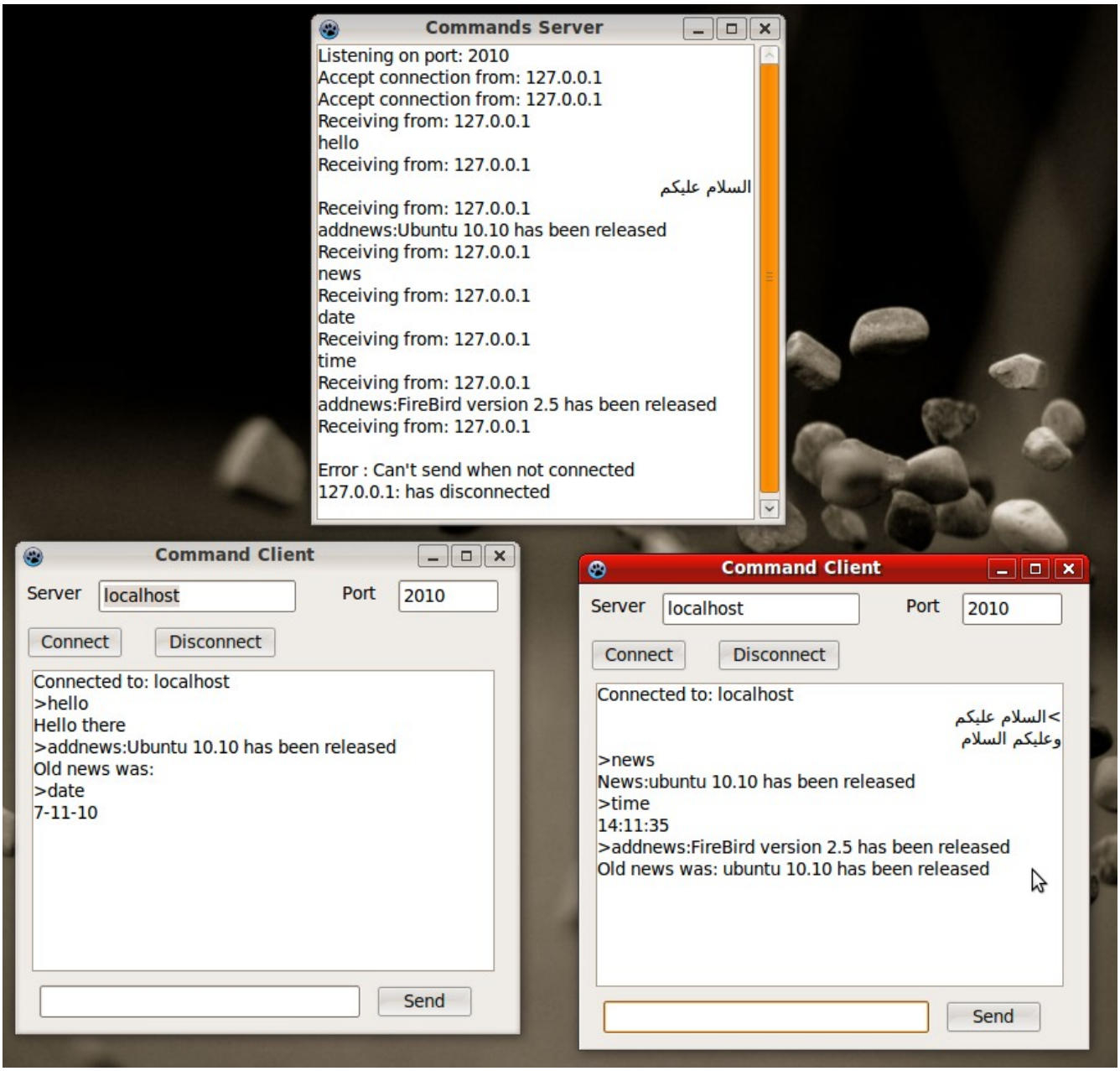


ثم نكتب الكود التالي في الوحدة المصاحبة لهذا الفورم:

```
unit main;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls,
  lNetComponents, lNet;

type

  { TfmMain }

  TfmMain = class(TForm)
    btConnect: TButton;
    btDisconnect: TButton;
    btSend: TButton;
    edCommand: TEdit;
    edPort: TEdit;
    edServer: TEdit;
    Label1: TLabel;
    Label2: TLabel;
```

```
    Memo1: TMemo;
    TCPClient: TLTCPComponent;
    procedure btConnectClick(Sender: TObject);
    procedure btDisconnectClick(Sender: TObject);
    procedure btSendClick(Sender: TObject);
    procedure TCPClientDisconnect(aSocket: TLSocket);
    procedure TCPClientReceive(aSocket: TLSocket);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  fmMain: TfmMain;

implementation

{$R *.lfm}

{ TfmMain }

procedure TfmMain.btConnectClick(Sender: TObject);
begin
  TCPClient.Port:= StrToInt(edPort.Text);
  TCPClient.Host:= edServer.Text;
  if TCPClient.Connect then
    Memo1.Lines.Add('Connected to: ' + edServer.Text);
end;

procedure TfmMain.btDisconnectClick(Sender: TObject);
begin
  TCPClient.Disconnect;
end;

procedure TfmMain.btSendClick(Sender: TObject);
begin
  TCPClient.SendMessage(edCommand.Text);
  Memo1.Lines.Add('>' + edCommand.Text);
  edCommand.Clear;
end;

procedure TfmMain.TCPClientDisconnect(aSocket: TLSocket);
begin
  Memo1.Lines.Add('Disconnected from server');
end;

procedure TfmMain.TCPClientReceive(aSocket: TLSocket);
var
  Res: string;
begin
  aSocket.GetMessage(Res);
  Memo1.Lines.Add(Res);
end;

end.
```

