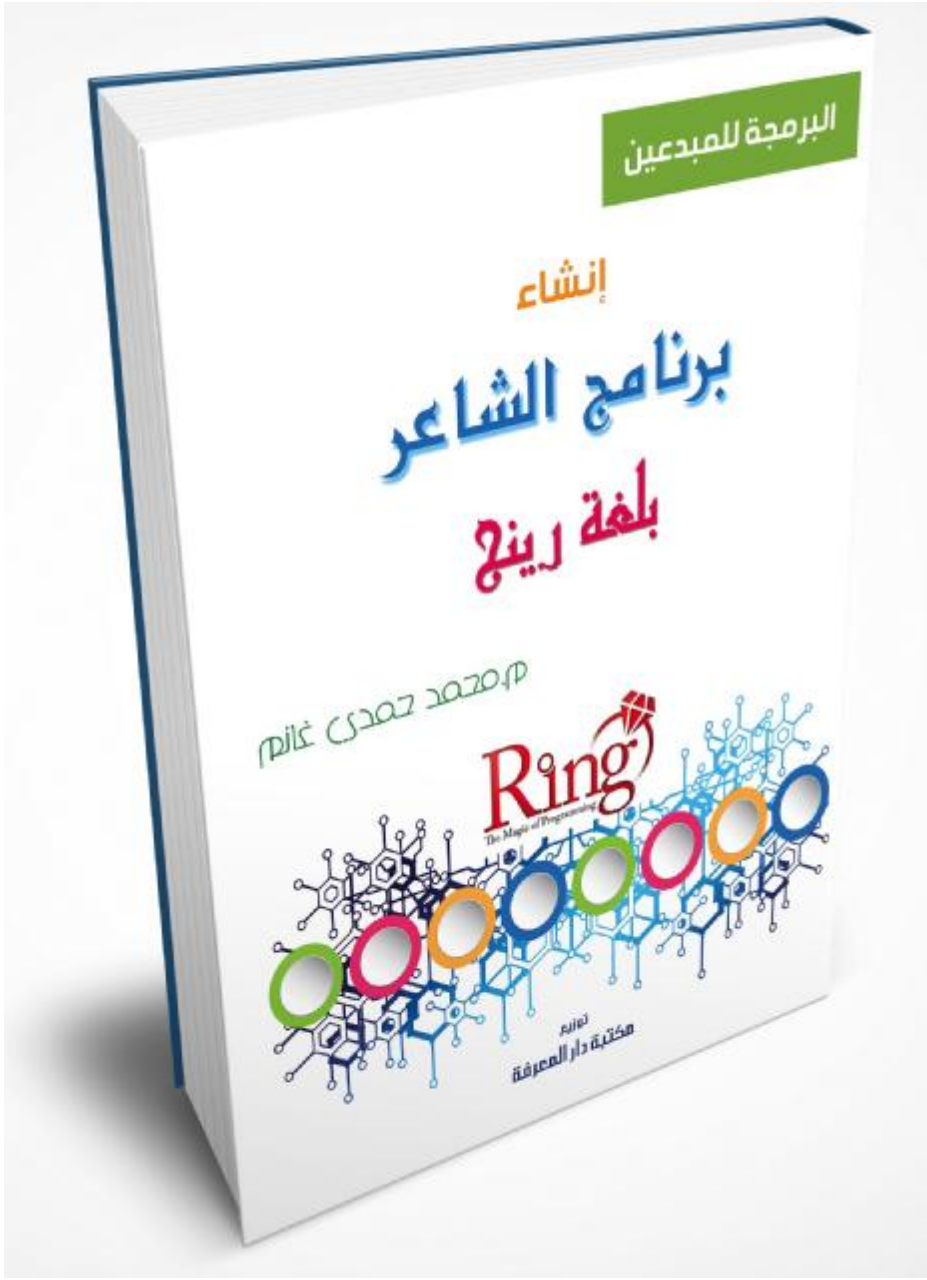


لتحميل كود برنامج الشاعر :

https://drive.google.com/file/d/1qL8ibGDl2KqXA5T0_O44TTHq3-24_IsO

البرمجة للمبدعين:

إنشاء برنامج الشاعر بلغة رينج

المؤلف:

محمد حمدي غانم

- رقم الإيداع بدار الكتب: ٢٠٢١/٢٣٩١٦
- الترقيم الدولي: 978-977-6928-38-1

تصميم الغلاف: رانيا أحمد
rania.newdream@gmail.com

الحصول على نسخة مطبوعة من الكتاب:

- **القاهرة:** مكتبة دار المعرفة في العباسية، ٣ج شارع السرايات (فوق محل فسخاني خميس)، أمام هندسة عين شمس بالقرب من محطة مترو عبده باشا.. مواعيد العمل من ١٠ صباحا إلى ٣ عصرا ما عدا الجمعة.. وأنصح بالتواصل مع أ. أحمد حماد على الهاتف أو الواتس قبل الذهاب ٠١٠٠٥٣٣٤٥٠٤

- **الجيزة:** مكتبة مشارق (مشهورة بمكتبة عمرو)، أمام مركز الدرة التعليمي السوداني في شارع النادي الرياضي وهو شارع جانبي بين أول فيصل وأول الهرم (يوجد محل أولاد رجب على أول الشارع من ناحية فيصل).. مواعيد العمل من بعد الظهر إلى العاشرة مساء.. وأنصح بالتواصل مع أ. محمد نبيل على الهاتف أو الواتس قبل الذهاب لضمان وجوده في المكتبة 01066373186.

- **الراغبون في الحصول على نسخة بطرد بريدي داخل مصر:**
يمكنهم مراسلتي على msvbnet@hotmail.com.

للتواصل مع الكاتب:

قناة يوتيوب:

https://www.youtube.com/mhmd_hmdy

صفحة فيسبوك:

https://www.facebook.com/Poet.Mhmd.Hmdy

حساب تويتر:

https://twitter.com/mhmd_hmdy

مشاريع جيتهاب:

https://github.com/VBAndCs

لتحميل كود برنامج الشاعر:

https://drive.google.com/file/d/1qL8ibGDl2KqXA5T0_O44TTHq3-24_IsO

مقدمة
الرحلة من الشاعر إلى رينج

بسم الله الرحمن الرحيم:

هذا كتاب غير تقليدي عن لغة برمجة غير تقليدية!

فأما الكتاب فهو غير تقليدي لأنه يعلمك كيف تبني خطوة بخطوة برنامج "الشاعر" الذي يحلل أبيات الشعر العربيّ موسيقيا، ويخبرك بالبحر الذي تنتمي إليه، ويعرض لك تقطيعها الموسيقي على هذا البحر.. ومن خلال إنشاء هذا البرنامج ستجد نفسك قد تعلمت ما يلي:

- البرمجة كتفكير منطقي وخطوات منظمة لحل المشاكل.. ولا يشترط لقراءة الكتاب أن تمتلك أي خلفية سابقة عن البرمجة أو درست في كليات الهندسة أو الحاسبات.. لهذا ربما يكون الكتاب مناسبا لسن ١٦ عاما فأكثر.
- أساسيات لغة البرمجة رينج، وكيف يمكن استخدامها لتطبيق أفكارك البرمجية وإنشاء برنامج بواجهة مرئية تتفاعل مع المستخدم فتأخذ منه المدخلات (الأبيات) وتعرض له النتائج (تحليل أبيات الشعر والبحر الذي تنتمي إليه).
- موسيقى الشعر وبحور الشعر العربي وأساسيات علم العروض التي وضعها الخليل بن أحمد.. وقد راعيت في هذا تبسيط القواعد والابتعاد عن المصطلحات المعقدة والتفاصيل الدقيقة التي لا تهم إلا الدارسين والنقاد، لهذا لا يشترط الكتاب أن تكون شاعرا أو تلقيت أي دراسة متخصصة في اللغة العربية.. الحد الأدنى المطلوب أن تكون ملما بقواعد الكتابة الإملائية التي تعلمناها جميعا في المرحلة الابتدائية.. وفي سبيل هذا التبسيط، قد يكون لدى بعض اللغويين والعروضيين بعض الاعتراضات على التعميمات التي يستخدمها البرنامج، لكن أرجو ألا ينسوا أن الكود كله بين أيديهم وأنهم بعد قراءة هذا الكتاب سيكونون

قادرين على فهمه وتعديله لتصير لديهم نسختهم الخاصة من برنامج الشاعر التي تعمل وفقا للقواعد التي يرونها أفضل وأدق.. فالخطأ مستدرك ووجهات النظر كلها قابلة للتعايش، دون أن يجبر أحد أحدا على ما لا يهواه.

وأمّا لغة البرمجة فغير تقليدية لأنها من إبداع المبرمج المصري م. محمود فايد، وهي بذلك أول لغة برمجة كاملة القدرات من ابتكار عربي، وقد حرصت على التأكيد على نضجها واكتمالها بإنشاء برنامج "الشاعر" بها وتقديم نسخة منه لسطح المكتب ونسخة تعمل عبر الإنترنت على موقع ويب بدون أي تعديل في الكود.. لكنّ قدرات رينج في الحقيقة تتجاوز هذا بكثير، فهي:

- لغة عابرة لنظم التشغيل، حيث يمكنك ترجمة نفس الكود إلى برامج تعمل على أجهزة سطح المكتب (على نظم ويندوز ولينوكس وماك) وعلى أجهزة المحمول (على نظام أندرويد) وعلى مواقع الويب وغيرها.
- لغة تمتلك عدة أساليب لصياغة الكود، بحيث يمكنك أن تكتب الكود بصيغة قريبة من لغات البيزيك أو تكتبه بصيغة قريبة من لغات سي أو تكتبه باللمسات الخاصة بلغة رينج.. ولا مانع أن تمزج بين هذه الصيغ الثلاث في نفس الكود.. بل يمكنك حتى أن تعيد تعريف الكلمات الأساسية الخاصة باللغة لتصنع الصياغة التي تناسبك أنت.
- لغة برمجة مرنة الأنواع Dynamic-Type Language فالمتغيرات تستطيع أن تحمل أنواعا مختلفة من البيانات.
- لغة برمجة هيكلية Structural Language تتبع التسلسل التقليدي لتنفيذ الأوامر.
- لغة برمجة موجهة بالكائنات Object Oriented Language.
- لغة برمجة موجهة بالدوال Functional Language.
- لغة برمجة تدعم استخدام اللغة طبيعية Natural Language Programming.

باختصار: لغة رينج تجمع ميزات عدة أنواع من لغات البرمجة في وعاء واحد، حيث يمكنك اختيار الصيغة التي تناسب برنامجك لكتابة الكود بأسرع وأكفأ طريقة.

وفي هذا الكتاب سنتعرف على سمات رينج كلغة برمجة هيكلية مرنة موجهة بالكائنات، أما قدراتها كلغة موجهة بالدوال وبرمجة اللغات الطبيعية فيمكنك التعرف عليها من خلال مصادر التعلم الموجود روابطها في مجلد أمثلة الكتاب.

لمن هذا الكتاب:

- لو كنت مبتدئا تريد التعرف على عالم البرمجة، فهذا الكتاب مدخل مناسب لك، ولغة رينج لغة مجانية يسهل عليك تنزيلها وتشغيلها على جهازك، وأسلوب صياغتها المرن سيجعل ما تتعلمه منها يفيدك عند الانتقال إلى لغة برمجة أخرى من عائلة لغات البيزيك (مثل VB.NET) أو لغات السي (مثل C++ أو Javaأ و C#) أو لغات البرمجة الديناميكية مثل Python و Ruby.
- لو كنت باحثا لغويا تريد أن تتعرف على كيف يمكن استخدام البرمجة لمعالجة اللغة العربية، فبرنامج الشاعر يقدم لك مثالا عمليا قويا على هذا المجال.
- لو كنت مبرمجا لديك خبرة بلغة برمجة سابقة، فسيفيدك أن تتعرف على لغة رينج وإمكاناتها المتنوعة، وربما تجد نفسك متحمسا للمشاركة في تطويرها بالانضمام إلى فريق الكود مفتوح المصدر الذي يطورها على جيتهاب.. ومن المؤكد أيضا أن كتابة برنامج لتحليل أوزان القصائد العربية هو خبرة مختلفة وجديدة تمثل تحديا غير تقليدي يضاف لخبراتك البرمجية، وسط طوفان الأكواد التقليدية التي تكتبها يوميا للتعامل مع قواعد البيانات وتصميم واجهات سطح المكتب والويب.. بإذن الله لن تخرج بدون مكاسب بعد قراءة هذا الكتاب.

الشاعر: البرنامج الذي جعلني مبرمجا:

من القلائل الذين تركوا بصمة في شخصيتي في كلية الهندسة، أ. د. محسن رشوان، وهو أستاذ الالكترونيات والهندسة الكهربية في قسم اتصالات بكلية الهندسة جامعة القاهرة، وشخصية رائعة أثرت في أجيال، وهو أحد مؤسسي الشركة الهندسية لتطوير البرمجيات RDI واسمها اختصار لـ "مؤسسة البحث والتطوير":

Research & Development Institution

وكان شعاره "إن لم نكسب منها نقودا، نكون قد أنتجنا بحوثا".. وتحت هذا الشعار الرائع من عالِم يبحث عن إضافة حقيقية للعلم في مصر والوطن العربي وليس النقود في المقام الأول، اقتحمت الشركة مجالات معقدة في التسعينيات، مثل ضغط الصوت والتعرف على الكلمات العربية المنطوقة أو المكتوبة، والتحليل الصرفي وفض الالتباس بين الاحتمالات الصرفية من خلال السياق، ونطق الكلمات العربية المكتوبة، والتشكيل النحوي الآلي، وصولا إلى "حفص": شيخ الكتاب الرقمي الذي يقيّم تجويدك للقرآن الكريم من خلال قراءتك الصوتية!.. وكل من يعملون في هذه التقنيات سجلوها كرسائل ماجستير ودكتوراه ونشروا عنها بحوثا في دوريات عالمية مشهورة، ومنهم أستاذي وصديقي د. م. محمد عطية العربي.

وبجوار ما يشرحه من مواد تخصصه، كان أ. د. محسن رشوان يشرح مادة الإنسانيات في الكلية، وهي مادة عابرة للتخصص الهندسي، تحاول أن تمنح المهندس خلفية بسيطة عن الاقتصاد والقانون والإدارة وغيرها من الأمور التي سيحتك بها المهندس في عمله بعد ذلك.

ومن خلال هذه المادة، كان د. محسن يمنحنا بعض الخبرات في إدارة البرمجيات، وفي السنة الثانية لنا في قسم الاتصالات بكلية الهندسة، طلب من كل منا أن يكتب في نصف العام مشروعا برمجيا بأي لغة برمجة وعن أي فكرة جيدة يختارها، منفردا أو مع مجموعة طلابية، ليعطينا على هذا المشروع درجات أعمال السنة.. وكانت هذه بدايتي الحقيقية في عالم البرمجة في يناير ١٩٩٨، حيث لجأت إلى صديقي وبلدياتي م. محمد جلال طالبا المساعدة.. كان محمد يدرس حينها في السنة الثانية في كلية

الحاسبات والمعلومات بجامعة عين شمس، وفي نصف العام كنا أنا وهو نمضي الإجازة في مدينتنا في دمياط.

ولا أنسى هنا أن أدعو بالرحمة لوالدة الصديق محمد جلال، فلطالما أكرمت ضيافتي في الأيام التي كنت أذهب فيها إلى بيت محمد لنكتب البرنامج.. رحمها الله وغفر لها وأسكنها فسيح جناته.

كانت تراودني في تلك الفترة فكرة جريئة، هي أن أكتب برنامجا أدخل إليه بيتا من الشعر، فيخبرني بالبحر الذي ينتمي إليه.. وحينما ناقشت هذا مع محمد جلال، شعر أنني مجنون، إذ كيف يمكن للحاسب أن يفعل شيئا كهذا؟.. فطلبت منه ألا يقلق، وأنني سأدرس الفكرة، وأقدمها له في شكل رياضي يستطيع الحاسب التعامل معه.

وبالفعل استعرت كتابا عن علم عروض الشعر من الصديق الشاعر أ. سامح النجار، وشرعت في العمل.

والحقيقة أنها كانت من أمتع فترات حياتي، أن أمزج بين عالم الشعر الذي أكتبه، وعالم الهندسة الذي أدرسه في عالم البرمجة الذي كنت شغوفا بتعلمه.

وما زلت حتى الآن أحتفظ بالأوراق التي حللت فيها العروض إلى قواعد تصلح للبرمجة (سأشرحها باختصار في هذا الكتاب).. ولم أتبع العروض الرقمي للدكتور أحمد مستجير فلم أكن أعلم بوجوده أصلا، وإنما اتبعت قواعد عروض الخليل بن أحمد الأساسية.

وقد افترضت أننا سنتعامل مع شطرة شعرية طولها الأقصى ٤ وحدات عروضية (تفعيلات)، وتخيلت كيف سيتم الأمر برمجيا، وذهبت إلى محمد جلال، الذي كان يمتلك حاسبا شخصيا ولديه خلفية بسيطة عن لغة فيجوال بيزيك ٤ التي درسها في التدريب الصيفي لمدة أسبوع لا أكثر، ونصحني أن نستخدمها لسهولتها مقارنة بعائلة لغات C++.. وكان هو المسئول عن كتابة البرنامج، وأنا مسئول عن الخوارزمية.

في البداية كنا نحتاج إلى تقطيع النصوص إلى حروف، ولم يكن م. محمد جلال يعرف الدالة التي تفعل هذا في فيجوال بيزيك.. وبدون انترنت وعدم توفر مكتبات فيها

مراجع، كان الأمر معاناة حقيقية، حتى اضطررنا إلى أن نرسل سؤالا إلى أستاذ في كلية العلوم بدمياط وقد تفضل مشكورا بإخبارنا بدوال تقطيع النصوص في الفيجوال بيزيك: Left و Right و Mid.. هذه دوال لها معزّة خاصة عندي، وما زالت موجودة في فيجوال بيزيك دوت نت حتى الآن بالمناسبة، ولهذا لا تستغرب من إضافتي لهذه الدوال في الفئة aString التي كتبتها بلغة رينج للتعامل مع النصوص العربية كما سنرى لاحقا في الكتاب ☺.

وهكذا تجاوزنا أول خطوة بحمد الله، ونجحنا في تقطيع الحروف، وطبقنا قواعد الكتابة العروضية وحصلنا على الرموز الصوتية، ولم يعد باقيا إلا تحليل هذه الرموز لنحصل على التفعيلات مثل (مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مُتَفَاعِلُنْ).. وبدأ محمد يطبق الخوارزمية التي كتبتُها، ولكنها لم تعطنا أي نتائج، وكانت صدمة محبطة لم نستطع أن نصل إلى سببها، رغم أنها ضيعت منا وقتا طويلا، إلى أن أجبرنا انقطاع التيار الكهربي على التوقف عن المحاولة!

وغادرت بيت محمد جلال شاعرا باليأس وأنا أفكر في فكرة مشروع جديد.. ولكنه اتصل بي في اليوم التالي، ليبشرني بأن الإلهام هبط عليه ليلا وهو في فراشه، وأنه اكتشف سبب المشكلة في خوارزميتي، واقترح تعديلا، فذهبت إليه ملهوفا.