وعند تشغيل نسخة المخدم ونسختين من برنامج العميل والضغط على الزر Connect في كلا البرامج الطرفية ثم نقوم بإرسال أوامر فنحصل على المثال التالي:



برنامج المحادثة

إحدى تطبيقات برتوكول TCP/IP هي برامج المحادثة Chat وهي إتصال شخصين أو أكثر من أجهزة كمبيوتر مختلفة والتحدث عن طريق رسائل نصية. ثم تطورت هذه البرامج وتمت عليها عدة إضافات مثل التحدث بالصوت ونقل الملفات أثناء المحادثة. ومن أشهر الأمثلة في هذا المجال برنامج Skype والذي تمت كتابته بواسطة أداة التطوير دلفي.
في هذا المثال سوف نقوم بعمل برنامج بسيط يُعتبر مخدم وطرفية في نفس الوقت، حيث سوف نستخدم فيه مكوني LNet TCPComponent أحدهما يستقبل الإتصال ليعمل كمخدم والأخر يقوم بالإتصال بالبرنامج الآخر ويعتبر كطرفية. وهذه الطريقة في الإتصال (أن يكون البرنامج مخدم وطرفية) تسمى طريقة إتصال Peer to Peer، لأن كلا البرنامجين متشابهان.

قمنا بإنشاء برنامج جديد وأسمينا Chat، ثم وضعنا فيه المكونات التالية:

ومن أهم المكونات التي إستخدمناها هي Server, Client وهي من النوع TLTCPComponent.
وعند تشغيل البرنامج يقوم المكون Server بأخذ وضع الإستماع Server.Listen في رقم بورت معين، في هذه الحال 2011 ويمكن تغييره.
كود الوحدة المصاحبة لهذا الفورم هي:

```
unit main;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls,
```

```
  ExtCtrls, lNetComponents, lNet, IniFiles;

type

  { TfmMain }

  TfmMain = class(TForm)
    btConnect: TButton;
    btDisconnect: TButton;
    btSend: TButton;
    Button1: TButton;
    edConnectTo: TEdit;
    edMessage: TEdit;
    edPort: TEdit;
    edServerPort: TEdit;
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Client: TLTCPComponent;
    Label3: TLabel;
    Server: TLTCPComponent;
    Memo1: TMemo;
    Panel1: TPanel;
    Panel2: TPanel;
    procedure btConnectClick(Sender: TObject);
    procedure btDisconnectClick(Sender: TObject);
    procedure btSendClick(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure ClientConnect(aSocket: TLSocket);
    procedure ClientError(const msg: string; aSocket: TLSocket);
    procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
    procedure FormCreate(Sender: TObject);
    procedure ServerAccept(aSocket: TLSocket);
    procedure ServerDisconnect(aSocket: TLSocket);
    procedure ServerError(const msg: string; aSocket: TLSocket);
    procedure ServerReceive(aSocket: TLSocket);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  fmMain: TfmMain;

implementation

{$R *.lfm}

{ TfmMain }

procedure TfmMain.ServerAccept(aSocket: TLSocket);
begin
  Memo1.Lines.Add('إستقبال الإتصال من : ' + aSocket.PeerAddress);
  if not Client.Connected then
    btConnectClick(nil);
end;
```

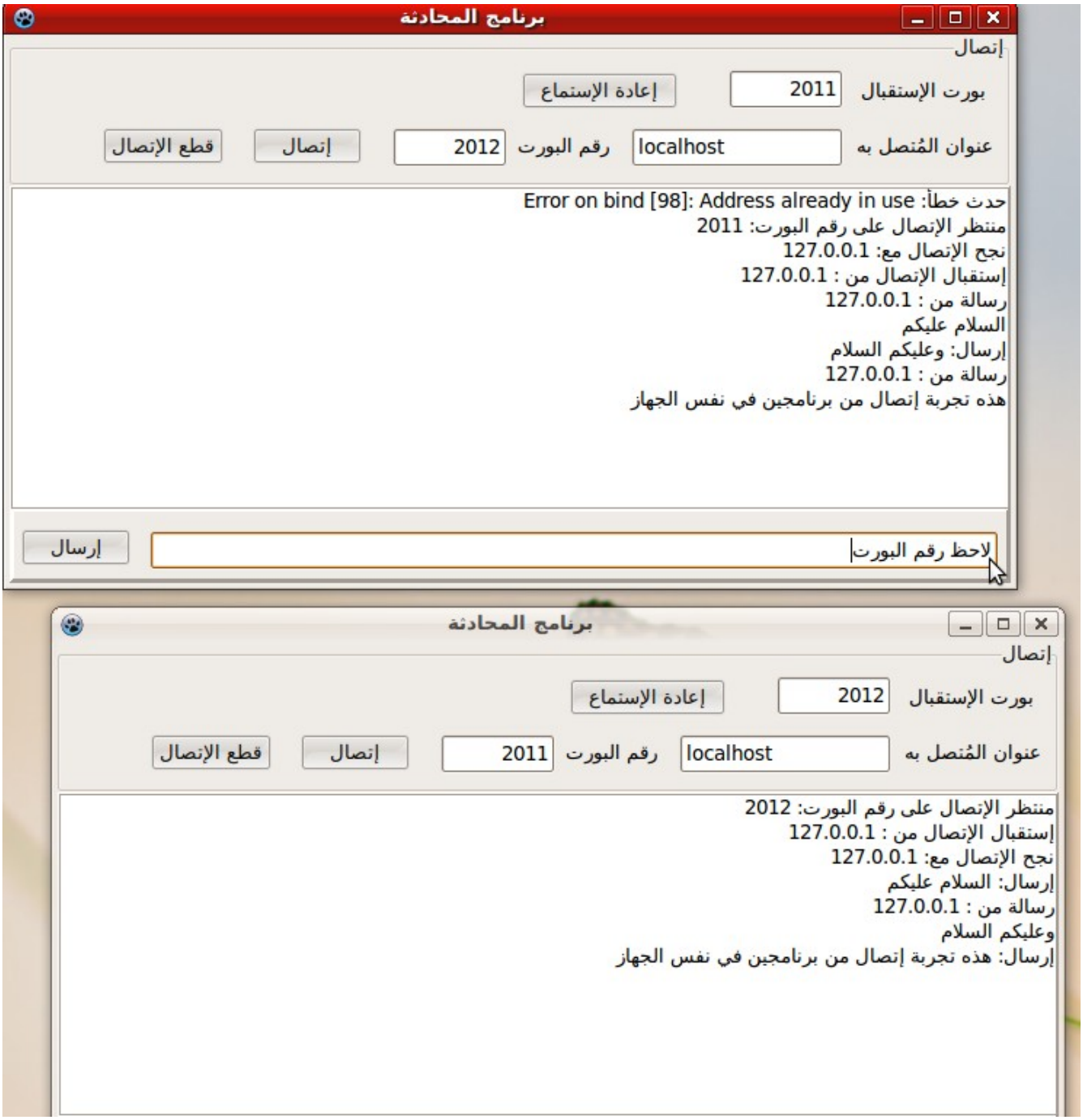
```
procedure TfmMain.btConnectClick(Sender: TObject);
begin
  if Client.Connected then
    Client.Disconnect;
  Client.Connect(edConnectTo.Text, StrToInt(Trim(edPort.Text)));
end;

procedure TfmMain.btDisconnectClick(Sender: TObject);
begin
  Client.Disconnect;
end;

procedure TfmMain.btSendClick(Sender: TObject);
begin
  if (Client.Connected) and (edMessage.Text <> '') then
  begin
    Client.SendMessage(edMessage.Text);
    Memo1.Lines.Add('إرسال: '  + edMessage.Text);
    Memo1.SelStart:= Length(Memo1.Text);
    edMessage.Clear;
  end;
end;

procedure TfmMain.Button1Click(Sender: TObject);
begin
  Server.Disconnect;
  Server.Port:= StrToInt(Trim(edServerPort.Text));
  if Server.Listen then
    Memo1.Lines.Add('منتظر الإتصال على رقم البورت: ' + IntToStr(Server.Port));
end;

procedure TfmMain.ClientConnect(aSocket: TLSocket);
begin
  Memo1.Lines.Add('نجح الإتصال مع: ' + aSocket.PeerAddress);
end;

procedure TfmMain.ClientError(const msg: string; aSocket: TLSocket);
begin
  Memo1.Lines.Add('حدث خطأ: '  + msg);
end;

procedure TfmMain.FormClose(Sender: TObject; var CloseAction: TCloseAction);
var
  Ini: TIniFile;
begin
  if Server.Connected then
    Server.Disconnect;
  if Client.Connected then
    Client.Disconnect;

  Ini:= TIniFile.Create(ChangeFileExt(ParamStr(0), '.ini'));
  Ini.WriteString('Settings', 'ServerPort', edServerPort.Text);

  Ini.WriteString('Settings', 'ConnectTo', edConnectTo.Text);
  Ini.WriteString('Settings', 'RemotePort', edPort.Text);
  Ini.Free;
```

```
end;

procedure TfmMain.FormCreate(Sender: TObject);
var
  Ini: TIniFile;
begin
  Ini:= TIniFile.Create(ChangeFileExt(ParamStr(0), '.ini'));
  Server.Port:= Ini.ReadInteger('Settings', 'ServerPort', 2011);
  edServerPort.Text:= IntToStr(Server.Port);
  edConnectTo.Text:= Ini.ReadString('Settings', 'ConnectTo', '');
  edPort.Text:= Ini.ReadString('Settings', 'RemotePort', '2011');
  Ini.Free;

  // Listen
  if Server.Listen then
    Memo1.Lines.Add('منتظر الإتصال على رقم البورت: ' + IntToStr(Server.Port));

end;

procedure TfmMain.ServerDisconnect(aSocket: TLSocket);
begin
  Memo1.Lines.Add('إنقطع الإتصال');
end;

procedure TfmMain.ServerError(const msg: string; aSocket: TLSocket);
begin
  Memo1.Lines.Add('حدث خطأ: ' + msg);
end;

procedure TfmMain.ServerReceive(aSocket: TLSocket);
var
  Msg: string;
begin
  Memo1.Lines.Add('رسالة من : ' + aSocket.PeerAddress);
  aSocket.GetMessage(Msg);
  Memo1.Lines.Add(Msg);
  Memo1.SelStart:= Length(Memo1.Text) - 1;
end;

end.
```