كانت خوارزميتي تجرب تقطيع الرموز إلى أطوال مختلفة (أربعة حروف أو خمسة أو ستة أو سبعة) ومقارنتها برموز التفعيلات للوصول إلى التتابع الصحيح للتفعيلات.. ولكني تعاملت مع هذا التقطيع بصورة خطية وافترضت أنني أتعامل مع مصفوفة مكونة من ٤ خانات، بينما اكتشف محمد أن المفروض أن تكون هناك شجرة احتمالات للأطوال المختلفة، كل جذر في هذه الشجرة تخرج منه ٤ فروع لدراسة كل التفعيلات المحتملة في كل موضع، وأننا لهذا نحتاج إلى مصفوفة عدد خاناتها ٤ + ١٦ + ٦٤ خانة للتعامل مع شطرة واحدة تتكون من ٣ تفعيلات!

(لا تقلق فسأشرح هذه التفاصيل في الكتاب، وستكتشف أنني مع الخبرة وصلت إلى أسهل وأفضل طريقة لتنفيذها.. لا تنسَ أنها كانت أول محاولة لمبتدئين في البرمجة).

وبالفعل عدلنا الخوارزمية لتغطي كل فروع الشجرة، وشرعنا في تجربتها.. ولكم أن تتخيلوا مقدار الفرحة التي شعرنا بها حينما رأينا النتائج تظهر لأول مرة!.. هذه البهجة تتكرر في حياة المبرمجين كثيرا، لكنها لا تأتي طبعا إلا بعد كثير من العرق والدموع وسلاسل من الإحباطات التي يمرون بها بسبب الأخطاء التي تحدث في الكود ☺.

وهنا ظهرت مشكلة عجيبة.. فالبرنامج بدلا من أن يعرض لنا: "مستفعلن مستفعلن"، عرض لنا: "فعلن فعولن فاعلن"!

وشعرت مرة أخرى بالخجل أمام محمد جلال، فمن الواضح أن الخوارزمية التي كتبتها فاشلة تماما!

لكن بعد قليل من التأمل، اتضح لنا أن "فعلن فعولن فاعلن" مكافئة صوتيا تماما لـ "مستفعلن مستفعلن" وأن هذا ليس خطأ منا ولكنها مشكلة في طريقة تقسيم التفعيلات في العروض الخليلي، وبتتبع الشجرة وجدنا أن هناك فعلا "مستفعلن مستفعلن"، وأنّ علينا استبعاد بعض الاحتمالات غير المصطلح عليها في علم العروض، فالتتابع "مستفعلن مستفعلن" ينتمي للبحر الكامل أو بحر الرجز، لكن "فعلن فعولن فاعلن" غير مستخدم (سأشرح في هذا الكتاب طريقة بسيطة وسريعة لاستبعاد هذه الاحتمالات).

وبدأنا في كتابة جمل الشرط النهائية التي تحصل على البحر من احتمالات التفعيلات.. وبدأ محمد جلال يتركني أكتب بعض جمل الشرط والكود بنفسي، واكتشفت أنني تعلمت الكثير بمشاهدته وهو يكتب، وكان مشكورا يشرح لي ما يفعله، وهو الذي علمني فيجوال بيزيك ٤ و فيجوال بيزيك ٥ الذي حصلنا عليه ونحن نكتب البرنامج وفي الصيف حصلنا على فيجوال بيزيك ٦ التي صدرت في ذلك العام، وأنا مدين لصديقي المهندس محمد جلال بهذا الفضل، أنا وكل من تعلم فيجوال بيزيك وسي شارب من كتبي، وكل من سيتعلمون رينج من هذا الكتاب أيضا.. فجزاه الله عنا خيرا، ولا تنسوه من دعائكم.

والحقيقة أننا فعلنا معجزة في أسبوعين، سواء بإنجاز الخوارزمية، أو تنفيذها بفيجوال بيزيك.. فلم نكن نعرف حتى كيف نكتب الدوال بفيجوال بيزيك، واضطررنا إلى كتابة

البرنامج كله في حدث ضغط الزر كإجراء واحد يتكون من مئات السطور، واستخدمنا أعقد كود إسباجيتي يمكنكم أن تتخيلوه حيث نقفز بجمل GoTo من كل مكان إلى كل مكان لتكرار الأجزاء التي من المفروض أن توضع في دوال!

وقد انتهت الإجازة وعدنا إلى القاهرة قبل أن نكتب أسماء كل البحور في البرنامج بسبب مشكلة مع الحروف العربية اكتشفنا لاحقا أن سببها نوع الخط، وحينما علم بالبرنامج الصديق الشاعر م. نزار شهاب الدين وهو مهندس اتصالات زميل في كلية الهندسة، عرض أن يكتب أسماء البحور الناقصة على حاسوبه، فذهبت معه، واضطر إلى كتابة أسماء البحور بحروف إنجليزية لأنه لم يستطع أيضا معرفة سبب المشكلة التي تجعل الحروف العربية تظهر كرموز غير مقروءة.

وكنا حينها نحتاج إلى تشكيل كل حروف بيت الشعر، وكان هذا يجعل الكتابة بطيئة خاصة لغير المتمرسين بلوحة المفاتيح مثلي، لكن نزار كان يجيد الكتابة بأصابعه العشرة على لوحة المفاتيح بدون حتى أن ينظر لها (وهي مهارة أغبطه عليها ما زلت عاجزا عن تعلمها حتى الآن)، لهذا فقد تطوع مشكورا بأن يكون هو من يكتب بيت الشعر الذي سيقوله د. محسن رشوان وهو يختبر البرنامج.

وأذكر دهشة د. محسن رشوان حينما مر علينا وسمع فكرة البرنامج الذي يقوم بوزن الشعر، وتعليقه:

- يا سلام!

ثم انشغل في مشاريع أخرى، وتابع معنا د. شريف عبد العظيم وطلب اختبار بعض أبيات الشعر ونجح البرنامج في وزنها بشكل صحيح، بشهادة نزار طبعا، لأن معلومات أساتذة الهندسة في الشعر ليست بهذا العمق ☺.. والحمد لله أخذت الدرجة النهائية.. وقد ذكرت اسم محمد جلال على التقرير، رغم أنه ليس معنا في الكلية ☺.

وفي الصيف قررنا أنا ومحمد جلال أن نطور برنامج الشاعر.. وبسبب شغفي بفيجوال بيزيك أخذت أقرأ عنها، وهذا جعلني أكتشف عيوب التنظيم في الكود الذي كتبناه، فتخلصت من جمل GoTo واستخدمت الدوال، وبدأنا نعمم البرنامج ليسمح بكتابة

قصيدة كاملة بدلا من بيت واحد من الشعر، وأضفنا شرحا لأساسيات علم العروض وتعريفات لمصطلحاته الأساسية، وبعد ذلك حلّلت معجم ديوان الأدب الذي ينظم الكلمات تبعا لأوزانها، واستخدمته لإضافة معجم الأوزان والقوافي للبرنامج، وقام بكتابته في قاعدة بيانات الصديق م. شريف محمد حمدي، وبهذا صار من الممكن أن أبحث عن الكلمات التي لها ووزن معين وقافية معينة، وهذا أفادني كثيرا في كتابة الشعر العمودي. (لا يتسع هذا الكتاب لشرح معجم الأوزان والقوافي لهذا لم أضفه لنسخة رينج من البرنامج).

بعد ذلك اشتري لي أبي رحمه الله حاسوبا لما رأى جنوني بالبرمجة، فأخذت أطور البرنامج، وسمحت بكتابة شعر التفعيلة، الذي يمكن أن يطول فيه السطر الواحد عن ٤ تفعيلات بكثير (وهو ما سنفعله في هذا الكتاب أيضا).. وهنا كان يجب تغيير خوارزمية الشجرة الرباعية الخاصة بمحمد جلال لأنها كانت تسبب خطأ تجاوز مساحة الذاكرة بعد عدد معين من التفعيلات بسبب زيادة عدد خاناتها بالأس الرباعي مع كل تفعيلة، ويحتاج التعامل مع خمس تفعيلات فقط مصفوفة عدد خاناتها = ٤ + ١٦ + ٦٤ + ٢٥٦ + ١٠٢٤ = ١٣٦٠ خانة، والتفعيلة السادسة ستضيف ٤٠٩٦ خانة على العدد السابق، وهلم جرا!.. لهذا كتبت خوارزمية خطية عامة، يمكن تسميتها بعناقيد التفعيلات، وهي عكس الخوارزمية الأولى، التي كنا نحدد فيها مقاطع ثابتة أطوالها ٤ أو ٥ أو ٦ أو ٧ حروف، ونحصل على كل تباديل وتوافيق هذه المقاطع في النص، وهذا يؤدي إلى عدد هائل من المقاطع المحتملة.. بينما في خوارزمية عناقيد التفعيلات كنت أحصل على عنقود التفعيلات المحتملة عند كل موضع في النص بغض النظر عن طولها، وتصير المسألة هي البحث عن مسار التفعيلات المستمرة، التي تصل بك إلى آخر حرف في النص بدون فجوات.. وفي نسخة رينج، لن نحتاج حتى إلى تكوين عناقيد التفعيلات، فنحصد النتائج فورا ونحن نمر على شجرة الاحتمالات، كما سأشرح في هذا الكتاب.

وقد خفّفت قواعد تشكيل الحروف بحيث يتطلب البرنامج وضع السكون والشدة والتنوين فقط على الحروف (وهو ما سنفعله في هذا الكتاب).. ويستطيع البرنامج تجاهل أي تشكيل يظهر في موضع غير منطقي (سأترك لك فعل هذا كتدريب عملي).. وقد بسّطت الكود الذي يحول الكتابة الإملائية إلى كتابة عروضية، لأنه سبب لي صداعا حينما كتبته بجمل شرط عادية، حيث كانت طويلة جدا ومتداخلة، فكتبت ما يشبه التعبيرات النمطية Regular Expressions (التي ظهرت بعد ذلك في دوت نت ولم تكن معروفة في فيجوال بيزيك ٦) حيث قمت بتعريف دوال بسيطة تستقبل بعض الصيغ الخاصة بي وتبحث عنها في النص وتحولها إلى صيغ أخرى.. بهذه الصيغ البسيطة تحول كود الكتابة العروضية إلى مجرد سطور قليلة واضحة المعنى.. هذا جعل الكود مختصرا وبسيطا وواضحا وسهل تعديله.

لكننا هنا في رينج سنفعل ما هو أفضل من هذا، وهو كتابة محرك تعبير نمطي Regular Expression Engine خاص بنا واستخدامه لتطبيق قواعد الكتابة العروضية.. هذا المحرك سيعطينا بعض القدرات الجديدة التي لا توجد في محركات التعبيرات النمطية القياسية، مثل كتابة دوال لفحص القيم Validation Functions بلغة رينج واستخدامها ضمن صيغة التعبير النمطي.

وقد حاولت حينها أن أضيف جزءا يحلل الأبيات المكسورة ليساعد في تحديد مواضع الكسر وإصلاحها.. وأضفت هذا المحلل إلى الإصدار الأول فعلا لكنه لم يعمل بشكل صحيح.. لكن منذ حوالي خمس سنوات أعدت كتابة نسخة من برنامج الشاعر بلغة فيجوال بيزيك دوت نت باستخدام تقنية WPF، وأضفت للبرنامج قاموسا للأوزان والقوافي مبني على لسان العرب يحوي ملايين الكلمات، بنيته آليا باستخدام محلل صرفي بدائي كتبته بنفسي في فترة الجامعة، وأعدت التفكير في مشكلة الكسور العروضية، ووصلت لحل نهائي لها وأضفته للبرنامج فعلا، ويعمل بكفاءة تامة، لكن الكتاب هنا لا يتسع لشرح هذا، لهذا اعتبره تحديا لنفسك وحاول أن تضيفه للبرنامج.

وهذا شجعني للانتقال للخطوة التالية، وهي اقتراح تصحيح لكسور الوزن باقتراح تغيير

أو إضافة أو حذف تشكيل، أو حتى اقتراح كلمات بديلة.. خاصة أن لدي التحليل الصرفي والكلمات، وكل المطلوب هو اقتراح التصحيح.. لكن المحاولة الأولى لتعديل التشكيل فشلت وتسبب خطأ حتى الآن، وانشغلت عنها ولم أكملها.. سأعود إليها في أي وقت بإذن الله لأكمل هذا الهدف.. ومن يدري، فربما تسبقني أنت إليه.. لديك برنامج الشاعر، ومعك لغة رينج، فما المانع؟

يمكن أن أقول إن برنامج الشاعر علمني عروض الشعر العربي والبرمجة وفيجوال بيزيك و WPF واستعنت به في كتابة الشعر، وكان جزءًا من سيرتي الذاتية حينما حاولت أن أعمل مبرمجا وأنا ما زلت طالبا لكن للأسف لم أوفق لأن الشركات كانت تريد مبرمجا متفرغا، لكني تعرفت على أ. بسيوني فتحي في شركة المتحدة وأعجب ببرنامج الشاعر لأنه خريج كلية دار علوم اللغة العربية، وقابلته بعدها في نادي كلية دار العلوم، وبعد التخرج والجيش، ذهبت مرة أخرى إلى المتحدة وقابلته وتذكرني، وأرسلني إلى مدير قسم البرمجة أ. سامي قنديل وتمت المقابلة بسلاسة وعملت في شركة المتحدة.

وبسبب عشقي للبرمجة بعد برنامج الشاعر، اخترت أن يكون مشروع تخرجي في كلية الهندسة مشروعا برمجيا مع د. محسن رشوان، وكان عن التعرف على الصور هل تحتوي على نصوص أم رسوم.. وقد أرسلني د. محسن إلى د. محمد عطية في شركة RDI ليساعدني في هذا المشروع، وهو مهندس اتصالات يكبرني بخمس سنوات، فانبهرت أنه بعقليته المذهلة وغزارة ثقافته في كل المجالات، التي تجعلك لا تمل من الحوار معه لساعات، لهذا صار أستاذي وصديقي من حينها إلى اليوم، وهو حاليا مواطن كندي، وله بحوث محكّمة في الدوريات العلمية العالمية وصار محكّما أيضا في بعض هذه المجلات، ولنا قصيدة مشتركة نبتت من تجلياته الرياضية هي قصيدة ظلال من رؤاي، التي يقول في مطلعها:

جميلٌ أنتَ في كلِّ المرايا = رشيقُ القَدِّ في كلِّ الزوايا

وقد قادني هذا المسار إلى ترجمتي لمرجع Mastering VB.NET عام ٢٠٠٣

ونشرته مجانا، وبسبب شهرته بدأت أكتب كتبي في البرمجة من عام ٢٠٠٨ إلى اليوم.
وقد كان هذا الكتاب سببا في معرفتي بالمهندس محمود فايد، فقد تأثر كثيرون وهو منهم بهذا الكتاب وخاصة مقدمته التي دعوت فيها المبدعين العرب إلى نشر العلم النافع خاصة في علوم البرمجة، فكتب كتابا عن فوكس برو وذكرني في الإهداءات، فلفت هذا نظري لإبداعاته، التي ظللت أتابعها عن بعد عبر ١٥ عاما، إلى أن قرأت عن إبداعه للغة رينج في مقال على صفحة "الباحثون المسلمون"، وأعجبت كثيرا بمواصفات اللغة التي تحاول أن تعطي للمبرمج الحرية المطلقة حتى في إعادة تعريف كلماتها الأساسية.

وقد راودتني دائما رغبة ملحة في أن أكتب كتابا عن برنامج الشاعر أشرح فيه فكرته، وأشرح من خلاله أساسيات علم العروض وأساسيات البرمجة، لكني كنت قد كتبت ما يكفي من الكتب عن لغتي VB.NET و C#، ولم يعد من المفيد أن أعيد تغطية نفس اللغتين بكتاب جديد يتناول الأساسيات، لهذا وجدت أن رينج ربما تكون أنسب لغة لهذا الغرض، فلا توجد كتب عربية عنها حتى الآن، ونقل كود البرنامج إليها يثبت قدرتها، كما أن وضع "الشاعر" و "رينج" في سياق واحد يجعل هذا الكتاب احتفاء واحتفالا بالإبداع العربي شعرا وعروضا وبرمجة، ويقدم للعالم كتابا غير مسبوق بكل معنى للكلمة، يثبت أن الإبداع العربي مستمر من أقدم شاعر عربيّ، مرورا بالخليل بن أحمد واضع علم العروض، وصولا إلى المبرمجين ومطوري لغات البرمجة العرب، وانتهاء إليك: أنت القارئ العربي المنوط به حمل هذا القبس ومواصلة هذا الإبداع.

ولأن الفكرة ولادة، فقد شجعني تعاملي مع رينج على إنشاء لغة Small Visual Basic بدءا من ٢٠٠١ وعلى مدار عامين، لتلافي عيوب لغة Small Basic!

جزى الله خيرا كل من علمني وشجعني وساندني، ممن ذكرتهم بالفضل في هذا السرد، أو لم أذكرهم.. الفضل لا ينسى.

وأخيرا وليس آخرا، بقي أن نسرد نبذة تاريخية قصيرة عن لغة رينج، وفريق المبدعين الذي يقف خلفها، لنعطي كل ذي فضل حقه.

رينج: لغة برمجة بأياد عربية:

مع بدء عامه العشرين في ديسمبر ٢٠٠٥، بدأ المهندس محمود سمير فايد – الذي كان ما يزال حينها طالبا بكلية الهندسة بجامعة المنوفية المصرية – في بناء الجيل الأول من تقنية البرمجة بدون كود Programming Without Coding Technology أو اختصار PWCT، وهي أداة لتطوير البرامج والتطبيقات من خلال البرمجة المرئية Visual Programming دون الحاجة الى كتابة الكود فى صورة نصية.. تفاصيل أكثر عن هذه التقنية في ويكيبيديا:

https://en.wikipedia.org/wiki/PWCT_(software)

وقد كتب م. محمود هذه التقنية باستخدام فوكس برو Microsoft Visual FoxPro بجانب لغات برمجة أخري مثل C و Harbour و Python وغيرها، والتى يستخدمها المشروع لتوليد الكود Code Generation.

ولكن ميكروسوفت توقفت فجأة عن تطوير Visual FoxPro بعد آخر إصدار لها في نهاية عام ٢٠٠٧، وهو ما أجبر م. محمود فايد على البحث عن لغة برمجة أخري لإعادة بناء المشروع بالكامل (أكثر من ٢٥٠ ألف سطر من الكود!).

وفى عام ٢٠١١ بدأ م. محمود فايد في إنشاء جيل جديد تقنية البرمجة بدون كود يواكب متطلبات العصر، ولا يكون قاصرا كالجيل الأول على نظام الويندوز فقط، بل يمكنه العمل على مختلف الأنظمة لتطوير تطبيقات سطح المكتب Desktop ومواقع الإنترنت Web وتطبيقات الجوال Mobile.

وقد فرضت طبيعة المشروع أن يعتمد على لغة برمجة ديناميكية Dynamic حتى يكون أكثر بساطة ومرونة، ولهذا فكر م. محمود في استخدام لغة بايثون Python أو روبي Ruby لإعادة بناء المشروع.