نلاحظ أن الإعدادات (أرقام البورت وعنوان الجهاز المراد الإتصال به) يتم تسجيلها في ملف .ini وذلك عند إغلاق البرنامج، وعند تشغيل البرنامج تتم قراءة تلك القيم.

يجب تشغيل البرنامج في جهازين في الشبكة، وكل مستخدم يجب أن يكتب إسم أو رقم عنوان الجهاز الآخر المراد الإتصال به، كذلك يجب التأكد من أن رقم البورت يتم الإستماع إليه في الجهاز الآخر قبل الإتصال.
يمكن كذلك تشغيل البرنامجين في نفس الجهاز لغرض التجربة، لكن هذه المرة بما أنهما يُمثلان نفس الجهاز، فيجب عندئذٍ تغيير رقم البورت، حيث لايمكن لبرنامجين يعملان في نفس الجهاز أن يقوما بحجز نفس رقم البورت والإستماع عنده. ويحدث في هذه الحالة الخطأ: Address is in use

هذا مثال لتشغيل نسختين من البرنامج (يوجدان في مجلدين منفصلين):



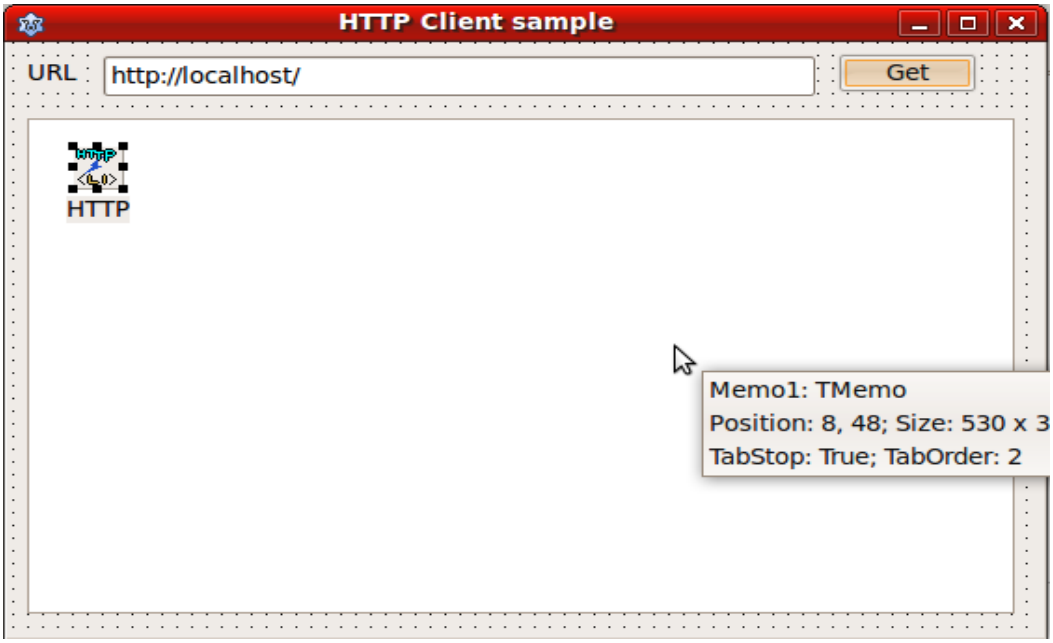
يمكن تطوير البرنامج بإضافة ميزات مثل نقل الملفات، او إضافة المحادثة الصوتية، ففي النهاية الملفات أو الصوت هي عبارة عن بايتات يتم إرسالها من برنامج وإستقبالها في البرنامج الآخر.

بروتوكول HTTP

وهو برتوكول مشهور جداً حيث يُستخدم بين متصفحات الإنترنت ومخدمات الويب. وهو بروتوكول يستخدم الـ TCP/IP كناقل له، ويستطيع المتصفح ومخدم الويب تبادل المعلومات بواسطته ، مثلاً عندما يطلب المتصفح صفحة ما، يقوم بإرسالها للمخدم عن طريق طلب HTTP، فيقوم المخدم بالرد عليه بإرسال (Download) محتويات هذه الصفحة. ويمكن أن تكون المحتويات صفحة أو ملف، مثلاً صفحة تحتوي على نص HTML يستطيع المتصفح عرضها بالشكل المعروف، أو نص بسيط Text أو صورة، أو ملف مضغوط، أو حتى طلب لتنفيذ إجراء في برنامج ويب. هذه الأشياء مرت علينا في فصل برامج الويب. وكان البرنامج الطرفي هو متصفح الإنترنت، أما في هذه المرة سوف نستبدل متصفح الإنترنت ببرنامج طرفي نقوم بصنعه بواسطة لازاراس وإستخدام المكون TLHTTPClientComponent الموجود في صفحة LNet.

برنامج طرفية HTTP

لعمل هذا المثال نقوم بإنشاء برنامج جديد نسميه HTTPClient ثم نقوم بإنزال مكون TLHTTPClientComponent ونسميه HTTP وبعض المكونات الأخرى بالشكل التالي في الفورم الرئيسي للبرنامج:



ثم نقوم بإضافة الوحدة lHTTPUtil ضمن الوحدات المستخدمة في الفورم الرئيسي. وهذه الوحدة تحتوي على الدالة DecomposeURL التي تقوم بتفسير عنوان صفحة الإنترنت إلى إسم المخدم، ورقم البورت والطلب.

وفي الزر Get نكتب الكود التالي:

```
procedure TfmMain.btGetClick(Sender: TObject);
var
  Host, URI: string;
```

```
  Port: Word;
begin
  DecomposeURL(edURL.Text, Host, URI, Port);
  if URI = '' then
    URI:= '/';
  HTTP.Host:= Host;
  HTTP.URI:= URI;
  HTTP.Port:= Port;
  Memo1.Clear;
  edURL.Font.Color:= clBlue;
  HTTP.SendRequest;
end;
```

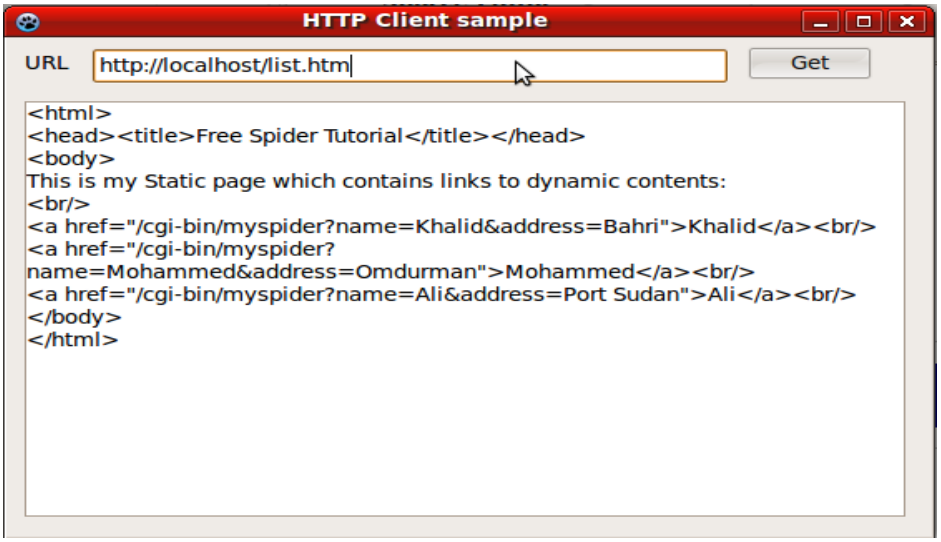
بهذا نكون قد أرسنا الطلب إلى المخدم. وسوف تأتي النتيجة لاحقاً على فترات في الحدث OnInput للمكون HTTP:

```
function TfmMain.HTTPInput(ASocket: TLHTTPClientSocket; ABuffer: pchar;
  ASize: integer): integer;
var
  Line: string;
  i: Integer;
begin
  for i:= 0 to ASize - 1 do
    if ABuffer[i] <> #0 then
    Line:= Line + ABuffer[i];
  Memo1.Text:= Memo1.Text + Line;
  Result:= ASize; // All received data has been read
end;
```