لكن لأسباب شخصية لم يقنعه أسلوب صياغة الكود Syntax الخاص بلغة بايثون، كما أنه أراد استخدام مكتبات Qt لبرمجة الواجهة الرسومية، وفى ذلك الوقت كان المتوفر لدعمها فى لغة بايثون مكتبة PyQt وكانت الرخصة الواحدة منها للاستخدام

التجاري مكلفة، وهذه التكلفة سيتحملها مستخدمو تقنية البرمجة بدون كود فى المستقبل، وهو ما لا يريده.

وبالتفكير فى لغة روبى Ruby، كان م. محمود يري بشكل شخصي أنها أجمل من حيث أسلوب الصياغة Syntax والمرونة أيضا، وتدعم مكتباتها تصميم مواقع الويب Web بقوة، لكن بتجربة دعمها لمكتبة Qt اتضح أنها غير ملائمة ولا تتمتع بمستوي الجودة المطلوب لإنجاز مشاريع كبيرة فى ذلك الوقت (٢٠١١).

لهذا كان الحل البديهى أن يكتب بنفسه المكتبات التي يحتاجها للعمل مع لغة بايثون أو روبى، ولكن إطلاعه على لغة روبي وانبهاره بمدي جمالها جعله يدرك أكثر مدى الروعة التي يمكن أن تصل إليها إذا قام بتصميم لغة برمجة جديدة تتميز بمزايا خاصة.

وقد كانت للمهندس محمود فايد تجربة فى تصميم لغة السوبرنوفا في عامي ٢٠٠٩ و ٢٠١٠، وهي لغة تتيح كتابة الكود بكلمات عربية أو إنجليزية، وهى مخصصة للواجهة الرسومية فقط والبرامج البسيطة، فهى لغة بحثية لتجربة بعض الأفكار وليس للاعتماد عليها بشكل كامل.. من هنا بدأ التفكير بشكل أكثر جدية فى عمل لغة برمجة جديدة يحقق من خلالها التالي:

١. لغة سهلة وبسيطة تجمع المزايا التى أعجبته فى اللغات الديناميكية مثل Python و Ruby.
٢. لغة صغيرة الحجم مثل لغة Lua يمكن بسهولة دمجها داخل مشاريع لغة C و C++.
٣. لغة ذات مكتبات قوية وأدوات تطوير تشبه لغة فيجوال بيزيكVisual Basic لتكون أكثر إنتاجية.
٤. لغة يطبق بها أفكاره الجديدة وينقل إليها أفكاره السابقة التى قام بتطويرها فى لغة سوبرنوفا Supernova فيما يتعلق ببرمجة اللغات الطبيعية Natural Language Programming

٥. لغة تتيح بسهولة إنشاء لغة مخصصة لمجال معين Domain-Specific Language بطرق جديدة تسهل دعم البرمجة التعريفية Declarative Programming والبرمجة الطبيعية Natural Programming.
٦. لغة مرنة من حيث أسلوب الصياغة Syntax الذي توفره بأكثر من شكل، مع إمكانية تغييره حسب المشروع أو فريق العمل.
٧. لغة مصممة باستخدام تقنية البرمجة بدون كود كإثبات عملى على قوة التقنية.
٨. لغة يمتلكها م. محمود فايد ويتحكم فى تطويرها بشكل مستمر ولا يتعرض لمخاطر فقدانها أو توقف الشركة المنتجة عن تطويرها بعد تجاربه المريرة السابقة، مثل توقف شركة CA عن تطوير لغة Clipper وتوقف شركة Microsoft عن تطوير Visual FoxPro.

ومن هنا كان ميلاد لغة رينج Ring والتى بدأ م. محمود فايد تطويرها في سبتمبر ٢٠١٣، وتوج م. محمود فايد ثلاثين عاما مع عمره أمضى معظمها شغوفا بالبرمجة بنشر أول إصدار لغة رينج فى شهر يناير عام ٢٠١٦ كمنتج حر مفتوح المصدر Free Open Source.

وقد كان الإصدار الأول من لغة الرينج (مترجم الكود Compiler + المكتبات البرمجية + الأمثلة) يحتوي على حوالى ١٠٠ ألف سطر من الكود.. أما اليوم فتتخطي لغة رينج ٥٠٠ ألف سطر من الكود، لكن تبقى نواة المشروع صغيرة جدا (حيث صمم المترجم Compiler وآلة التشغيل الافتراضية Virtual Machine) بدقة عالية ليكونا صغيري الحجم (٢٠ ألف سطر من الكود فقط)، بينما تتخطى مكتبات رينج اليوم مئات الآلاف من سطور الكود.

فريق عمل رينج:

يحاول هذا المشروع التقنى أن يعكس رسالة مهمة فى عالمنا العربي:

"بإستطاعتنا تطوير مشاريع مفتوحة المصدر فى مجالات متقدمة يشارك بها عدد كبير من المطورين وتستمر لسنوات طويلة وتستخدم فى مختلف أنحاء العالم.. لا شيء يمنعنا إذا توفر الدافع والدعم، فلدى شبابنا كل الإمكانيات التى تؤهله لذلك".

هذه الرسالة ستصلك بوضوح حينما تنظر لفريق تطوير اللغة، والإسهامات التي قدمها.. فمنذ طرح الإصدار الأول من لغة رينج، انضم لها عدد كبير من المطورين من جنسيات مختلفة، ليساهموا فى اختبارها واكتشاف الأخطاء وتطوير الأمثلة والتطبيقات التي يتم إرفاقها مع اللغة بجانب تحديث مكتباتها.. ويتواصل هؤلاء المطورون من خلال مجموعة جوجل التالية:

https://groups.google.com/forum/#!forum/ring-lang

ومن أهم هؤلاء المطورين والذين لهم مساهمات كبيرة فى المشروع:

- Gal Zsolt من المجر: كتب مئات الأمثلة بلغة رينج ويحمل لقب "سيد الأمثلة" Samples Master فى فريق التطوير.
- Bert Mariani من كندا: طور العديد من التطبيقات بلغة رينج ويحمل لقب "سيد التطبيقات" Applications Master فى الفريق.
- م. منصور عيوني من تونس: ألف كتابا باللغة الانجليزية عن لغة رينج بعنوان Beginning Ring Programming وهو متاح على موقع أمازون للراغبين في شرائه.
- م. أحمد حسونة من مصر: قدم مئات من دروس الفيديو لتعليم لغة رينج للجمهور العربي، منشورة على قناة أكاديمية حسونة على يوتيوب.. وستجد رابطا لهذه الدروس في المجلد "مصادر تعلم رينج" داخل مجلد أمثلة الكتاب.
- د. منى أحمد كامل من مصر: تولت اختبار لغة رينج بشكل مكثف على أنظمة الماك لاكتشاف الأخطاء.

- د. مجدي سبيان من اليمن: تولى اختبار لغة رينج على نظام اللينكس بجانب المساهمة فى زيادة سرعة تطوير RingQt.
- م. أحمد سمير فايد من مصر: صمم شعار لغة رينج وأبدع التصميمات المطلوبة للعبة Gold Magic 800 التى تأتى مع اللغة.

- م. منير إدريسي من فرنسا: أدخل العديد من التحديثات على لغة رينج التى ساهمت فى زيادة الأداء Performance وهو مطور أداة VeraCrypt الشهيرة:

https://en.wikipedia.org/wiki/VeraCrypt

- م. محمد سمير فايد من مصر: قام بحجز النطاق Domain الخاص باللغة والإشراف على تجديده سنويا:

http://ring-lang.net

الصعود المستمر للغة رينج:

- فى عام ٢٠١٨ قام موقع TIOBE Index الخاص بترتيب لغات البرمجة بإدراج لغة رينج ضمن ٢٢٨ لغة برمجة يقوم الموقع بمتابعتها وترتيبها عالميا.
- وفى نفس العام صنف رينج ضمن أكثر ١٠٠ لغة برمجة شهرة عالميا، لتكون ضمن المجموعة من (٥١ الى ١٠٠) فى TIOBE Index.

- في عام ٢٠١٨ تم نشر أول لعبة بلغة رينج على موقع Steam للألعاب، وهى لعبة Goal Magic 800، وهى لعبة بسيطة استغرق تطويرها ٢٤ ساعة فقط (مقسمة على أسبوع عمل) لإظهار ملامح اللغة، وقد جذب هذا مطورين أجانب فى دار نشر Apress الأمريكية للتعرف على اللغة والرغبة في نشر كتاب عنها.
- في عام ٢٠١٩ قام مطورو ألعاب من اليابان بتقديم نسخة يابانية من موقع اللغة والتوثيق الخاص بها.

http://ring-lang-081.osdn.jp

- في عام ٢٠٢٠ قامت دار نشر Apress الامريكية بطباعة كتاب باللغة الإنجليزية عن لغة الرينج، وهو كتاب م. منصور عيوني بعنوان Beginning Ring Programming الذي أشرنا إليه سابقا.
- فى عام ٢٠٢١ أعلن م. محمود فايد عن تقدم كبير فى تطوير الجيل الجديد من تقنية البرمجة بدون كود باستخدام لغة رينج وأنه سيتوفر نسخة كاملة منه باللغة العربية. (لتفاصيل أكثر اضغط الرابط الموجود في مجلد مصادر تعلم رينج ضمن أمثلة الكتاب).

- ١ -

تنزيل وإعداد لغة رينج

تنزيل رينج Download Ring:

رينج لغة مجانية، ويمكنك الحصول عليها بسهولة من موقع اللغة (ستجد الرابط في مجلد مصادر تعلم رينج ضمن أمثلة الكتاب حيث يمكنك ضغطه مباشرة):

https://ring-lang.sourceforge.io/

عندما تدخل الموقع، اضغط الزر Downloads الموجود تحت شعار اللغة أعلى الصفحة.. سيعرض لك هذا قائمة بالمنتجات التي تستطيع تنزيلها إلى جهازك، وتشمل:

- برنامج إعداد أحدث إصدار للغة رينج (الإصدار ١,١٥ وقت صدور هذا الكتاب) وستجد منه عدة نسخ تعمل على نظم تشغيل مختلفة (ويندوز، لينوكس، ماك).
- وثائق تعليمات رينج Documentations بصيغ (PDF، CHM، HTML).

اختر النسخة المناسبة لنظام تشغيلك، فلو كنت مثلا تستخدم نظام ويندوز، فاضغط الرابط download الموجود بجوار الملف:

Ring 1.15 For Windows (32bit and 64bit)

تنبيه هام:

قد لا يعمل الإصدار ١,١٥ على ويندوز ٣٢ بت (على عكس المكتوب في عنوان الملف)، لهذا لو كنت ما زلت تستخدم ويندوز ٧ إصدار ٣٢ بت وواجهتك مشكلة في تشغيل أحدث إصدار من رينج، فعليك تنزيل نسخة أقدم قليلا، ولا تقلق فستظل قادرا على قراءة هذا الكتاب وتنفيذ ما به من أكواد.. لفعل هذا اضغط القائمة العلوية Download ومن القائمة المنسدلة اختر الإصدار ١,١٠ واتبع نفس الخطوات المشروحة هنا لتنزيله إلى جهازك.

سيؤدي ضغط رابط التنزيل إلى نقلك إلى صفحة أخرى، ستخبرك أن تنزيل الملف سيبدأ بعد ٥ ثوان، فإن لم يبدأ تلقائيا بعد انتهاء العد التنازلي (وهذا لا يحدث معي) فيمكنك ضغط الرابط الظاهر أمامك يدويا.. سيبدأ المتصفح الذي تستخدمه في تنزيل الملف وحفظه في مجلد التنزيلات Downloads على جهازك.

إعداد رينج Ring Setup:

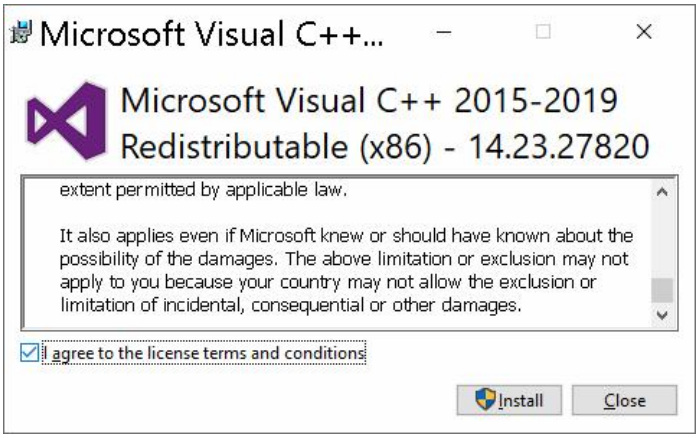
بعد انتهاء عملية التنزيل، اذهب إلى الملف الذي تم تنزيله، وهو ملف تنفيذي له الامتداد exe.، مهمته فك ضغط ملفات اللغة إلى المجلد الذي تختاره.. انقره مرتين بالفأرة، لتظهر هذه النافذة:

اضغط الزر المجاور لخانة مسار المجلد (الزر المكتوب عليه ثلاث نقاط)، لعرض نافذة اختيار المجلد.. استخدم شجرة المجلدات للوصل إلى المكان الذي تريد فك الضغط فيه، ونصيحتي أن تنشئ مجلدا جديدا في هذا الموضع لهذا الغرض.. لفعل هذا اضغط الزر Make new folder.. سيضاف مجلد جديد داخل المجلد الذي تحدده حاليا، وسيكون في وضع التحرير Edit Mode، لهذا يمكنك تغيير اسمه إلى Ring وضغط زر الإدخال Enter من لوحة المفاتيح لإنهاء التحرير.. اضغط الزر OK لإغلاق النافذة، والعودة إلى نافذة فك الضغط.. اضغط الزر Extract واصبر لحين انتهاء العملية، ثم أغلق النافذة.

والآن، اذهب إلى المجلد الذي اخترته.. ستجد فيه ما يلي:

١. الملف readme.txt وفيه معلومات وإرشادات حول اللغة.

٢. الملف VC_redist.x86.exe وهو يعد الملفات اللازمة لتشغيل بعض مكونات لغة رينج المكتوبة بلغة Visual C++.. انقره مرتين بالفأرة لتشغيله، وضع علامة الاختيار في مربع الموافقة على تعليمات الترخيص واضغط الزر Install، وواصل خطوات الإعداد.



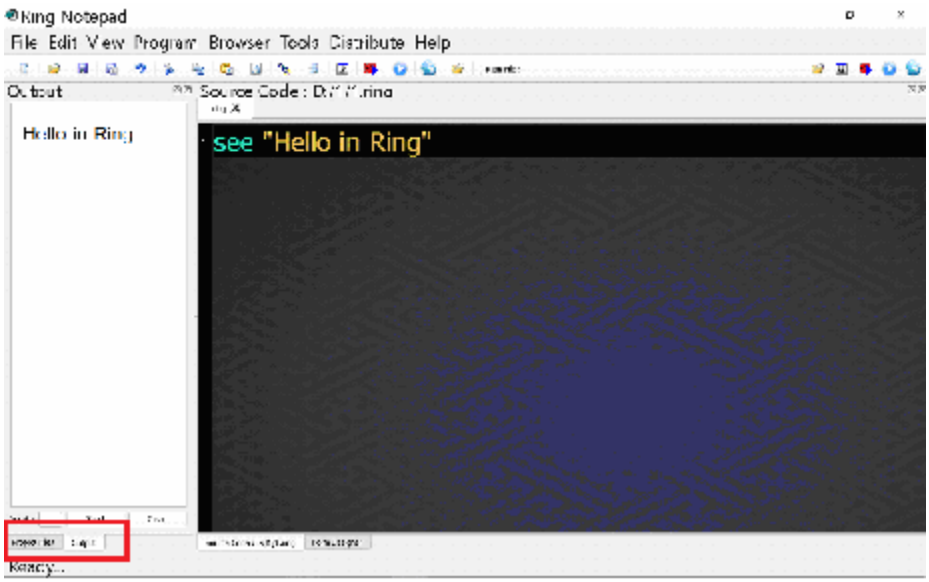
لاحظ أن جهازك قد يحتوي بالفعل على هذه الملفات، وفي هذه الحالة سيخبرك برنامج الإعداد أنك لا تحتاج إلى إعدادها، ومن ثم يمكنك إغلاقه.

٣. المجلد Ring وهو يحتوي على مكونات لغة رينج، وبعض الأمثلة والبرامج المكتوبة باللغة (في المجلدين Applications و Samples).. لكن ما يهمنا الآن هو الملف RingNotepad.exe (لو أعددت إصدارا قديما من رينج فستجد اسمه RNote.exe).. لو نقرت هذا الملف مرتين بالفأرة، فستظهر لك بيئة تطوير رينج.

مرحبا بك في لغة رينج!

محرر كود رينج Ring Code Editor:

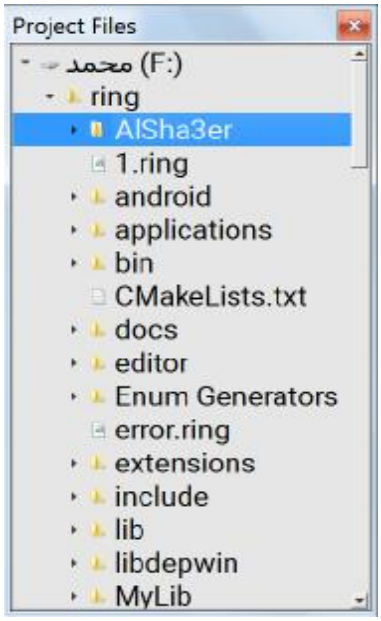
عندما تفتح دفتر ملاحظات رينج RingNotepad، ستكون عمليا داخل بيئة تطوير رينج Ring IDE، حيث يمكنك كتابة الكود وتشغيله وتحويله إلى ملف تنفيذي قابل للتوزيع لو أردت.. والصورة توضح لك الشاشة التي ستراها عند تشغيل هذا البرنامج:



المنطقة الزرقاء (التي تراها في الصورة سوداء) هي نافذة الكود التي نكتب فيها كود رينج، أما على اليسار (أو اليمين في الإصدارات القديمة) فترى نافذة المخرجات Output، وهي التي سنعرض فيها بعض النتائج ونحن نختبر الأكواد التي نكتبها.. ويمكنك سحب هذه النافذة لعرضها في اليمن أو اليسار أو عائمة في أي موضع من الشاشة، حيث يمكنك أن تنقر مرتين بالفأرة فوق الشريط العلوي للنافذة لإعادتها إلى وضعها الأصلي.. وهذا ينطبق أيضا على باقي النوافذ.