وعند إكتمال إستلام الرد من المخدم يتم إستدعاء الحدث OnDoneInput :

```
procedure TfmMain.HTTPDoneInput(ASocket: TLHTTPClientSocket);
begin
  edURL.Font.Color:= clBlack;
end;
```

عند تشغيل البرنامج يمكننا كتابة أي عنوان، فيقوم البرنامج بإستقبال الصفحة لكن في شكل كود لغة HTML:



برتوكول FTP

يعتبر برتوكول FTP من التطبيقات القديمة لإستخدام TCP/IP. ويُستخدم في نقل الملفات من وإلى مخدم FTP. ولتجربته، علينا أولاً تثبيت مخدم FTP. مثلاً في اوبونتو لينكس نقوم بكتابة الأمر التالي:

```
sudo apt-get install vsftpd
```

بعدها يمكن تجربته بواسطة برنامج سطر الأوامر:

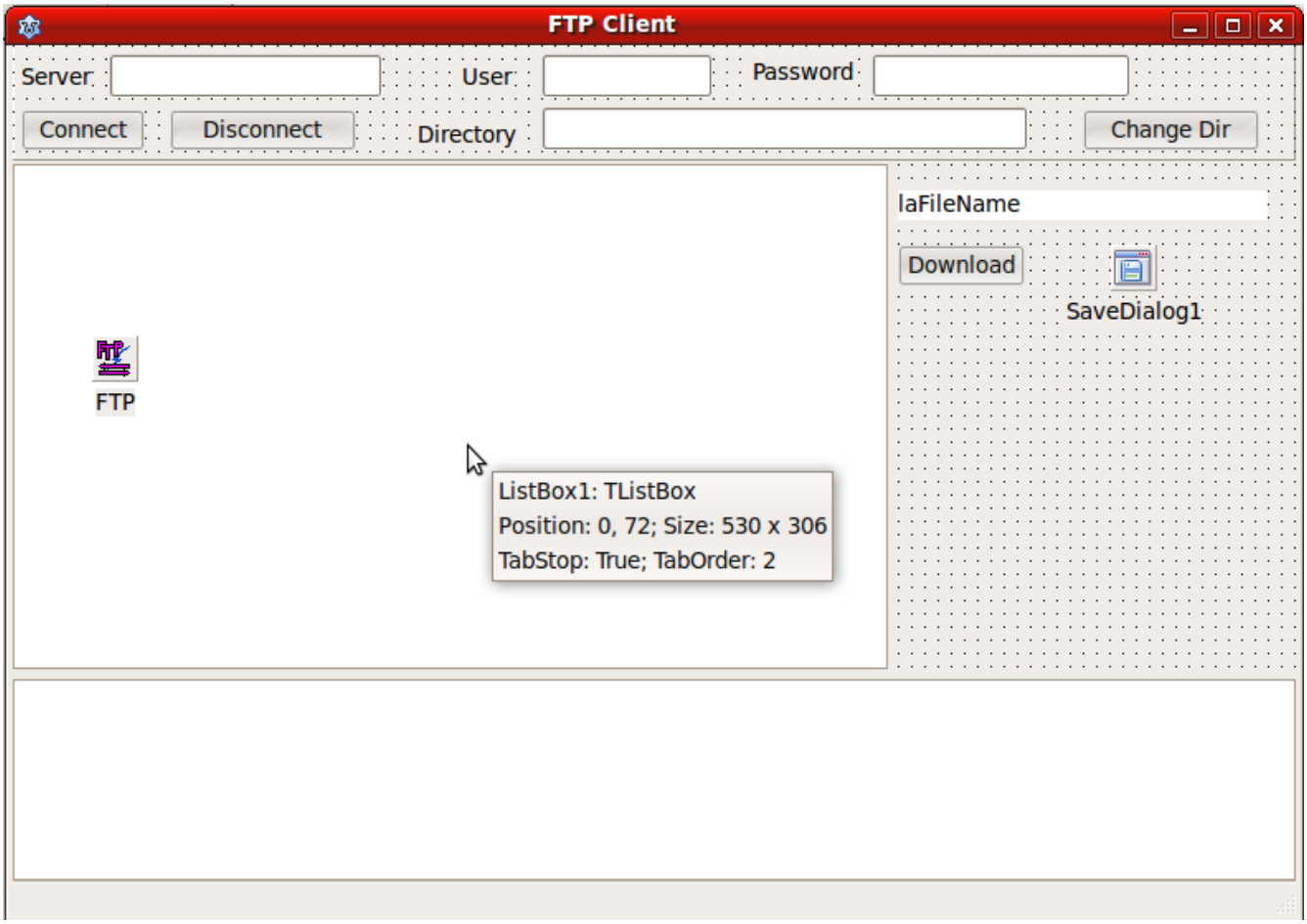
```
ftp localhost
```

والذي يقوم بالإتصال بالمخدم برقم البورت المستخدم للـ FTP وهو 21

برنامج FTP CLient

لكتابة برنامج طرفي (عميل) لمخدم FTP نقوم بإستخدام المكون TLFTPClientComponent الموجود ضمن حزمة LNet