ولو دققت في الصورة قليلا، فستجد شريطين أسفل نافذة المخرجات (وضعت حولهما مستطيلا لتمييزهما)، أحدهما هو شريط نافذة المخرجات Output والآخر هو شريط نافذة ملفات البرنامج Project Files، ولو ضغطت الشريط الأخير فسيعرض لك

شجرة المجلدات الموجودة على جهازك، لتختار منها المجلد الذي تريد حفظ ملفات مشروعك فيه، ليصير المجلد الافتراضي الذي تعرضه لك نافذة حفظ الملف عندما تحاول حفظ أي ملف جديد، لكن هذا لن يمنعك حينها من اختيار أي مكان آخر لحفظ الملف لو أردت.. هذا فقط تسهيل لتسريع حفظ كل ملفات مشروعك في مكان واحد.. لكن هناك عيب في هذه النافذة فهي لا تسمح بإنشاء مجلدات جديدة، لهذا عليك أن تنشئ المجلد بنفسك أولا من خلال متصفح الملفات الخاص بنظام تشغيلك مثل Windows File Explorer، ثم تختاره في نافذة الملفات الخاصة بمحرر رينج.



بالنسبة لمشروعنا، سنضع كل ملفات برنامج الشاعر في مجلد اسمه AlSha3er (لاحظ أن الرقم 3 هو بديل لحرف العين العربي).. أنشئ هذا المجلد في الموضع الذي تريده على جهازك، ثم اختره في نافذة ملفات رينج، وأغلقها لو أردت.

تحذير:
يجب أن تسمي الملفات والمجلدات الخاصة ببرامج رينج بأسماء أجنبية لا تتخللها مسافات ولا رموز، وأن تحفظ هذه الملفات في مسار تنطبق على كل مجلداته نفس الشروط.. لهذا مثلا لو حفظت الملف في مجلد يحتوي اسمه على حروف عربية، فلن يعمل البرنامج وستعطيك رينج رسالة خطأ تقول إنها لا تستطيع العثور على الملف!.. ومن المتوقع إصلاح هذا الأمر في إصدار تال من رينج بإذن الله.

ملحوظة:

لكل نافذة من النوافذ السابقة زر x يمكنك ضغطه لإغلاقها.. ولإعادة فتح النافذة مجددا، اضغط القائمة الرئيسية View من شريط القوائم الموجود تحت شريط عنوان نافذة RingNotePad، واضغط اسم النافذة التي تريد عرضها (أو إخفاءها لو كانت معروضة).. سترى في القائمة View العديد من أسماء النوافذ ومن بينها:

- Project Files: عرض نافذة ملفات المشروع.
- Source code: عرض نافذة محرر الكود.
- Output window: عرض نافذة المخرجات.

نحن الآن جاهزون للشروع في إنشاء برنامج الشاعر.

-٢-

التعامل مع النصوص العربية في رينج

النصوص Strings:

حينما نكتب برنامجا لوزن الشعر العربي، فالطبيعي أن نتوقع أن يدخل المستخدم للبرنامج بعض أبيات من الشعر.. هذا يعني أن البيانات التي سيتعامل معها برنامجنا هي النصوص العربية Arabic Texts.

والنص هو سلسلة من الحروف String of characters، ولهذا تستخدم لغات البرمجة الاسم المختصر String للإشارة إلى المتغير الذي يستطيع تخزين النصوص في ذاكرة الحاسوب وإجراء العمليات عليها.

والمثال التالي يريك تعريف متغير اسمه sSample في لغة رينج، ووضع نص فيه:

sSample = "Test"

في هذا الكود نلاحظ ما يلي:

- النص sSample هو اسم متغير Variable Name، بينما النص Test هو قيمة نصية String literal لأنه موضوع بين علامتي تنصيص ""..ولا تعتبر علامتي التنصيص جزءا من النص وإنما جزء من صياغة لغة البرمجة، يخبرها أن ما بين علامتي التنصيص هو قيمة نصية.
- يستخدم الرمز = في معظم لغات البرمجة لوضع القيم في المتغيرات وهو ما يسمى بالتعيين Assignment.. إذن فالجملة السابقة ليست معادلة رياضية، والرمز = هنا لا يعني "يساوي"، بل يعني "سوف يساوي"، لأن المتغير الموجود في الطرف الأيسر للعملية، سيساوي القيمة الموجودة في الطرف الأيمن بعد تنفيذ هذا الكود.. فالتساوي هنا باعتبار ما سيكون، وهذا هو المنطق وراء استخدام الرمز = .

- لم نحدد هنا نوع البيانات التي سيتعامل معها المتغير sSample.. هذا لأن لغة رينج مرنة الأنواع Dynamic Typing، أي أنك تستطيع وضع أي نوع من البيانات في نفس المتغير، وستختار اللغة النوع المناسب للقيمة وتحجزه في الذاكرة وتجعل المتغير يشير إليه.. إذن يمكنك أن تكتب:

```
sSample = "Test"
sSample = 5
sSample = new QString("text")
```

بعد تنفيذ السطر الأول سيكون المتغير sSample من النوع String وفيه النص Test، وبعد تنفيذ السطر الثاني سيصير من النوع Number وفيه الرقم ٥، وبعد تنفيذ السطر الثالث سيصير كائنا من النوع QString (الذي سنتعرف عليه بعد قليل) وفيه نسخة جديدة من هذا النوع.. هذه المرونة تسهّل عليك تعريف المتغيرات وتختصر كتابة الكود.

- حتى لا تربكك النقطة السابقة أثناء كتابة البرنامج أو مراجعة الكود، أنصحك بألا تستخدم نفس المتغير للتعامل مع أنواع مختلفة من البيانات (إلا إن كانت هناك ضرورة ملحة تجبرك على هذا)، وأن تضيف بادئة لاسم المتغير توضح نوع القيم التي ستضعها فيه.. فكما تلاحظ مثلا في المتغير sSample، استخدمت البادئة s للإشارة إلى أنني سأضع فيه نصوصا.. وإذا أردت تعريف متغير رقمي، فسمه مثلا nBooks.. وهكذا.

ولا يوجد ما يمنعك من وضع النصوص العربية في المتغيرات.. هكذا مثلا يمكنك تعريف متغير اسمه sBayt، ووضع بيت من الشعر فيه:

```
sBayt = "قالتْ لي أحلامُ شبابي: أزْهر للكونِ رياحينَكْ"
```

لكن ماذا لو أردت أن تضع علامة تنصيص داخل النص؟

لحل هذه المشكلة تسمح لك رينج أيضا بوضع النص بين علامتي تنصيص فرديتين Single quotes.. علامة التنصيص الفردية هي الرمز الإنجليزي ' (الذي يسمى Apostrophe ونستخدمه في كلمات مثل I'm).. وفي هذه الحالة يمكنك استخدام علامة التنصيص المزدوجة داخل النص:

sBayt = '"قالتْ لي أحلامُ شبابي: "أزْهر للكونِ رياحينَكْ"'

والعكس أيضا متاح، أي أنك تستطيع استخدام علامة التنصيص المفردة داخل النص، إذا وضعت النص كله بين علامتي تنصيص مزدوجتين.. هذا لن يكون ملائما للنصوص العربية، ولكنه مقبول في النصوص الإنجليزية:

sSample = "I'm a programmer."

قراءة حروف النص:

النص كما ذكرنا هو سلسلة من الحروف.. هذه السلسلة تسمى في البرمجة مصفوفة Array أو قائمة عناصر List (سنستخدم المصطلحين في هذا الكتاب للإشارة إلى نفس المعنى).. ويمكن التعامل مع كل عنصر في المصفوفة أو القائمة من خلال رقمه في ترتيب العناصر.. هناك لغات برمجة مثل VB.NET و C# يبدأ ترقيم العناصر فيها بالرقم صفر، وهناك لغات أخرى مثل VB6 و Small Basic يبدأ الترقيم فيها بالرقم ١.. وتنتمي لغة رينج إلى النوع الأخير، حيث يبدأ ترقيم أول حرف في النص أو في أي قائمة بالرقم ١.

ويمكنك الإشارة إلى رقم الحرف، بوضعه بين قوسين مضلعين بعد اسم المتغير.. والمثال التالي يعرض على الشاشة الحرف الثاني في النص sSample:

```
sSample = "Test"
? sSample[2]  #e
```

لاحظ أن علامة الاستفهام تطلب من رينج كتابة الناتج التالي لها في نافذة المخرجات ثم كتابة سطر جديد.

أما الرمز # فيخبر رينج أن ما يأتي بعده هو مجرد تعليق Comment، وليس أمرا برمجيا، وهذا يجعل رينج تتجاهله كأنه غير موجود.. التعليق هو مجرد ملاحظة لتذكير المبرمج بشيء.. وقد وضعت التعليق هنا لأخبرك بالناتج المتوقع أن تراه على الشاشة، وهو الحرف e.. يمكنك إزالة التعليق وأنت تجرب المثال، ولن يختلف شيء.. كما يمكنني أن أجعل الأمور أوضح لك بكتابة تعليق أطول:

```
? sSample[2]  # result: e
```

تشغيل البرنامج Run:

لتنفيذ الكود الذي تكتبه في محرر الكود، استخدم أيا مما يلي:

- من القائمة الرئيسية Program اضغط الأمر Run.
- من لوحة المفاتيح اضغط زر التحكم مع الزر R.. يمكنك معرفة اختصار كل أمر برؤية النص المكتوب في أوامر القوائم، فالأمر Run مثلا مكتوب بجواره CTRL+R.
- من شريط الأدوات الموجود أسفل القوائم الرئيسية، اضغط زر التشغيل

الآن، دعنا نجرب نفس المثال مع نص عربي:

```
sSample ="محمود"
? sSample[2]
```

لاحظ أن مصفوفة الحروف التي تمثل النص لا تختلف في شيء عند وضع حروف لاتينية أو عربية فيها.. في كل الأحوال يوضع أول حرف في النص في الخانة رقم ١، ولا يهم هنا إن كان النص يظهر على الشاشة من اليمين إلى اليسار أو من اليسار إلى اليمين، فنظام الويندوز هو المسئول عن طريقة عرض النص ولا علاقة للغة البرمجة بهذا.. وهذا معناه أن الحرف الثاني في النص محمود هو ح كما هو متوقع.

لكنك لو نفذت المثال السابق، فسيظهر على الشاشة هذا الرمز الغريب ◆ بدلا من حرف الحاء.. هذا الرمز يدل على أن الحرف غير قابل للكتابة (ليس حرفا أبجديا ولا رقما ولا علامة ترقيم ولا مسافة).

ستسألني: وهل يوجد في النصوص شيء غير الحروف الأبجدية والأرقام والمسافات؟

نعم.. هناك بعض الحروف الوظيفية التي لا يمكن كتابتها.. مثلا: هناك رمز لكل زر على لوحة المفاتيح، وأنت تعرف أن بعض الأزرار تنفذ وظائف ولا تكتب شيئا مثل زر الحذف Delete والمسافة الخلفية BackSpace وزر الهروب (الإلغاء) Escape.

جميل.. لكن ما علاقة الرموز الوظيفية بالنص محمود، ولماذا لم يظهر حرف الحاء؟

لنفهم الإجابة عن هذا، يجب أن نتعرف أولا على الترميز القياسي ASCII والترميز العالمي Unicode.

ترميز الحروف Encoding:

في بداية اختراع الحاسوب في أمريكا، كانوا مهتمين فقط بالحروف الإنجليزية دون باقي لغات العالم.. لهذا مثلوا الحروف رقميا بترميز قياسي اسمه ASCII وهو يستوعب ٢٥٦ حرفا فقط تشمل الأرقام 9-0 والحروف الإنجليزية الصغيرة a-z والحروف الإنجليزية الكبيرة A-Z، إضافة إلى المسافات وعلامات السطر الجديد وعلامات الترقيم وبعض الرموز الوظيفية.. هذا الترميز يستخدم بايت واحد فقط لحفظ كل حرف.. والبايت Byte هو وحدة التخزين المستخدمة في الذاكرة ووسائط التخزين الأخرى مثل القرص الصلب.

لكن بعد هذا، ظهرت الحاجة لتمثيل باقي لغات دول العالم، لهذا تم إنشاء الترميز العالمي Unicode الذي يستوعب آلاف الحروف الأبجدية والرموز الرياضية والكيميائية وغيرها.. هذا الترميز يستخدم وحدتي ذاكرة 2-Bytes لحفظ كل حرف.

مع ملاحظة أن هناك أنواعا أخر للترميز مثل Utf-7 وغيرها.

إذن وكما تتوقع، تنتمي الحروف العربية للترميز العالمي، ويحتاج كل حرف إلى ٢ بايت لحفظه في الذاكرة.. لكن النص في رينج مكتوب لحفظ كل حرف في بايت واحد فقط.. لهذا عند وضع النص محمد في المتغير sSample، سيحتوي هذا المتغير على ٨ حروف بدلا من ٤، لأن كل حرف عربي سيخزن في الذاكرة في خانتين متتاليتين بترميز Unicode، لكن رينج تظن أنهما حرفان مختلفان مكتوبان بترميز ASCII.. لهذا حينما طلبنا عرض الحرف الثاني من النص محمود، قرأت رينج الخانة الثانية التي هي جزء من تمثيل الحرف الأول!

لكي تتأكد من هذا، دعنا نعرض عدد حروف النص باستخدام الدالة Len:

```
sSample ="محمود"
? Len(sSample)
```

سترى على الشاشة العدد ١٠، مع أن الكلمة محمود تتكون من ٥ حروف فقط!

فما هو الحل إذن؟

استخدام الفئة QString:

تدعم لغة رينج مكتبة رسومية Graphics Library اسمها Qt، ويوجد ضمن هذه المكتبة فئة اسمها QString (وهناك نسخة معدلة منها اسمها QString2).. هذه الفئة تسمح لك بالتعامل مع النصوص الممثلة بالترميز العالمي، وهذا معناه أنها تصلح للتعامل مع اللغة العربية.

الفئة Class والكائن Object:
إذا اعتبرت الإنسان فئة Class، فأنا وأنت وعلي وسميرة نعتبر كائنات Objects تنتمي لفئة الإنسان، ونمتلك جميعا نفس وظائف فئة الإنسان مثل يأكل ويشرب ويمشي وينام.. إلخ.. كما نمتلك خصائص فئة الإنسان مثل الاسم والعمر والطول واللون.. إلخ، لكن يظل لكل كائن منا قيم مختلفة لهذه الخصائص، فاسمي مختلف عن اسمك، وطولك مختلف عن طول علي.. وهكذا. وتسمى البرمجة التي تعتمد على الفئات بالبرمجة الموجهة بالكائنات Object Oriented Programming أو اختصارا OOP.

وللتعامل مع الفئة QString، عليك تحميل المكتبة guilib.ring، بإضافة السطر التالي في بداية ملف الكود:

```
Load "guilib.ring"
```

الآن يمكننا إعادة كتابة المثال السابق كالتالي:

```
qsSample = new QString( )
qsSample.append("محمود")
? qsSample.mid(1, 1) #ح
? qsSample.Count() #5
```

دعنا نفهم هذا الكود:

في السطر الأول استخدمنا الكلمة new لإنشاء نسخة جديدة (كائن) من فئة النص QString، ووضعناها في المتغير qsSample.

الآن صار لدينا نص لكنه فارغ.. ولإضافة نص يمكننا استدعاء دالة الإلحاق

Append لإضافة نص جديد إلى نهاية النص الموجود حاليا.. وقد أرسلنا هذا النص الجديد إلى الدالة Append بوضعه بين قوسين ( ) بعد اسم الدالة.

وللأسف لن نستطيع استخدام القوسين المضلعين [ ] مع الفئة QString للوصول إلى الحرف عن طريقة رقمه، فهذه الفئة لا تدعم هذا، لكنها تحتوي على دالة اسمها Mid (اختصار للكلمة Middle أي "منتصف") التي يمكننا استخدامها لقراءة حرف أو أكثر من أي موضع من النص.. وتستقبل هذه الدالة معاملين:

- الأول يستقبل موضع أول حرف تريد قراءته.

- والثاني يستقبل عدد الحروف التي تريد قراءتها، وسنرسل إليه ١ لنقرأ حرفا واحدا.

لكن لماذا أرسلنا العدد ١ إلى المعامل الأول، ونحن نريد الحرف الثاني في النص؟

السبب هو أن المكتبة Qt ليست خاصة بلغة رينج ولكنها مكتوبة بلغة C++، لهذا يبدأ ترقيم الحروف فيها من الصفر.. ولهذا سيكون للحرف الثاني الترقيم ١ لأن الحرف الأول هو الحرف رقم صفر.

لو جربت الكود الجديد، فسيظهر الحرف "ح" في نافذة المخرجات كما أردنا.

أما السطر الأخير، فقد استخدمنا فيه الوسيلة Count لحساب عدد حروف النص، وستعيد لنا هنا العدد ٥ وهو الطول الصحيح للنص العربي.

ملحوظة١:
رينج غير حساسة لحالة الأحرف، فالحروف الصغيرة Small تعامل معاملة الحروف الكبيرة Capital، ولا فارق بين QString و qstring وبين Append و append.

ملحوظة٢:
مكتبة Qt لا تعمل بشكل جيد على ويندوز ٣٢ بت، لهذا لو كنت تتعامل مع هذا النظام بإصدار قديم من رينج، فقد تحدث معك مشاكل في تجربة الأكواد.. لهذا السبب ستجد أنني بنيت الفئة aString المرفقة بأمثلة الكتاب بطريقة أخرى لم أستخدم فيها الفئة QString، حتى يمكنها أن تعمل على جميع نسخ الويندوز.

إنشاء فئة النصوص العربية aString:

في برنامج الشاعر، سنحتاج كثيرا للتعامل مع الحروف العربية، وسيكون متعبا أن نتعامل مع الفئة QString بهذه الطريقة مرارا وتكرارا لنكتب في كل مرة عدة سطور لتنفيذ شيء بسيط.. ناهيك عن الارتباك الذي سيصيبنا بسبب اختلاف قاعدة ترقيم أول حرف ما بين رينج و QString.. لكل هذا من الأفضل أن ننشئ فئة خاصة بنا نخصصها للتعامل مع الحروف العربية بالشكل الذي يريحنا، ونجعل أول حرف في النص فيها يأخذ الترقيم ١، كما سنضيف إليها بعض الخصائص والوسائل بأسماء مألوفة أكثر، مثل الخاصية Length التي تعيد طول النص بدلا من الوسيلة Count.