قمنا بإنشاء برنامج بالشكل التالي:



وكتبنا الكود التالي في الوحدة المصاحبة لهذا الفورم:

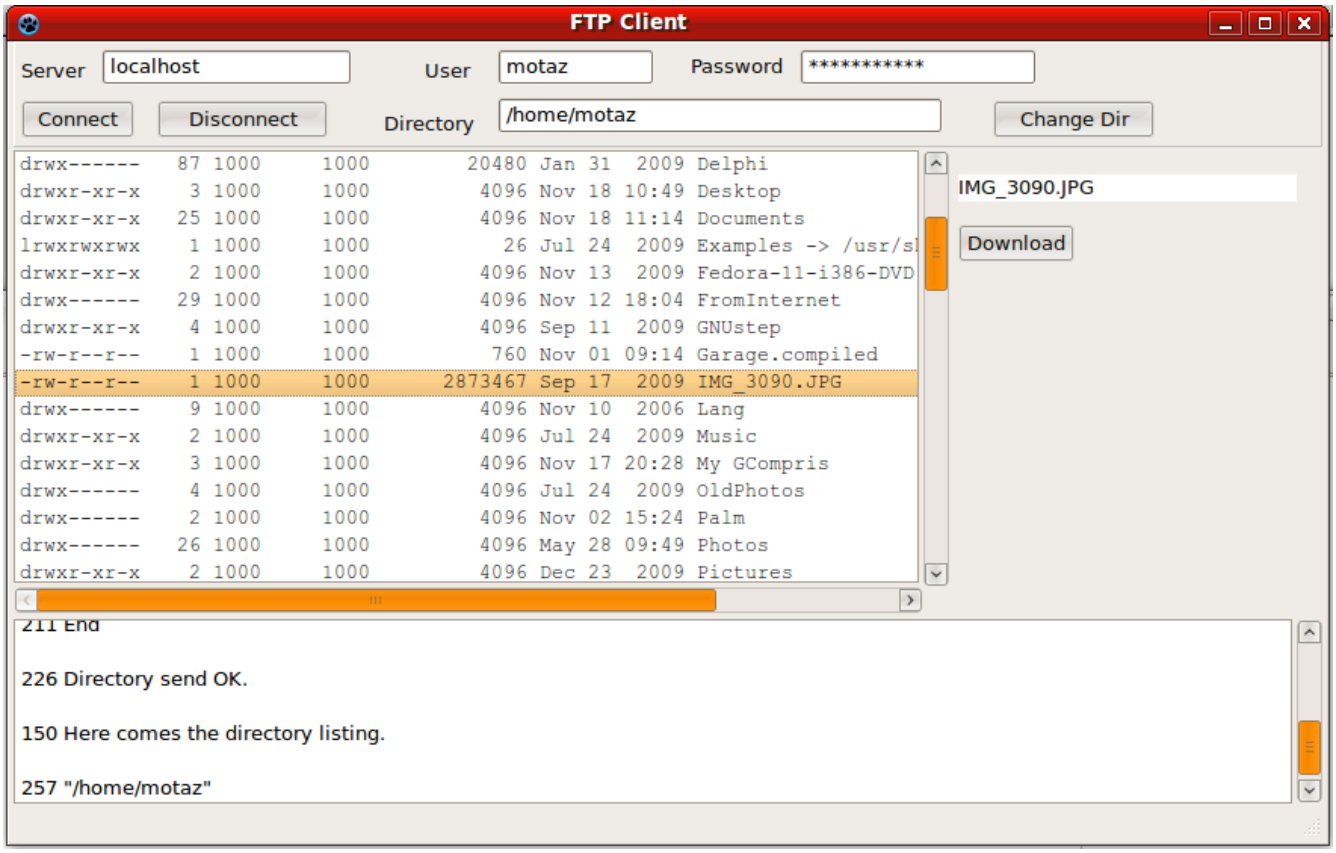
```
unit main;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls,
  ComCtrls, ExtCtrls, lNetComponents, lNet, lFTP;

type

  { TfmMain }

  TfmMain = class(TForm)
    btChangeDir: TButton;
    btConnect: TButton;
    btDisconnect: TButton;
    btDownload: TButton;
    edDir: TEdit;
    edPassword: TEdit;
    edServer: TEdit;
    edUser: TEdit;
    FTP: TLFTPClientComponent;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    laFileName: TLabel;
    ListBox1: TListBox;
    Memo1: TMemo;
    Panel1: TPanel;
    SaveDialog1: TSaveDialog;
    StatusBar1: TStatusBar;
    procedure btChangeDirClick(Sender: TObject);
    procedure btConnectClick(Sender: TObject);
    procedure btDisconnectClick(Sender: TObject);
    procedure btDownloadClick(Sender: TObject);
    procedure FTPConnect(aSocket: TLSocket);
    procedure FTPControl(aSocket: TLSocket);
    procedure FTPError(const msg: string; aSocket: TLSocket);
    procedure FTPFailure(aSocket: TLSocket; const aStatus: TLFTPStatus);
    procedure FTPReceive(aSocket: TLSocket);
    procedure FTPSuccess(aSocket: TLSocket; const aStatus: TLFTPStatus);
    procedure ListBox1Click(Sender: TObject);
  private
    GLine: string;
    DownloadStarted: Boolean;
    FS: TFileStream;

    { private declarations }
  public
    { public declarations }
  end;
```

```
var
  fmMain: TfmMain;

implementation

{$R *.lfm}

{ TfmMain }

procedure TfmMain.btConnectClick(Sender: TObject);
begin
  ListBox1.Clear;
  FTP.Connect(edServer.Text);
end;

procedure TfmMain.btChangeDirClick(Sender: TObject);
begin
  ListBox1.Clear;
  FTP.ChangeDirectory(edDir.Text);
  FTP.List('');
end;

procedure TfmMain.btDisconnectClick(Sender: TObject);
begin
  ListBox1.Clear;
  FTP.Disconnect;
end;

procedure TfmMain.btDownloadClick(Sender: TObject);
begin
  SaveDialog1.FileName:= laFileName.Caption;
  DownloadStarted:= False;
  if SaveDialog1.Execute then
    FTP.Retrieve(laFileName.Caption);
end;

procedure TfmMain.FTPConnect(aSocket: TLSocket);
begin
  FTP.Authenticate(edUser.Text, edPassword.Text);
  FTP.Binary:= True;
  FTP.ListFeatures;
  FTP.List('');
end;

procedure TfmMain.FTPControl(aSocket: TLSocket);
var
  Line: string;
begin
  if FTP.GetMessage(Line) > 0 then
  begin
    Memo1.Lines.Append(Line);
    Memo1.SelStart:= Length(Memo1.Text) - 1;
  end;
end;

procedure TfmMain.FTPError(const msg: string; aSocket: TLSocket);
begin
```

```
  StatusBar1.Panels[0].Text:= msg;
end;

procedure TfmMain.FTPFailure(aSocket: TLSocket; const aStatus: TLFTPStatus);
begin
  StatusBar1.Panels[0].Text:= 'Fail';
end;

procedure TfmMain.FTPReceive(aSocket: TLSocket);
var
  Data: string;
  i: Integer;
  Buf: array [0 .. 4095] of Byte;
  NumRead: Integer;
begin
  if FTP.CurrentStatus <> fsRetr then // List files
  begin
    Data:= FTP.GetDataMessage;
    for i:= 1 to Length(Data) do
    begin
      if (Data[i] in [#10, #13]) then
      begin
        if Length(Gline) > 1 then
          ListBox1.Items.Append(GLine);
        GLine:= '';
      end
      else
      GLine:= GLine + Data[i];
    end;
  end
  else
  if FTP.CurrentStatus = fsRetr then // Download file
  begin
    if not DownloadStarted then
      FS:= TFileStream.Create(SaveDialog1.FileName, fmCreate or fmOpenWrite);
    DownloadStarted:= True;
    repeat
      NumRead:= FTP.GetData(Buf, SizeOf(Buf));
      Memo1.Lines.Add('Downloading ' + IntToStr(NumRead) + ' Bytes');
      Memo1.SelStart:= Length(Memo1.Text);

      if NumRead > 0 then
        FS.Write(Buf, NumRead);

    until NumRead <= 0;
  end;

end;

procedure TfmMain.FTPSuccess(aSocket: TLSocket; const aStatus: TLFTPStatus);
begin
  case aStatus of
    fsPWD : edDir.Text:= FTP.PresentWorkingDirectoryString;
    fsList : FTP.PresentWorkingDirectory;
  end;

end;
```

```
procedure TfmMain.ListBox1Click(Sender: TObject);
var
  Line: string;
  i: Integer;
begin
  if ListBox1.ItemIndex <> -1 then
  begin
    Line:= ListBox1.Items[ListBox1.ItemIndex];
    for i:= 1 to 8 do
    begin
      Delete(Line, 1, Pos(' ', Line));
      Line:= Trim(Line);
    end;
    laFileName.Caption:= Line;
  end;
end;

end.
```

بعد تشغيل البرنامج وكتابة إسم المخدم أو رقمه الشبكي، وكتابة إسم المستخدم وكلمة المرور، نقوم بالإتصال، فنحصل على شكل كالآتي:



وفي الختام، نتمنى أن تكونوا قد نلتم الفائدة من هذا الكتاب والبرامج المصاحبة له.

وإلى لقاءٍ في كتاب أو مشروع آخر إن شاء الله.

معتز عبدالعظيم الطاهر

http://code.sd