من القائمة الرئيسية File اضغط الأمر New لفتح ملف جديد في محرر كود رينج.. ستظهر نافذة حفظ ملف، وستعرض تلقائيا محتويات مجلد برنامج الشاعر لأننا جعلناه مجلد ملفات المشروع.. اكتب الاسم aString.ring في خانة اسم الملف، واضغط الزر Save لحفظه في المجلد وإغلاق النافذة.. ستجد هذا الملف مفتوحا في محرر الكود لكنه فارغ طبعا.. أضف إليه الكود التالي:

```
Class aString
  Text
  Length
```

بهذه السطور الثلاثة فقط صار لدينا فئة اسمها aString لها خاصيتان هما Text و Length.. لاحظ ما يلي:

- تستخدم الكلمة class لتعريف الفئة، ويتبعها اسم الفئة.
- يمكنك لو أردت أن تستخدم قوسين مضلعين لتغليف محتويات الفئة:

```
Class aString {
  Text
  Length
}
```

أو يمكنك استخدام الكلمة EndClass في نهاية مقطع الفئة:

```
Class aString
  Text
  Length
EndClass
```

- يجب أن يكون تعريف الفئة آخر شيء في الملف.. أي أنك تستطيع إضافة أي كود قبلها (خارجها) لكن لا تستطيع إضافة أي كود بعدها في نفس الملف، إلا إذا كان فئة أخرى.. هذا هو السبب الذي يتيح لك عدم استخدام القوسين {} أو EndClass لتمييز مقطع الدالة، فحينما ترى رينج الكلمة Class في بداية الفئة، تعرف أن كل ما يليها ينتمي إلى مقطع الفئة إلى أن تصل إلى الكلمة Class في تعريف فئة تالية أو تصل إلى نهاية الملف.. لكني أنصك دائما بوضع نهاية لتعريف الفئة منعا للارتباك، سواء باستخدام {} أو EndClass.
- يتم تعريف الخصائص (وتسمى أيضا السمات Attributes) بعد اسم الفئة مباشرة.. بعد ذلك يمكن تعريف الدوال.. لكن لو أضفت أي خصائص بعد الدوال فسيتم تجاهلها كأنها غير موجودة!.. إذن: الخصائص أولا ثم الدوال لاحقا.
- يمكنك أن تضع قيما افتراضية في أي خاصية في سطر تعريفها، مثل:

Text = ""

حيث وضعنا نصا فارغا (علامتي تنصيص ليس بينها أي شيء) في الخاصية.

- تذكر أن المتغيرات والخصائص في رينج مرنة ولا تأخذ أنواعا.. لهذا نكتب اسم الخاصية وحده بدون نوع، ويمكننا أن نضع فيها أي شيء نريده، فلا يوجد ما يمنعنا من وضع نص في الخاصية Length مثلا.. لن نفعل هذا طبعا لأنه سيفسد وظيفة الخاصية.. في الحقيقة يجب ألاّ نضع أي شيء في هذه الخاصية حتى لو كان رقما، لأنها تعيد إلينا طول النص ولا يصح تغييره.. سنرى كيف سنحل هذه المشكلة بعد قليل.

يمكننا الآن تجربة هذه الفئة البسيطة.. افتح ملفا جديدا واحفظه بالاسم aString.Tests.ring، واكتب فيه:

```
Load "aString.ring"

s1 = new aString
s1.Text ="محمد"
s1.Length = 4
? s1.Text + " " + s1.Length
```

لو نفذت الكود، فستعرض نافذة المخرجات:

محمد ٤

كما ترى، يحمل المتغير s1 كائنا من الفئة aString، لهذا استطعنا وضع قيم في الخاصتين Text و Length لهذا الكائن، ثم قرأنا قيمتيهما وعرضناهما على الشاشة. وكما ذكرنا، تستطيع تعريف أي عدد تريده من الكائنات من نفس الفئة، حيث سيكون لكل كائن قيم مستقلة لخصائصه.. أضف هذا الكود بعد الكود السابق:

```
s2 = new aString
s2.Text ="محمود"
s2.Length = 5
? s2.Text + " " + s2.Length
```

لو نفذت الكود، فسيظهر على الشاشة:

محمد ٤

محمود ٥

جميل.. فلنرجع الآن إلى ملف الفئة.. لاحظ وجود شرائط Tabs فوق نافذة الكود، تسمح لك بالانتقال بين الملفات المفتوحة.. اضغط الشريط المكتوب عليه aString.ring للانتقال إلى صفحة الفئة.

لاحظ أن الفئة الخاصة بنا تتعامل مع النصوص العادية حتى هذه اللحظة، فالنص الذي وضعناه في الخاصية Text هو نص عادي من نصوص رينج، لهذا يجب أن نجد طريقة لاستخدام QString داخل هذه الفئة.

لفعل هذا، سنقوم بتعريف متغير داخل الفئة الخاصة بنا وليكن اسمه qstr لنستخدمه كمخزن داخليّ يحمل النصوص الخاصة بنا، لكنها ستكون من النوع QString.

كل ما نحتاجه هو إضافة هذا السطر داخل الفئة (بعد تعريف الخصائص):

```
qstr = new QString2( )
```

لكن لاحظ أن هذا سيجعل qstr خاصية جديدة، ونحن نريدها مجرد متغير داخلي خاص بالفئة لا يمكن استخدامه من خارجها.. فما الحل؟

لو أردت تعريف متغيرات ودوال خاصة بالفئة فقط، فيمكنك تعريفها داخل مقطع العناصر الخاصة Private.. هذا المقطع يجب أن يكون في نهاية الفئة، ويبدأ تعريفه

بالكلمة Private.. هذا معناه أن الفئة تتكون من ٣ مقاطع هي بالترتيب:

- الخصائص Properties (وتسمى أيضا السمات Attributes).
- الدوال Functions (وتسمى أيضا الوسائل Methods).
- المقطع Private ويحتوي على بعض الخصائص والدوال التي يمكن استخدامها داخل الفئة فقط.. لاحظ أيضا أن تعريف الخصائص في القسم الخاص يجب أن يسبق تعريف الدوال.. الترتيب في رينج هو كل شيء، وهذا يمنحك حرية استخدام أو عدم استخدام القوسين المتعرجين { } لتمييز نهاية المقطع Private.

الآن يجب أن يبدو كود الفئة كالتالي:

```
Class aString
  Text
  Length

  private
    qstr = new QString2( )
EndClass
```

الآن علينا ربط الخاصيتين Text و Length بالمتغير qstr، بحيث تعبر الخاصية Text عن النص الموجود فيه، وتعبر الخاصية Length عن طول هذا النص.. لفعل هذا، يجب أن نستخدم دالتين:

- دالة الكتابة Setter:

يبدأ اسم هذه الدالة بالكلمة Set متبوعة باسم الخاصية (مثل SetText و SetLength)، ويكون لها معامل واحد يحمل القيمة المراد وضعها في الخاصية، وعليك أنت أن تضعها بنفسك في الخاصية.

وتستدعي رينج هذه الدالة تلقائيا عند وضع قيمة في الخاصية من كود مكتوب خارج الفئة (مثل الكود الذي كتبناه في الملف aString.Tests.ring)، لكن رينج لا تستدعيها إذا وضعت أنت قيمة في الخاصية من داخل الفئة نفسها، لكنك تستطيع استدعاءها بنفسك إذا احتجت لهذا، بنفس طريقة استدعاء أي دالة عادية (بكتابة اسمها وبعده قوسين كما سنرى بعد قليل).

- ودالة القراءة Getter:

يبـدأ اسـم هـذه الدالــة بالكلمــة Get متبوعــة باسـم الخاصـية (مثـل GetText و GetLength)، وليس لها أي معاملات، لكنها يجب أن تعيد قيمة الخاصية. وتستدعي رينج هذه الدالة تلقائيا عند قراءة قيمة الخاصية من كود مكتوب خارج الفئة، لكن رينج لا تستدعيها إذا قرأت أنت قيمة الخاصية من داخل الفئة نفسها، لكن يمكنك استدعاؤها بنفسك كما تستدعي أي دالة عادية أخرى.

إذن فكل المطلوب عند وضـع نص فـي الخاصية Text أن نستخدم الدالـة SetText لوضع هذا النص في المتغير qstr:

```
func SetText(str)
    qstr.clear( )
    qstr.append(str)
```

لاحظ ما يلي:

- تستخدم الكلمـة func اتعريـف الدالـة متبوعـة باسم الدالـة، ثم قوسين ( ).. يمكن تـرك هـذين القوسـين فـارغين، أو تعريـف معـاملات بينهمـا (معامـل واحـد أو أكثـر مفصولة بالعلامة ,).. وفي حالتنا هذه لدينا معامل اسمه str (يمكنك تسميته بأي اسم آخر كما تشاء)، سيستقبل النص المراد وضعه في الخاصية.
- يمكن تغليف محتوى الدالة بقوسين متعرجين {} كما هو متبع في عائلة لغات C، أو بطريقة بديلة، يمكن ختام مقطع الدالة بالكلمة EndFunc وهذا شبيه بالمتبع في عائلة لغات البيزيك، أو يمكنك تركها كما تراها في المثال حيث تستطيع رينج ببراعة استنتاج أن الدالة قد انتهت عندما تجد دالة أخرى تالية لها أو تجد الكلمة private في بداية مقطع العناصر الخاصة.. لكني أنصحك دائمة بتغليف الدالة بالقوسين المتعرجين أو EndFunc منعا للارتباك.
- قبل وضع النص في المتغير qstr يجب أن نمحو أي نص موجود فيه حاليا.. ولهذا استدعينا الدالة Clear الخاصة بالفئة QString.
- بعد هذا استدعينا دالة الإلحاق Append وهي تضيف النص المرسل إليها إلى

نهاية النص الموجود في الفئة QString.. لهذا أفرغنا المتغير qstr أولا حتى لا يضاف النص الجديد (الموجود في المعامل str) إلى نهاية النص الذي كان موجودا سابقا في المتغير qstr:

```
qstr.clear( )
qstr.append(str)
```

أما عند قراءة قيمة الخاصية Text، فسنستخدم الدالة GetText لنعيد النص الموجود في المتغير qstr:

```
Func GetText( )
   return qstr.mid(0, qstr.count( ))
EndFunc
```

لاحظ ما يلي:

- نستخدم الأمر return لإعادة ناتج الدالة لمن يستدعيها.. بعد هذه الكلمة نكتب أي قيمة نريدها أو أي كود يحسب قيمة.. وينتهي تنفيذ كود الدالة عند وصولها إلى Return، ولو كتبت أي كود في السطور التالية للأمر Return فلن ينفذ.
- للحصول على النص الموجود في الفئة QString استخدمنا الوسيلة mid، وأرسلنا إليها موضع أول حرف في النص (الموضع رقم صفر) وعدد حروف النص كله (حصلنا عليه باستخدام الدالة Count).

بالمثل، سنستخدم الدالة GetLength لنعيد طول النص الموجود في المتغير qstr:

```
func GetLength( )
    return qstr.count( )
```

أما عند محاولة وضع قيمة في الخاصية Length فسنعتبر هذا خطأ غير مسموح به:

```
func SetLength(str)
   Raise("'Length' is a read-only property.")
```

لاحظ استخدامنا للدالة Raise لإطلاق الخطأ، وهي تنهي عمل البرنامج في الحال، وتعرض على نافذة المخرجات رسالة الخطأ التي أرسلناها إليها.. ويمكنك أن ترسل الرسالة بالعربية لو أردت:

```
Raise("هذه الخاصية للقراءة فقط.")
```

وهناك حيلة جميلة تجعل رينج نفسها ترفض وضع أي قيمة في الخاصية بدون حتى استخدام الأمر Raise، وذلك بعدم تعريف متغير للدالة SetLength، وبالتالي عندما يضع أحد قيمة في الخاصية Length فستحاول رينج إرسال هذه القيمة إلى الدالة SetLength لكنها ستجدها بدون معاملات، فتنهي البرنامج وتعرض رسالة الخطأ التالية:

Calling function with extra number of parameters In method setlength()

أي أنك تحاول استدعاء دالة بعدد معاملات زائد عما تحتاجه الدالة!

إذن، يمكننا إعادة تعريف الدالة SetLength هكذا ببساطة في سطر واحد فقط:

```
Func SetLength( ) #ReadOnly
```

أضفت التعليق #ReadOnly ليذكرك بفائدة هذا السطر حينما تقرؤه في المستقبل.

هكذا يجب أن يكون كود الفئة الآن:

```
Class aString
  Text
  Length

  Func SetText(str)
    qstr.clear( )
    qstr.append(str)
  EndFunc

  Func GetText( )
    return qstr.mid(0, qstr.count( ))
  EndFunc

  func SetLength( ) #ReadOnly

  func GetLength( )
      return qstr.count( )
  EndFunc

  private
    qstr = new QString2( )

EndClass
```

تعال نجربها.. انتقل إلى صفحة الاختبار aString.Test.ring، واضغط زر تشغيل البرنامج لتنفيذ الكود الذي كتبناه سابقا.. ستلاحظ ظهور رسالة خطأ في قسم المخرجات تخبرك أن خاصية الطول للقراءة فقط:

```
Line 5 Error (R20) : Calling function with extra number of parameters!
In method setlength() in file aString.ring
called from line 5 in file astring,test.ring
```

من المهم أن نفهم رسائل الخطأ التي تعرضها لنا رينج، لأنها هي الوسيلة المتاحة لتصحيح البرنامج.. تعرض الرسالة البيانات التالية:

- رقم السطر الذي حدث فيه الخطأ، وهو هنا السطر رقم ٥
- رسالة الخطأ الذي حدث وهي هنا وجود معامل زائد.
- تسلسل الدوال الذي تسبب في حدوث الخطأ، وهو هنا الدالة setlength في الملف aString.ring، والتي تم استدعاؤها من السطر الخامس في الملف astring,test.ring.

واضح بالنسبة لنا أن المشكلة ليست في الملف aString.ring، لكن المشكلة الحقيقة تكمن في السطر الخامس في ملف الاختبار.. علينا إذن الانتقال إلى هذا السطر، وهو أمر سهل لأن محرر رينج يعرض الرقيم في بداية كل سطر، كما أن هناك تسهيلا آخر لو كان الملف طويلا، حيث يمكنك ضغط الأمر Go to line من القائمة الرئيسية Edit (أو ضغط Ctrl+G مباشرة من لوحة المفاتيح) وكتابة رقم السطر في النافذة التي ستظهر لك وضغط زر الموافقة للانتقال إلى السطر مباشرة.. هذا هو السطر الذي حدثت فيه المشكلة:

s1.Length = 4

وهو يحاول وضع قيمة في الخاصية Length التي جعلناها للقراءة فقط.

احذف هذا السطر، واضغط Ctrl+F5 لتنفيذ الكود.. سيحدث نفس الخطأ مرة أخرى، لكن هذه المرة عند السطر:

s2.Length = 5

لهذا عليك حذفه هو أيضا.. لو جربت البرنامج الآن فلن تحدث أخطاء، وستعرض

نافذة المخرجات:

محمد ٤

محمود ٥

إذن فالفئة aString صارت قادرة على التعامل مع الحروف العربية بشكل صحيح.

اختبارات الكود Tests:

لا يفهم الحاسوب شيئا من الكود الذي كتبته بلغة رينج (أو أي لغة برمجة أخرى).. اللغة الوحيدة التي يفهمها الحاسوب هي لغة الآلة Machine Language وهي تتكون من مجموعة كبيرة من الأصفار والآحاد يصعب علينا فهمها.. ولتنفيذ البرنامج، يجب ترجمته من لغة رينج إلى لغة الآلة، وهذا ما يفعله جزء من لغة البرمجة يسمى بالمترجم Compiler، وهو يفحص الكود لاكتشاف الأخطاء المحتملة قبل ترجمته، لهذا لو وجد أي خطأ في الصياغة فسينبهك له.. لكن الأخطاء الأخرى الناتجة عن خلل في وظيفة الكود لا يمكن اكتشافها إلا أثناء تنفيذ البرنامج.. لهذا لم تكتشف Ring تكرار خطأ وضع قيمة في الخاصية Length إلا بعد تنفيذ البرنامج مرة أخرى.

الأخطر من هذا أن رينج هي لغة برمجة مرنة Dynamic Language، وهذا معناه أنها تترك لك حرية نقل القيم بين المتغيرات، واستدعاء خصائص تنتمي لفئات معينة على مسئوليتك.. وكمثال، في هذا السطر:

```
Func GetLength( )
  return qstr.count( )
EndFunc
```

نحن نتعامل مع المتغير qstr باعتباره كائنا من نوع الفئة QString لهذا استدعينا الوسيلة Count التابعة لها.. لكن لا يوجد في رينج ما يمنعك من وضع أي قيمة في المتغير qstr في أي لحظة، مثل:

```
Func GetLength ( )
  qstr = 12
  return qstr.count( )
EndFunc
```

واضح أن هذا سيؤدي إلى خطأ في البرنامج، لأن العدد ١٢ لا يملك وسيلة اسمها Count، لكن للأسف لن يستطيع مترجم رينج اكتشاف هذا الخطأ إلا عند حدوثه في وقت التنفيذ، والأخطر أنك لو نفذت البرنامج ولم يكن في الكود أي جزء يستدعي الدالة GetLength أو يقرأ قيمة الخاصية Length التي تستدعي الدالة GetLength، فسيظل هذا الخطأ مختفيا عن عينيك لفترة طويلة.

بل أكثر من هذا، لو أنك استدعيت دالة اسمها MyFunc مثلا:

result = MyFunc( )

ولا يوجد تعريف لها في برنامجك، فسيعمل البرنامج بشكل صحيح طالما لم يصل إلى السطر الذي تستدعي فيه هذه الدالة!.. وهذا يعني أنك لو أخطأت في كتابة حرف في اسم إحدى الدوال في الكود، فلن تكتشف هذا الخطأ إلا أثناء تشغيل البرنامج!

نفس الأمر سيحدث لو استدعيت دالة موجودة، لكن أرسلت لها عددا أقل أو أكثر من المعاملات التي تنتظرها، فلن تكتشف هذا الخطأ إلا عند تنفيذ السطر الذي يفعل هذا أثناء تشغيل البرنامج!

مفسر الكود Interpreter:
هذا السلوك من مترجم رينج يجعله مفسرا Interpreter وليس مترجما صرفا Compiler.. المفسر يجعل تنفيذ البرنامج سريعا مهما كان حجمه كبيرا، وهذا يوفر لك وقت تجريب الكود أثناء إنشاء البرنامج، لكنه في المقابل يجعل اكتشاف الأخطاء أصعب.. في الحقيقة تستخدم رينج نظاما هجينا يمزج بين المترجم والمفسر، فهي تستخدم آلة افتراضية Virtual Machine(VM) لتسمح لنا بتشغيل البرنامج على نظم تشغيل مختلفة، وهذا يعني أن ترجمة الكود تتم على مرحلتين، مرة بواسطة المترجم ومرة بواسطة المفسر (الآلة الافتراضية VM).

لكل هذا، من المهم عند كتابة المشاريع الكبيرة إضافة اختبارات لكل دالة تكتبها في الكود، وتشغيل كل هذه الاختبارات كلما أجريت أي تعديل في أي جزء من البرنامج، حتى تكتشف أي أخطاء قد يسببها هذا التعديل في أجزاء أخرى من البرنامج لست

منتبها لها.. هذا الأمر مهم في كل لغات البرمجة، لكنه أشد أهمية في لغات البرمجة المرنة مثل رينج، ففي لغات البرمجة ثابتة الأنواع Static-typing، لا تستطيع أن تضع في المتغير قيمة غير مناسبة لنوعه، وهذا يمنع أخطاء كثيرة من المنبع.. هذا يجعل الكود الذي تكتبه في تلك اللغات أصعب وأطول لكنه أكثر أمنا وأقل عرضة للأخطاء أثناء تنفيذ البرنامج.. أما لغات البرمجة المرنة، فهي تجعلك تكتب كودا أقل وتنشئ برنامجك بسرعة، لكن الثمن الذي تدفعه هو إمكانية حدوث أخطاء غير متوقعة أثناء تشغيل البرنامج، بسبب تعديل أدخلته على الكود أفسد القيم المتوقعة للمتغيرات.. هذا غير خطير في البرامج الصغيرة التي تحل مشكلة عابرة أو تجرب بها شيئا، لكن لو كنت تخطط لإنشاء برنامج كبير وبيعه للمستخدمين، والاستمرار في تطويره وصيانته لسنوات، فمن الضروري أن تحد من إمكانية حدوث هذه الأخطاء من منبعها، بكتابة اختبارات لكل جزء من الكود، وتشغيل هذه الاختبارات بعد كل تعديل تجريه.

لتطبيق هذا أنشأت ملفا اسمه aString.Tests.ring (ستجده كل اختبارات برنامج الشاعر في المجلد Tests بالأمثلة المرفقة بالكتاب)، وأضفت فيه عدة دوال كل منها تختبر إحدى دوال الفئة aString.. فمثلا، الدالة Test_aString_Text تختبر الخاصية aString.Text.. وهكذا.

والاختبار في الواقع بسيط، فكل ما سنفعله، هو أن نضع قيمة في الخاصية Text ثم نقرأ قيمة الخاصية Text مرة أخرى، ونتأكد من أن القيمة التي وضعناها فيها، تساوي القيمة التي قرأناها منها، أي أنها تعمل بشكل صحيح.. فإن كانت القيمتان مختلفتين، نطلق خطأ ورسالة باستخدام الأمر Raise نقول فيها إن الاختبار قد فشل.

ولتنفيذ كل هذه الاختبارات، يجب علينا استدعاء دوال الاختبار واحدة بعد الأخرى في بداية الملف aString.Tests.ring.

ولتسهيل كتابة الاختبارات، أنشأت ملفا اسمه Tests.ring يحتوي على بعض الدوال التي يتكرر استخدامها في الفحص وعرض رسائل الخطأ مثل الدالة AssertEqual التي تتحقق من تساوي القيمتين المرسلتين إليها، والدالة AssertError التي تتحقق

من أن الكود المرسل إليها يجب أن يسبب خطأ، والدالة PassedTest التي تعرض رسالة أن العنصر محل الاختبار قد نجح في الاختبارات.. لاحظ أن الملف كذلك يحتوي على متغير اسمه TestTarget، يجب أن تضع فيه اسم الوسيلة التي تختبرها في بداية دالة الاختبار.. هذا الاسم سيظهر في بداية رسائل الخطأ ليكون معناها واضحا.. هذا يجعل كتابة الاختبارات أسرع وأسهل وأقل حجما.

وستجد الملف Tests.ring في المجلد Tests ضمن الأمثلة المرفقة بالكتاب، ويمكنك استخدامه في أي ملف اختبار بتحميله في بداية الملف:

```
Load Tests.ring
```

والمثال التالي يريك كيف تستخدم هذه الدوال لاختبار الخاصية Text:

```
Func Test_aString_Text( )
  Testtarget = "aString.Text"
  str.Text = "تجربة"
  Assert(1, str.Text, "تجربة")
  PassedTest( )
EndFunc
```

لو فشلت الخاصية Text (اعتبر مثلا أنها تعيد النص "تجر")، فستظهر الرسالة:

```
aString.Text test1 failed:
"تجربة" and "تجر" are not equal.
```

أما لو نجح الاختبار، فستظهر لك الرسالة:

```
aString.Text passed.
```

لن أخوض أكثر من هذا في موضوع الاختبارات، لكنك ستجد ملف اختبار لكل فئة نكتبها في هذا المشروع، وما عليك إلا أن تفتحه وتنفذ الكود لتتأكد أن العنصر محل الاختبار قد عبر كل الاختبارات بنجاح.

واضع القيم الابتدائية للفئة Initializer:

هناك تسهيل مهم نحتاج لإضافته للفئة aString وهو إرسال النص إليها مباشرة في سطر إنشائها، كالتالي:

```
astr = new aString("تجربة")
```

سيكون هذا أفضل من أن نستخدم سطرين لفعل نفس الأمر:

```
astr = new aString
astr.Text = "تجربة"
```

في رينج، يتم تمرير النص المرسل في سطر إنشاء الفئة إلى دالة اسمها Init (اختصار للكلمة Initialize بمعنى وضع القيم الابتدائية).. إذن فكل ما عليك هو إضافة هذه الدالة إلى الفئة aString وتعريف معامل واحد لها:

```
Func init(str)
  qstr.append(str)
EndFunc
```

لاحظ أن كل ما فعلنا هو استخدام الوسيلة QString.Append لإدراج النص المرسل في نهاية النص الموجود في المخزن الداخلي qstr.. ربما تسألني هنا:

- ألن نحصل على نص خاطئ لو كان المتغير qstr يحتوي على نص سابق؟

نعم (تذكر أن السؤال المنفي يجاب عنه بنعم لتأكيد النفي وبلى لنفي النفي ☺).. ألم نقل إن الدالة Init تستدعى فقط عند إنشاء نسخة جديدة من الفئة؟.. هذا معناه أن النص الداخلي للفئة ما زال فارغا بكل تأكيد.

وهناك تسهيل آخر أريد إضافته، وهو السماح بإرسال نسخة من الفئة aString إلى حدث الإنشاء لنسخ النص منها.. مثل:

```
astr1 = new aString("تجربة")
astr2 = new aString(astr1)
```

لفعل هذا، يجب أن نتأكد أولا إن كان المعامل نصا عاديا أم كانا من النوع aString.. وقد كتبت دالة اسمها IsAString لتؤدي هذه الوظيفة، وسأضيفها في بداية الملف aString.ring قبل كود الفئة aString (تذكر أن الفئة يجب أن تكون آخر شيء في الملف):

```
Func IsAString(obj)
  If type(obj) = "OBJECT" and ClassName(obj) = "astring"
    return true
  else
    return false
  end
EndFunc
```

هذه الدالة تستخدم جملة شرط (سنتعرف على جمل الشرط في المقطع التالي)، لتعيد

True إذا كان المعامل المرسل إليها كائنا من الفئة aString، وغير هذا تعيد False.. وقد وضعناها خارج الفئة aString لأنها لا تؤدي وظيفة داخل الفئة، وإنما تتعامل مع كائنات من أي نوع، وتخبرنا إن كانت من نوع الفئة aString.. لو كنا داخل الفئة aString فنحن نعرف بالتأكيد أننا نتعامل مع aString فلا داعي للفحص ☺.

لاحظ أن الدالة Type تعيد نصا كل حروفه كبيرة، يمثل نوع القيمة المرسلة كمعامل.. هذه الدالة تتعامل مع الأنواع الأساسية الخاصة برينج، لهذا تنحصر القيم العائدة منها في الجدول التالي:

NUMBER	عدد
STRING	نص
LIST	قائمة
OBJECT	كائن

لهذا احتجنا إلى استخدام دالة أخرى اسمها ClassName تعيد اسم الفئة التي ينتمي إليها الكائن.. هنا قد تسألني:

- لماذا لم نستخدم الدالة classname فقط؟

لأنها ستسبب خطأ لو أرسلت إليها متغيرا عاديا ليس كائنا (مثل النصوص والأرقام).. لهذا تأكدنا أن نوع المتغير Object، قبل أن نحاول معرفة اسم الفئة التي ينتمي إليها.

لاحظ أننا نستطيع اختصار كود الدالة السابقة في سطر واحد كالتالي:

```
Func IsAString(obj)
  return type(obj) = "OBJECT" and ClassName(obj) = "astring"
EndFunc
```

فالمعامل and يربط بين شرطين ويعيد true أو false.. لهذا يمكننا إعادة ناتجه مباشرة بدون الحاجة إلى كتابة جملة شرط.

والآن، دعنا نستخدم الدالة IsAString داخل الدالة Init.. هذا هو الكود المعدل:

```
Func init(str)
  If IsAString(str)
    qstr.append(str.Text)
  Else
    qstr.append(str)
  End
EndFunc
```

كل ما فعلناه هو كتابة جملة شرط لفحص نوع المعامل str المرسل للدالة init:

- فإن كان من النوع aString فستعيد الدالة IsAString القيمة True وهذا معناه أن الشرط صحيح، وفي هذه الحالة سنقرأ النص من المتغير str باستخدام الخاصية Text ونضيفه للمتغير الداخلي qstr.
- أما إن كان من نوع آخر فستعيد الدالة IsAString القيمة False، وهذا معناه أن الشرط خاطئ، وسيقفز التنفيذ إلى المقطع Else، حيث سنفترض أن المعامل مجرد نص عادي، ونضيفه مباشرة للمتغير الداخلي qstr.. لاحظ أن المستخدم لو أرسل نوعا ثالثا غير نصوص رينج والنصوص الخاصة بنا، فسيتم تنفيذ المقطع Else أيضا، وستحاول رينج تحويل هذا النوع إلى نص، وهذا مفيد مثلا في التعامل مع الأرقام.

هنا، علينا أن نتوقف قليلا لنتعرف على جملة الشرط والمعاملات المنطقية التي تستخدم في تكوين الشروط.

جملة الشرط If Statement:

كتابة الشروط هي أهم جزء في البرمجة، ويمكنك أن تعتبرها البرمجة الحقيقية، لأن الشروط تمثل المنطق الذي يتحكم في تنفيذ البرنامج، كأن تجعل الكود يعرض رسالة تحذير للمستخدم إذا أدخل قيمة خاطئة، أو تنفذ البرنامج إذا أدخل قيمة مناسبة.

وتستخدم جملة الشرط If لكتابة الشروط في معظم لغات البرمجة، وهذه هي الصيغة العامة لجملة الشرط في رينج:

```
If cond1
  # Code Block 1
ElseIf cond2
  # Code Block 2
.
.
.
ElseIf CondN
  # Code Block N
Else
  # Code Block N + 1
End
```

حيث cond هو أي شرط مقبول برمجيا تكون نتيجته True (الشرط صحيح) أو False (الشرط خاطئ).

لاحظ أن المقاطع ElseIf و Else هي مقاطع اختيارية، أي أنك تستطيع فحص شرط واحد فقط في المقطع If، وقد تضيف جملة ElseIf لفحص شرط آخر Cond2 في حالة كون الشرط Cond1 خاطئا، ولا قيود عليك في إضافة أي عدد تريده من مقاطع ElseIf على حسب احتياجك.. وفي النهاية، إذا فشلت كل الشروط، يمكنك إضافة مقطع Else واحد فقط (وهو اختياري أيضا)، ولا يتم فيه فحص أي شروط، بل يعتبر الحالة العامة (الافتراضية) التي يتم اللجوء إليها إذا كانت كل الشروط السابقة خاطئة.

وفي المثال السابق، رأينا جملة شرط تتكون من مقطعين فقط:

- المقطع If يتأكد من أن المتغير ينتمي للنوع aString وفي هذه الحالة تعيد الدالة القيمة True.
- والمقطع Else إذا كان المتغير من أي نوع آخر (غير aString)، وفيه تعيد الدالة القيمة False.

وقد يكون الشرط عملية مقارنة بسيطة مثل:

```
if x > 1
```

أو قد يكون الشرط مركبا من أكثر من عملية مقارنة يدمجها معا معامل منطقي مثل:

```
if x > 1 or x = -1
```

ويكون ناتج كل عملية مقارنة قيمة منطقية:

- فالشرط الصحيح تكون قيمته True ويتم تمثيلها في رينج بالعدد ١.
- والشرط الخطأ تكون قيمته False ويتم تمثيلها في رينج بالعدد صفر.

والجدول التالي يلخص معاملات المقارنة والمعاملات المنطقية.. لاحظ أن نتائج الأمثلة تفترض أن x = 1 و y = 10.

المعامل	معناه	مثال	النتيجة
=	يساوي	? x = y	0
=!	لا يساوي	? x != y	1
>	أكبر من	? x > y	0
<	أصغر من	? x < y	1
>=	أكبر من أو يساوي	? x >= 1	1
<=	أصغر من أو يساوي	? x <= 1	1
And	و	? x = 1 and y = 2	0
&&		? x = 1 && y = 10	1
Or	أو	? x = 0 or x = 1	1
||		? x = 0 || x = 2	0
Not	ليس	? Not (x = 1)	0
!	(عكس الشرط)	? !(x = 0)	1

ملحوظة:

المعاملان And و Or يهملان الشروط التي لا يمكن أن تؤثر على النتيجة، وهو ما يسمى بفصل التيار Short Circuiting.. ففي حالة المعامل And لو كان شرط الأول خاطئا لا يتم فحص الشرط الثاني لأن النتيجة دائما ستكون False، وفي حالة المعامل Or لو كان الشرط الأول صحيحا، لا يتم فحص الشرط الثاني لأن النتيجة

ستكون دائما True.

هذا هو السبب في عدم حدوث خطأ في الكود:

If type(obj) = "OBJECT" and ClassName(obj) = "astring"

فلو كان الشرط الأول خاطئا (أي أن المتغير obj ليس كائنا) فلن تفحص رينج الشرط الثاني، وهذا جنبنا حدوث خطأ عند إرسال نص وليس كائنا إلى الدالة ClassName.

وكالعادة، هناك عدة أشكال لصياغة جملة If:

- يمكنك استخدام but بدلا من ElseIf و Ok بدلا من end:

```
if x < 10
  ? "x is less than 10"
but x < 20
  ? "x is greater than 9 and less than 20"
else
  ? "x is greater than 19"
ok
```

- يمكن تغليف المقطع if بقوسين متعرجين:

```
if x < 10 {
  ? "x is less than 10"
elseIf x < 20
  ? "x is greater than 9 and less than 20"
else
  ? "x is greater than 19"
}
```

فاختر الصيغة التي تستريح لها أكثر.

***

الآن، نحن جاهزون لاستخدام فئة النصوص العربية.. هناك دوال أخرى كثيرة نحتاج أن نضيفها لها، وستجدها بالفعل في الملف aString.ring المرفق بأمثلة الكتاب وسنعتمد عليها في كتابة كود برنامج الشاعر، لكني لن أشرح كيف كتبتها هنا، لأنها ستحتاج كتابا كاملا دون أن تضيف لك معلومات جديدة، بينما يمكنك فهم معظم الكود

بنفسك.. صحيح أنني لم لا أستخدم الفئة QString في تلك النسخة من الفئة aString، لكنك ستستطيع فهم هذا الكود بعد إتمام قراءة هذا الكتاب بإذن الله.. ولا يوجد مانع أن تحاول أن تضيف نفس الوسائل الموجودة في الفئة aString الخاصة بي إلى الفئة aString التي أنشأناها هنا بالاعتماد على الفئة QString.. سيكون تدريبا جيدا لك في وقت فراغك ☺.. المهم أن تتأكد من تشغيل ملف الاختبار aString.test.ring لتجربة الفئة التي تكتبها، لتتأكد أنها تعمل بشكل صحيح.

دوال الفئة aString:

سيكون من المفيد أن ألخص لك هنا وظائف الوسائل الموجودة في الفئة aString المرفقة بأمثلة الكتاب، لتكون مفهومة لك حينما نستخدمها في كتابة برنامج الشاعر.
وتنقسم هذه الدوال إلى ثلاثة أنوع:

دوال تغير النص مباشرة:

لا تعيد هذه الدوال أي قيمة، ولكنها تؤثر مباشرة على النص الموجود في الكائن الحالي.. هذه الدوال ليست كثيرة، وقد أضفتها لتوفر الكثير من الوقت عند إجراء تعديلات متتالية على النص (داخل حلقة تكرار مثلا).. والجدول التالي يلخصها لك:

SetText(str): تمحو النص الحالي، وتضع بدلا منه النص str.. ويتم استدعاء هذه الوسيلة تلقائيا لو وضعتَ نصا في الخاصية Text.
SetCharAt(index, chr): تضع الحرف chr بدلا من الحرف الموجود في الموضع Index.
Append(str): تضيف النص str إلى نهاية النص الحالي.. ويمكنك أن ترسل إليها أيضا مصفوفة لتضيف عناصرها إلى نهاية النص الحالي.. ويمكنك أيضا أن ترسل إليها أرقاما أو مصفوفة من الأرقام، حيث ستعاملها كأنها نصوص وتضيفها لنهاية النص الحالي.

AppendFrom(str, stratAt, charCount): تشبه الدالة السابقة، لكنها تسمح لك أن تأخذ من النص str حروفا عددها charCount بدءا من الموضع stratAt وإضافتها لنهاية النص الحالي.
Clear( ) : تمحو النص الحالي (سيصير في الفئة نص فارغ "").
Delete(startAt, charCount): تحذف حروفا من النص عددها charCount بدءا من الموضع startAt.
ShrinkTo(newLength): تقلّص طول النص إلى newLength، وهذا معناه حذف كل الحروف التي تزيد عن هذا الطول من نهاية النص.. ولو كانت newLength صفرا أو عددا سالبا فسيتم محو النص كله، ولو كان أكبر من طول النص الحالي فلن يحدث شيء.

دوال تعيد نصا معدلا:

هذه الدوال لا تؤثر على الكائن الذي يستدعيها، ولكنها تعيد كائنا جديدا يحتوي على النص المعدل.. معظم وسائل الفئة aString تنتمي لهذا النوع، لأن المحافظة على النص القديم تسمح لك باستخدامه في عمليات مختلفة.. لاحظ وجود نسختين من بعض الدوال مثل RemoveAt و RemoveAtStr، وكلتاهما لها نفس المعاملات وتؤدي نفس الوظيفة، لكن الفارق بينهما أن الدالة التي لا تنتهي بـ Str تعيد كائنا من النوع aString بينما الدالة التي تنتهي بـ Str تعيد نصا عاديا من نصوص رينج.. والجدول التالي يلخص لك هذه الدوال:

RemoveAt(startAt, charCount) و RemoveAtStr(startAt, charCount): تعيد نصا جديدا ناتجا عن حذف حروف من النص الحالي عددها charCount بدءا من الموضع startAt.
Remove(str) و RemoveStr(str): تعيد نصا جديدا ناتجا عن حذف النص str من كل مواضع ظهوره في النص الحالي.

Insert(str, pos) و **InsertStr(str, pos):** تعيد نصا جديدا ناتجا عن إضافة النص str في الموضع pos من النص الحالي.
+ : استخدم علامة الجمع للحصول على نص جديد ناتج عن تشبيك نصين من النوع aString (ويمكن أن يكون المعامل الثاني بعد + نصا من نصوص رينج العادية).

دوال تعيد جزءا من النص:

تعيد هذه الدوال النص كله أو جزءا منه بدون أي تغيير في النص الأصلي أو الجزء العائد.. والجدول التالي يلخص لك هذه الدوال:

GetText( ): تعيد كل النص.. ويتم استدعاؤها تلقائيا لو قرأتَ الخاصية Text.
ToString( ): تعيد كل النص.
Right(charCount) و **RightStr(charCount):** تعيد جزءا من نهاية النص طوله charCount.
Left(charCount) و **LeftStr(charCount):** تعيد جزءا من بداية النص طوله charCount.
RemoveFirst(charCount) و **RemoveFirstStr(charCount):** تتجاوز جزءا من بداية النص طوله charCount وتعيد باقي النص.
RemoveLast(charCount) و **RemoveLastStr(charCount):** تتجاهل جزءا من نهاية النص طوله charCount وتعيد باقي النص.
Mid(startAt, charCount) و **MidStr(startAt, charCount):** تعيد جزءا من النص يبدأ عند startAt وطوله charCount.

:ReadNumberAt(startAt)

تعيد الجزء الرقمي من النص الحالي الموجود بدءا من الموضع startAt.. فإذا لم يكن الحرف الموجود في هذا الموضع رقما، فستعيد صفرا، ولو كان رقما، فستستمر في قراءة الحروف التالية له إلى أن تصل إلى حرف لا يصلح كرقم.. فمثلا:

```
x = new aString("Day 11")
? x. ReadNumberAt(5)
```

سيعرض ١١ في نافذة المخرجات.

:Split(sep) و SplitStr(sep)

تعيد مصفوفة عناصرها هي النصوص الناتجة عن تقسيم النص عند الفاصل sep.. فمثلا يمكنك تقسيم النص "1, 2, 3" عند الفاصل "," لتحصل على مصفوفة تحتوي على ثلاث عناصر هي بالترتيب "1" و "2 " و "3 ".. ولا مانع من استخدام فاصل يتكون من أكثر من حرف، ففي المثال السابق يمكنك مثلا استخدام الفاصل " ," وهذا سيخلصك من المسافة الزائدة التي تظهر قبل العدد ٢ والعدد ٣ في المصفوفة الناتجة.

دوال البحث في النص:

استخدم هذه الدوال للبحث في النص الحالي عن نص جزئي.. وهذه الدوال هي:

:IndexOf(str, startAt)

تبحث عن النص str في النص الحالي بدءا من الموضع startAt، فلو وجدته تعيد الموضع الذي وجدته فيه، ولو لم تجده تعيد صفرا.

:IndexOfAny(lstStr, startAt)

تشبه الدالة السابقة، لكنها تستقبل مصفوفة نصوص للبحث عن أي من عناصرها في النص الحالي، وتعيد موضع أول عنصر تجده أو صفرا أن لم تجد أي عنصر.

:StartsWith(str)

تعيد True إذا كان النص الحالي يبدأ بالنص str.

:EndsWith(str) تعيد True إذا كان النص الحالي ينتهي بالنص str.
:EndsWithAny(strList) تعيد True إذا كان النص الحالي ينتهي بأي نص تحتويه المصفوفة strList. ملحوظة: لم أحتج للدالة StartsWithAny(strList) لهذا سأترك لك كتابتها بنفسك كتدريب.
:ContainsAt(str, startAt) تعيد True إذا كان النص str موجودا في النص الحالي في الموضع startAt.. أي أن النص الجزئيّ الذي يبدأ عند startAt وطوله Len(str) يساوي النص str.

دوال أخرى:

هناك بعض الدوال الأخرى التي تتعامل مع النصوص العربية (سأشرحها في الفصول التالية)، إضافة إلى هاتين الدالتين الهامتين:

:CompareTo(str) تقارن النص الحالي بالنص str، حيث تعيد ناتج المقارنة كالتالي: -١: النص الحالي أصغر من النص str (يسبقه أبجديا). ٠: النصان متساويان. ١: النص الحالي أكبر من النص str (يليه أبجديا). لاحظ أنك تستطيع استخدام معاملات المقارنة (=> ,> ,=< .< ,=) مباشرة للمقارنة بين نصين من نوع الفئة aString (ويمكن أن يكون النص الثاني أحد نصوص رينج العادية)، علما بأن كل هذه المعاملات تستخدم الدالة CompareTo لأداء وظيفتها.
:IsNumeric( ) تعيد True إذا كان النص الحالي مكونا كله من أرقام (يصلح كعدد).

والآن دعنا نركز على إكمال مهمتنا الأساسية هنا وهي برنامج الشاعر.

-٣-

تقطيع النص إلى سطور وكلمات

القائمة List (المصفوفة Array):

أول خطوة للتعامل مع القصيدة التي يكتبها مستخدم برنامجنا، هي تقطيع القصيدة إلى سطور ووضع كل منها في قائمة List.

لقد تعرفنا من قبل على القائمة، حينما ذكرنا أن النص هو مصفوفة (أو قائمة) من الحروف، ويمكن الوصول إلى كل حرف عن طريق رقمه في القائمة.

في الحقيقة، هذا ليس النوع الوحيد من القوائم، فبإمكانك وضع أي نوع من البيانات في القائمة.. هذه مثلا قائمة رقمية:

```
lstNum = [100, 200, 300]
```

تحتوي القائمة lstNum على ثلاث خانات، هي الخانات رقم ١ و ٢ و ٣، والقيم الموضوعة فيها بالترتيب هي ١٠٠ و ٢٠٠ و ٣٠٠.

1	2	3
100	200	300

والكود التالي يضيف ٥ إلى العنصر الثاني في القائمة، ويعرض قيمتها على الشاشة:

```
lstNum[2] += 5
? lstNum[2]      #205
```

لاحظ أن الرمز =+ يعني إضافة القيمة الموجودة على الجهة اليمني إلى القيمة الموجودة على الجهة اليسرى.. في الحقيقة هذا مجرد اختصار للكود التالي:

```
lstNum[2] = lstNum[2] + 5
```

طبعا الكود الأول أكثر اختصارا وأقل إرباكا من الكود الأخير.. فلو لم تكن لك سابق خبرة في البرمجة، فسيبدو لك الكود السابق كأنه معادلة لا يمكن أن يتساوى طرفاها!

وسبب الارتباك هنا أن الرمز = في كثير من لغات البرمجة له وظيفتان مختلفتان:

١- المقارنة Comparison:

حيث تفحص اللغة تساوي القيمتين وتعيد قيمة منطقية True أو False، مثل:

If x > y

٢- الإسناد Assignment:

ويعني وضع قيمة في متغير، مثل:

x = 3

وينشأ الارتباك هنا حينما نستخدم المتغير على الجهتين، مثل:

x = 10 – x

فلأول وهلة ستسأل نفسك: كيف يمكن أن يتساوي x مع 10 – x ؟

هنا يجب أن تتذكر أن هذه ليست مقارنة، ولا تعيد True أو False، وإنما هذه عملية وضع قيمة في متغير، حيث تحسب رينج قيمة التعبير المكتوب على يمين العلامة = ثم تضع قيمته في المتغير الموجود على يسار العلامة.. لهذا تستطيع أن تقرأ هذا الكود باعتباره:

x **will** = 10 – x

أي أن قيمة x الجديدة، ستساوي ناتج طرح قيمة x الحالية من العدد ١٠.

لاحظ أن القائمة غير مخصصة لنوع واحد من البيانات.. فلا يوجد ما يمنعك من إضافة نص إلى الخانة الأولى في القائمة lstNum:

```
lstNum[1] = "Ali"
? lstNum[1]
```

يمكنك القول إن كل خانة في القائمة هي متغير، تنطبق عليه نفس قواعد المتغيرات في لغة رينج.. لكن القائمة تتيح لنا التعامل مع كل متغير برقمه بدلا من اسمه، وهذا يوفر علينا تعريف متغيرات كثيرة عند التعامل مع البيانات المتشابهة.

ويمكنك إضافة عنصر جديد إلى نهاية القائمة باستخدام علامة الجمع.. مثال:

```
lstNum + 400
? lstNum[4]      #400
```

بل يمكنك أن تجمع قائمة أخرى، لكن في هذه الحالة ستضاف كعنصر واحد في

الخانة الأخيرة في القائمة.. ألم نقل إن كل خانة في المصفوفة تعتبر متغيرا؟.. إذن يمكنك أن تضع فيها نصا أو رقما أو كائنا أو حتى قائمة جديدة:

```
lstNum + [500, 600]
```

وفي حالتنا هذه، يمكننا أن نتعامل مع عناصر المصفوفة الداخلية كالتالي:

```
? lstNum[5][1]  #500
? lstNum[5][2]  #600
```

فالفهرس [5] يعيد إليك العنصر الخامس من القائمة lstNum، لكن هذا العنصر هو نفسه قائمة تحتوي على خانتين، لهذا استخدمنا الفهرسين [1] و [2] لقراءتهما.

ويمكنك أن تعيد كتابة الكود السابق بطريقة أخرى أكثر وضوحا، بوضع المصفوفة الموجودة في الخانة الخامسة في متغير، والتعامل معها من خلاله:

```
lst = lstNum[5]
? lst[1]  #500
? lst[2]  #600
```

معنى هذا أننا نستطيع تعريف قائمة متعددة الأبعاد multi-dimensional أو بتعبير آخر شجرة من العناصر، حيث كل خانة في القائمة يمكن أن تحتوي على قائمة من العناصر، وكل خانة في القائمة الداخلية يمكن أن تحتوي على قائمة من العناصر... وهكذا، على حسب احتياجك.

وهناك استخدام مفيد وشائع للمصفوفة ثنائية البعد، يسمى بالقاموس Dictionary.

أنت تعرف أن القاموس في لغات البشر يحتوي على كلمات ومعانٍ، وكذلك القاموس في لغات البرمجة فهو يحتوي على مفاتيح Keys وقيم Values.. والمفتاح هو قيمة متفردة لا يمكن أن تتكرر في القاموس، ويحتفظ القاموس بقيمة تناظر كل مفتاح، بحيث يمكنك الوصول إلى القيمة باستخدام المفتاح بدلا من استخدام رقم الخانة.

افترض أن لدينا هذه المصفوفة ثنائية البعد:

```
lstNums = [ ["one", 1], ["two", 2], ["three", 3] ]
? lstNums[1][1]     #one
? lstNums[1][2]     #1
```

لا جديد حتى الآن.. لكن رينج تقدم تسهيلا، لتجعلك تتعامل مع هذه القائمة باعتبارها قاموسا، حيث يمكن اعتبار النصوص one و two و three مفاتيح، لنحصل على

القيم المناظرة لأي منها مباشرة كالتالي:

```
? lstNums["one"]     #1
? lstNums["two"]     #2
? lstNums["three"]   #3
```

وهناك تسهيل آخر، حيث يمكنك التعامل مع أي نص لا يحتوي على مسافات ورموز بالصيغة one: بدلا من الصيغة "one".. وهذا يسمح لنا بإعادة تعريف القاموس السابق كالتالي:

```
lstNums = [ [:one, 1] , [:two, 2] , [:three, 3] ]
? lstNums[:one]   #1
? lstNums[:two]   #2
? lstNums[:three] #3
```

وهذا يوصلنا إلى أسهل صيغة تتيحها لك رينج لتعريف القاموس، بوضع كل قيمة في المفتاح المناظر لها مباشرة باستخدام العلامة = بدون الحاجة إلى استخدام أقواس القوائم الداخلية:

```
lstNums = [:one = 1 , :two = 2, :three = 3]
```

هذه الصيغة أكثر اختصارا ووضوحا وأقل عرضة للأخطاء، وهي مماثلة تماما للصيغ السابقة (لهذا ما زلت تستطيع التعامل مع القوائم الداخلية بأرقام خاناتها لو أردت) لكن هذه الصيغة أكثر دلالة على تعريف قاموس له مفاتيح وقيم.. وبالطبع تستطيع وضع المفاتيح بين علامتي تنصيص، وهو ما سيكون ضروريا إذا كان اسم المفتاح يحتوي على مسافات أو رموز.

تقطيع السطور:

نريد الآن تقطيع القصيدة إلى سطور.. وبما أننا سنتعامل مع النصوص العربية من خلال الفئة aString، فمن المنطقي أن نضع القصيدة كلها في متغير من نوع هذه الفئة، ولهذا سنضيف إليها دالة (وليكن اسمها GetLines) لتقطيع النص الموجود فيها إلى سطور مستقلة، ووضع كل سطر في نسخة خاصة به من الفئة aString، وإضافة كل هذه السطور إلى قائمة ونجعل هذه الدالة تعيدها إلينا.

فلنبدأ بأول خطوة: كيف نقطّع النص إلى سطور؟

يتم تمثيل فواصل السطور في البرمجة برمزين:

- الحرف رقم ١٣ في ترميز ASCII ويسمى ناقل السطر (Carriage Return (CR، ويوضع في نهاية السطر القديم.. لاحظ أن بعض نظم التشغيل لا تستخدم هذا الحرف وتكتفي بالحرف LF.
- الحرف رقم ١٠ في ترميز ASCII ويسمى بمغذي السطر (Line Feed (LF، ويوضع في بداية السطر الجديد.

وتسهل رينج الأمور بتعريف متغير اسمه nl (اختصارا لـ New Line) يمثل سطرا جديدا، حيث يمكنك إضافته لنهاية أي نص لإضافة سطر جديد.. جرب مثلا:

```
? "Line1" + nl + "Line2"
```

سترى على الشاشة:

```
Line1
Line2
```

كما تقدم لك رينج دالة جاهزة اسمها str2List (حيث 2 هي اختصار للكلمة To، أي أن اسم الدالة هو Str To List) لتقطيع سطور النص ووضعها في قائمة.. مثال:

```
x = "Line1
line2

line3"
lines = str2list(x)
? Len(lines)
? lines
```

لاحظ ما يلي:

- يمكنك كتابة عدة سطور ضمن النص الموجود بين علامتي التنصيص.
- تستخدم الدالة Len لمعرفة عدد عناصر القائمة.. في مثالنا هذا سيكون عدد العناصر ٤، لأن الدالة str2list لا تحذف السطور الفارغة.
- يتيح لك الرمز ? طباعة كل عناصر القائمة دفعة واحدة، حيث سيظهر كل منها في سطر منفصل.

لكن نظرا لأن الدالة str2list لن تحذف الخانات الفارغة وستعيد نصوصا عادية (ليست من النوع aString) ، فلن نعتمد عليها هنا، وعلينا تقطيع السطور بأنفسنا يدويا.. فكيف سنفعل هذا يا ترى؟

خوارزمية تقطيع السطور:

الخوارزمية Algorithm هي خطوات حل المسائل، وهي منسوبة لاسم عالم الرياضيات المسلم "الخوارزمي".. ولتقطيع النص إلى سطور سنستخدم الخوارزمية التالية:

- سنعرّف قائمة اسمها Lines ستكون فارغة في البداية.. يتم تعريف القائمة الفارغة في رينج باستخدام قوسين فارغين [ ].

Lines = []

- سنعرّف متغيرا اسمه Line نصع فيه في البداية نصا فارغا "".

line = ""

- سنعرف متغيرين يمثلان رمزي نهاية السطر، وسنستخدم في هذا الدالة Char (اسمها اختصار لكلمة "الحرف" Character)، وهي تأخذ رقما من ترميز ASCII وتعيد إلينا الحرف المناظر له:

Lf = char(10)
Cr = char(13)

- بدءا من أول النص، سنفحص كل حرف، لنتأكد إن كان واحدا من رمزي السطر:

  - فإن كانت الإجابة لا، فسنضيف الحرف إلى المتغير Line.
  - وإن كانت الإجابة نعم، فقد وصلنا إلى نهاية السطر، ولدينا الآن سطر كامل موضوع في المتغير Line، وعلينا إضافته إلى قائمة السطور Lines وإفراغ المتغير Line بوضع نص فارغ فيه لنعيد استخدامه في الحصول على السطر التالي.

- لاحظ أن هذا سيضيف خانات فارغة إلى المصفوفة بسبب وجود رمزين مختلفين لكل سطر (Cr, LF)، وكذلك لو كانت هناك سطور فارغة،.. لحل

هـذه المشـكلة يجـب أن نفحـص المتغيـر Line ولا نضـيف قيمتـه إلـى قائمـة السطور إذا كان فارغا.

- بعد أن حصلنا على السطر، سننتقل إلى الحرف التالي في النص ونكرر نفس الخطوات السابقة، وهكذا إلى آخر حرف في النص.. تسمى هذه العملية بحلقة التكرار Loop، وهذا هو موضوع الفقرة التالية.

حلقة التكرار For Loop:

تستخدم حلقات الدوران Loops لتكرار تنفيذ بعض الأوامر البرمجية لعدة مرات.. والمثال التالي يطبع الأعداد من ١ إلى ١٠ على الشاشة:

```
For i = 1 to 10
   ? i
Next
```

السطور الثلاثة السابقة توفر عليك كتابة ١٠ سطور مستقلة لطباعة عشر قيم على الشاشة.. وهذا بفضل حلقة التكرار For Next.. هذه الحلقة تستخدم عدادا counter وهو متغير بأي اسم تختاره أنت مثل i.. وتستمر حلقة التكرار في الدوران من قيمة ابتدائية للعداد (وهي هنا ١) تظل تزيد باستمرار إلى أن تصل إلى قيمة نهائية (وهي هنا ١٠).. ويمكنك لو أردت أن تحدد مقدار الزيادة في كل خطوة باستخدام الكلمة Step.. والكود التالي يطبع الأعداد الفردية من ١ إلى ١٠ بزيادة ٢ في كل خطوة:

```
For i = 1 to 10 Step 2
   ? i
Next
```

ويمكنك كتابة حلقة تكرار عكسية (تعد من الأكبر للأصغر) باستخدام خطوة سالبة.. والكود التالي يطبع الأعداد من ١٠ إلى ١:

```
For i = 10 to 1 step -1
   ? i
Next
```

وتظهر أهمية حلقات التكرار عند التعامل مع القوائم، حيث نستخدم العداد كمفهرس indexer للمرور عبر كل عناصر القائمة.. والكود التالي يضاعف كل عنصر في

القائمة lstNum بضربه في ٢:

```
lstNum = [1, 2, 3, 4]
For i = 1 to 4
   lstNum[i] *= 2
Next
```

لاحظ أن الرمز =* يشبه الرمز =+ لكن النجمة في البرمجة هي علامة الضرب، وهذا يعني أن الكود:

```
lstNum[i] *= 2
```

هو اختصار للكود:

```
lstNum[i] = lstNum[i] * 2
```

ويمكنك تعميم نفس القاعدة على المعاملين =- و =/ حيث / هي علامة القسمة.

وقد لا يكون من المناسب أن تكتب عدد خانات المصفوفة يدويا، فقد يتغير أثناء تنفيذ البرنامج بإضافة عناصر للقائمة أو حذف عناصر منها، لهذا فالأفضل أن تستخدم الدالة Len لمعرفة عدد العناصر:

```
For i = 1 to Len(lstNum)
   lstNum[i] *= 2
Next
```

ولأن المرور عبر عناصر القائمة أمر شائع جدا في الكود، تقدم لغات البرمجة صيغة أخرى مختصرة له، لهذا تمنحنا رينج حلقة التكرار For … in، ومعناها بوضوح: لكل عنصر في المصفوفة نفذ ما يلي.. هكذا سيصير الكود السابق لو كتبناه بهذه الصيغة:

```
For n in lstNum
   n *= 2
Next
```

في هذه الصيغة نستخدم المتغير n ليحمل قيمة الخانة الحالية التي نمر عبرها في المصفوفة.. في أول لفة في حلقة التكرار سيشير n إلى قيمة أول خانة في المصفوفة، وفي اللفة التالية سيشير إلى قيمة الخانة التالية وهكذا.. والجميل في رينج أن المتغير n هو متغير مرجعي Reference Variable، وهذا يعني أن أي تغيير يحدث للمتغير n سينتقل إلى الخانة التي يشير إليها في القائمة، وهذا واضح في الكود

السابق، فقد غيرنا قيمة n فقط، لكنك لو عرضت عناصر المصفوفة lstNum بعد تنفيذ حلقة التكرار، فسترى أن قيمها قد تضاعفت فعلا.. جرب:

```
? lstNum
```

وبما أن النص هو قائمة من الحروف، فمن السهل أن نستخدم هذه الصيغة للمرور عبر كل حروف النص.. والكود التالي سيطبع كل حرف من النص في سطر مستقل:

```
s = "test"
For c in s
   ? c
Next
```

الدالة GetLines:

الآن، لدينا كل ما نريده لكتابة الدالة GetLines.. أضف هذا الكود داخل الفئة aString (قبل القسم الخاص private):

```
Func GetLines( )
    Lines = []
    line = ""
    Lf = char(10)
    Cr = char(13)
    n = qstr.Count( ) - 1
    For i = 0 to n
        c = qstr.mid(i, 1)
        If c != Lf and c != Cr
          line += c
        ElseIf len(line) > 0
          lines + new aString(line)
          line = ""
        End
    Next
    return lines
EndFunc
```

هذه الدالة فيها مشكلة صغيرة، فهي لن تعمل بشكل صحيح إلا إذا كان النص ينتهي بسطر فارغ، لأننا لا نضيف السطر إلى القائمة إلا عندما نصل إلى علامة السطر

الجديد!.. لهذا لو كان النص ينتهي بحرف عاديّ، فسيحتوي المتغير line على هذا السطر، لكنه لن يضاف إلى القائمة Lines!

لحل هذه المشكلة، أضِف هذا الكود بعد السطر next وقبل السطر return lines:

```
If Len(line) > 0
  lines + new aString(line)
End
```

الآن ستعمل الدالة GetLines بشكل صحيح.

تقطيع السطر إلى كلمات:

الخطوة التالية هي تقطيع كل سطر (شطر شعري من أبيات القصيدة) إلى كلمات، مع ملاحظة أن المسافة ليست الحرف الوحيد الذي يفصل الكلمات، فقد تكون هناك فاصلة أو نقطة أو غير ذلك.. على كل حال، يجب أن نتخلص من كل الرموز التي لا تمثل حروف الأبجدية العربية وعلامات التشكيل، فهذه فقط هي ما يهمنا عند تحليل وزن القصيدة.

سنحتاج إذن لكتابة دالة تخبرنا إن كان الحرف الذي نرسله إليها حرفا عربيا أو أحد علامات التشكيل.. فكيف نفعل هذا؟.. لدينا هنا عدة طرق للتفكير:

الطريقة الأولى:

أن نكتب جمل شرط كثيرة تقارن الحرف المرسل بكل حرف من حروف اللغة العربية وكل علامة تشكيل، ونعيد True عند حدوث التساوي.. شيء مثل هذا:

```
If hrf = "أ"
  return True
ElseIf hrf = "ب"
  return True
// .....
Else
 return False
End
```

لكن مشكلة هذه الطريقة أن الكود سيكون طويلا جدا، ناهيك أنه سيجري عددا كبيرا من المقارنات قبل أن يخبرك مثلا أن الياء هي حرف عربي لأنها تقع في آخر الأبجدية!.. لا تنسَ أننا سنفحص كل حرف في القصيدة (قد تتكون القصيدة من مئات الحروف)، وهذا لا يبدو شيئا ذكيا في البرمجة.. صحيح أن الحواسيب اليوم سريعة، لكن من الجيد أن تتدرب دائما على تحسين كفاءة الكود الذي تكتبه، لأن هذا سيفيدك في البرنامج التي تحتاج لحسابات طويلة تستهلك وقتا ملموسا.. أيضا: ما المكسب من أن نكتب كودا طويلا يضيع منا وقتا في كتابته ويضيع من الحاسوب وقتا في تنفيذه؟.. هذا بالتأكيد أسوأ كود يمكن أن يكتبه المبرمج.

الطريقة الثانية:

أن نضع كل الحروف العربية في قائمة ونستخدم الدالة Find للبحث في القائمة عن الحرف.. هذه الدالة تعيد رقم العنصر في القائمة، ولو لم تجده تعيد صفرا.. هذا الحل أفضل قليلا من الحل الأول لأنه سيختصر الكود الذي نكتبه، لكنه ما زال سيجري عمليات مقارنة كثيرة إلى أن يجد الحرف في القائمة.

```
Tashkeelat = ["ْ", "ّ", "َ", "ً", "ُ", "ٌ", "ِ", "ٍ"]

arabicLetters = ["ء","ا", "أ", "إ", "آ", "ئ", "ؤ", "ب", "ت", "ة",
"ض", "ص", "ش", "س", "ز", "ر", "ذ", "د", "خ", "ح", "ج", "ث",
"ى", "و", "ه", "ن", "م", "ل", "ك", "ق", "ف", "غ", "ع", "ظ", "ط",
"ي"]

Func IsTashkeel(hrf)
   return find(Tashkeelat, hrf) > 0
EndFunc

Func IsALetter(hrf)
   return find(arabicLetters, hrf) > 0
EndFunc
```

ويمكنك تعديل هذه الطريقة قليلا بوضع كل الحروف العربية متلاصقة في نص واحد، واستخدام الدالة SubStr للبحث عن الحرف الذي تريده، فإن أعادت رقما أكبر من

صفر، فهذا معناه أنها وجدت الحرف (ونفس الأمر مع علامات التشكيل):

```
arabicLetters = "اأإآىءئؤبتةثجحخدذرزسشصضطظعغفقكلمنهوي"

Func IsALetter(hrf)
    return SubStr(arabicLetters, hrf) > 0
EndFunc
```

الطريقة الثالثة:

يمكن أن نستخدم الدالة StrCmp لمقارنة الحروف.. هذا سيجعل الكود أقل ما يمكن، مع إجراء ثلاث أو أربع مقارنات على الأكثر.

وتستقبل الدالة StrCmp نصين، وتعيد رقما يخبرك بنتيجة المقارنة بينهما:

- فتعيد صفرا إذا كان النصان متساويين.. هذا يعني أن لهما نفس الحروف بنفس الترتيب بنفس الحالة (لاحظ أن الحرف a لا يساوي الحرف A).
- وتعيد رقما أصغر من صفر إذا كان النص الأول أصغر من النص الثاني أي أنه يسبقه في الترتيب الأبجدي (النص "أحمد" أصغر من النص "محمد").
- وتعيد رقما أكبر من صفر إذا كان النص الأول أكبر من النص الثاني أي أنه يليه في الترتيب الأبجدي (النص "محمد" أكبر من النص "أحمد").

وباستخدام الدالة StrCmp يمكننا أن نعرف أن الحرف عربي إذا كان أكبر من أو يساوي أول حرف عربي (وهو الهمزة "ء") وأصغر من أو يساوي آخر حرف عربي (وهو الياء "ي"). لكن لاحظ أن الحروف العربية تنقسم إلى مجموعتين منفصلتين:

- الأولى تمثل الحروف بين ء و غ.
- والثانية تمثل الحروف بين ف و ي.

ملحوظة:
السبب في وجود فجوة بين هاتين المجموعتين، هو وجود بعض الحروف الفارسية والباكستانية في هذا الجزء من الترميز، وهما من اللغات التي تكتب بالأبجدية العربية لكنهما تضعان بعض العلامات فوق بعض الحروف العربية.

أما علامات التشكيل، فتنحصر بين تنوين الفتح والسكون، وهي تالية مباشرة للمجموعة الثانية من الحروف الأبجدية، ويمكن دمجها معها إذا لم تكن تريد التفرقة بين الحروف العربية والتشكيلات.. هذا إذن هو الكود المطلوب:

```
Func IsTashkeel(hrf)
  return strcmp(hrf,"ً") >= 0 and strcmp(hrf,"ْ") <= 0
EndFunc

Func IsALetter(hrf)
  return (strcmp(hrf, "ء") >= 0 and strcmp(hrf, "غ") <= 0 ) or
         (strcmp(hrf, "ف") >= 0 and strcmp(hrf, "ي") <= 0 )
EndFunc
```

لاحظ أهمية وضع الأقواس في الشرط المركب في الدالة IsALetter، فهو يخبر رينج بأولوية تنفيذ and و or.. إزالة الأقواس أو تغيير مواضعها قد يعطيك نتائج لا تتوقعها.. فمثلا ناتج هذا الشرط سيكون صفرا:

```
(1 and 0 or 1) and 0
```

بينما ناتج هذا الشرط سيكون ١:

```
(1 and 1) or (1 and 0)
```

ويمكننا إضافة دالة اسمها IsAChar تعيد True إذا كان الحرف عربيا أو إحدى علامات التشكيل، وكل ما ستفعله هو استدعاء الدالتين IsTashkeel و IsALetter ودمج نتيجتيهما باستخدام المعامل or.. تذكر أن المعامل Or لن يستدعي الدالة IsALetter إلا إذا أعادت الدالة IsTashkeel القيمة False، وهذا يعني أن هذا الكود لن يُجري مقارنات كثيرة لا ضرورة لها، وهو مناسب جدا ومختصر.

```
Func IsAChar(hrf)
  return IsTashkeel(hrf) or IsALetter(hrf)
EndFunc
```

وستجد هذه الدوال في الملف ArabicLetters.ring وستجد اختبارات لها في الملف ArabicLetters.Tests.ring في المجلد Tests.

الآن، سنضيف جملة تحميل هذا الملف في بداية ملف الفئة aString:

```
Load "ArabicLetters.ring"
```

وبهذا يمكننا إضافة دالة تقطيع الكلمات كالتالي:

```
Func GetWords( )
   words = []
   word = ""
   n = qstr.Count( ) - 1
   For i = 0 to n
         c = qstr.mid(i, 1)
         If IsAChar(c)
            word += c
         ElseIf c != Letters.Kashida and len(word) > 0
            words + new aString(word)
            word = ""
         End
   Next

   If len(word) > 0 { words + new aString(word) }
   return words
EndFunc
```

كما تلاحظ، الكود مشابه لكود الدالة GetLines، لكن الفواصل هنا هي أي حروف غير عربية، مع تجاهل الكشيدة (فهي ليست حرفا نضيفه إلى الكلمة، وليست فاصلا بين كلمتين).. فالكشيدة هي مجرد وصلة تزيد المسافة بين الحروف (كأن تكتب "غـــد" بدلا من "غد").. ويمكنك أن تكتبها بضغط زر التحويل Shift مع الحرف ت من لوحة المفاتيح.. هذا يعني أن الكشيدة ليست حرفا ولا تشكيلا، ولكنها تعطي شكلا جماليا.. وقد تخلصنا منها لأنها لا تؤثر على وزن الشعر، لكننا أيضا لم نعتبرها فاصلا بين كلمتين.. هذا هو سبب الشرط:

```
c != Letters.Kashida
```

لاحظ أن Kashida هي خاصية عرفتها في فئة اسمها Letters تحمل بعض أسماء الحروف العربية التي يصعب على العين تمييزها بين الكود المكتوب بالأنجليزية.. لكنك تستطيع أن تكتب الكشيدة مباشرة في الكود السابق كالتالي:

```
c != "-"
```

لكن مع الحذر أنك قد تظن فيما بعد أن هذه هي علامة الطرح أو الشرطة المنخفضة!

الثوابت Constants والمرقمات Enums:

تقدم معظم لغات البرمجة نوعا يسمى الثابت Constant، وهو يشبه المتغير لكن قيمته تظل ثابتة لا يمكن تغييرها.. هذا غير موجود في رينج، والطريقة الوحيدة للحصول عليه هي تعريف الثابت كخاصية داخل إحدى الفئات واستخدام وسيلة وضع القيمة SetX() بدون متغير لمنع تغيير قيمة هذه الخاصية، ويمكنك أيضا أن تضعها في الجزء الخاص Private من الفئة.

نفس الأمر ينطبق على المرقم Enumerator، وهو تركيب برمجي يحتوي على مجموعة ثوابت تربطها علاقة ما، مثل مرقم أيام الأسبوع، الذي يحتوي على ثوابت بأسماء الأسبوع، موضوع في كل منها رقم اليوم في الأسبوع.. لا تدعم رينج هذا الأمر أيضا، لكنك تستطيع تعريف هذه الثوابت كخصائص للفئة وجعلها للقراءة فقط.

ولتسهيل هذا عليك، كتبت دالة لإنتاج المرقمات، تستقبل قاموسا يحتوي على أسماء الثوابت وقيمها، وتعيد إليك نصا يمثل كود الفئة.. وكتبت أيضا دالة لإنتاج المؤشرات Flags، وهي حالة خاصة من المرقمات لا مجال لشرحها الآن.. وقد نشرت هاتين الدالتين مع أمثلة لاستخدامهما على موقع GitHub:

https://github.com/VBAndCs/Ring-Enum-Generator

وقد تمت إضافتهما إلى مجلد الأمثلة في آخر إصدار من رينج.

لاحظ أنني في الكود السابق فضلت استخدام القوسين { } في مقطع If لأن مقطع الكود مكون من جملة واحدة، ومن الأفضل وضعه في نفس السطر مع جملة الشرط:

```
If len(word) > 0 { words + new aString(word) }
```

هذا أكثر اختصارا من:

```
If len(word) > 0
  words + new aString(word)
End
```

في الحقيقة تسمح رينج بكتابة أكثر من أمر متتالٍ بدون فواصل مثل:

```
If len(word) > 0 words + new aString(word)
```

لكنـي لا أنصـحك بفعـل هـذا، لأنـه مربـك ويجعـل قـراءة الكـود وفهمـه أصـعب، لهـذا استخدمت { } كعلامة مميزة.. لكنْ هناك حل بديل، هو أن تضع الشرط بين قوسين () وتستغني عن القوسين المتعرجين، لكن في هذه الحالة عليك إنهاء السطر بالكلمة end كالتالي:

If (len(word) > 0) words + new aString(word) End

استخدم الصيغة التي تراها أسهل بالنسبة لك.

ملحوظة:
في الفصول القادمة سنبدأ في تحليل القصيدة عروضيا.. وسيقدم لك هذا الكتاب شرحا مبسطا مختصرا لأساسيات علم العروض التي نحتاجها لإنشاء برنامج الشاعر، لكن لو شئت تفاصيل إضافية، فيمكنك قراءة دروس "**موسيقى الشعر العربي**" المنشورة على مدونتي (سأضع لك الرابط أيضا في المجلد "عن المؤلف" المرفق بأمثلة الكتاب): **http://mhmdhmdy.blogspot.com/2013/02/blog-post_1516.html**

– ٤ –
كلمات خاصة إملائيا

الكتابة الإملائية والكتابة العروضية:

هناك قواعد للإملاء تحكم كتابتنا لكلمات اللغة العربية كما في هذا النص الذي تقرأه، وقد تعلمناها جميعا في المدارس.

لكن الشعر فن موسيقي يتعلق بما نسمعه لا ما نراه مكتوبا.. لهذا، فالقاعدة الأساسية في الكتابة العروضية هي أن نكتب ما نسمعه فقط، فنحذف أيّ حروف غير منطوقة، ونضيف كل الأصوات غير المكتوبة.. فلو أخذنا الجملة التالية كمثال:

ذهب محمّد إلى المدرسة راكبا الدرّاجة

هذه الجملة تكتب عروضيا كالتالي:

ذهب محمْمَدن إللْ مدرسة راكبنِدْ درْراجة

حيث تلاحظ ما يلي:

- الحرف المشدّد (الذي عليه شدّة) ينطق حرفين، أولها ساكن والثاني متحرك، لهذا تحولت "محمّد" إلى "محمْمد" و"درّاجة" إلى درْراجة".
- عند التقاء حرفي علة ساكنين لا ينطقان، لهذا تحولت "إلى المدرسة" إلى "إللْ مدرسة" بحذف الألف اللينة من نهاية "إلى" وألف الوصل من بداية "المدرسة".
- التنوين يكتب نونا ساكنة لهذا تحولت "محمّدٌ" إلى "محمْمدنْ".
- عند التقاء حرف ساكن بألف الوصل يُحرك الحرف الساكن وتُحذف ألف الوصل، لهذا سنحول "راكبا الدراجة" مبدئيا إلى "راكبنِلْ درْراجة"... لكن...

- تذكّر أن أل التعريف لها طريقتان في النطق، فقد تكون لاما قمرية تنطق لاما عادية مثل "المدرسة"، أو لاما شمسية لا تنطق بل يشدد الحرف التالي لها مثل "الدراجة" التي ننطقها "ادْدراجة".. هذا هو السبب الذي جعل "راكبنِلْ درْراجة" تتحول إلى "راكبنِدْ درْراجة".

وهناك المزيد من القواعد التي سنتعرف عليها في هذا الفصل ونحن نكتب الكود الذي يحول الكتابة الإملائية إلى كتابة عروضية.. لكن أهم ملاحظة تعنينا هنا هي أننا لا نستطيع أن نفعل هذا بشكل صحيح إذا لم يكن النص مُشكّلا.. والحد الأدنى من التشكيل الذي نحتاج إليه هو وضع علامات الشدة والسكون والتنوين، ووضع الحركات على بعض حروف الياء والواو إذا كانت متحركة.. وهذا معناه أننا:

- سنفترض أن الحرف متحرك (عليه فتحة أو ضمة أو كسرة) ما لم تكن عليه سكون أو شدة أو تنوين.
- وأن حروف الياء والواو ساكنة (سواء نطقت ساكنة مثل "لوْم" أو ممدودة مثل "يلعبُون") إلا إذا جاءتا في أول الكلام مثل "يذهب" و "وقف"، أو كانت عليهما حركة مثل "ليَذهب" و "ووَقف"، أو إذا كان الحرف السابق لهما ساكنا مثل "مشْية" و "هفْوة" .

هذه التسهيلات ستجعل مستخدم البرنامج يضيف تشكيلات قليلة للغاية.. لاحظ أن الوضع المثالي هو أن نجعل البرنامج يشكل الكلمات تلقائيا، ولكن هناك عدة عوائق:

١- أن هذا يحتاج إلى معجم وبرنامج تحليل صرفي.. والحقيقة أنني أضفتهما لنسخة البرنامج القديمة، لكن من الصعب شرح هذا هنا.

٢- أننا حتى بعد امتلاك برنامج تحليل صرفي سنصطدم بحقيقة أن تشكيل الكلمة له عدة احتمالات، ومع زيادة عدد الكلمات في بيت الشعر، سيكون أمامنا شجرة احتمالات كبيرة للتشكيلات المحتملة، وهذا يحتاج إلى برنامج تشكيل آلي لفك الالتباس والوصول إلى التشكيل الأكثر احتمالا في هذا السياق، بناء على خبرات إحصائية من ملايين القصائد الشعرية، وهذا جهد يحتاج إلى شركة

بحجم شركة RDI التي أنتجت برنامج فصيح للتشكيل الآلي وفك الالتباس.

٣- أننا حتى لو تجاوزنا العقبتين السابقتين، فسنصطدم بمشكلة التشكيل الإعرابي (النحوي) لنهاية كل كلمة، وهذا يحتاج إلى محلل إعرابي، وهو الأمر الأعقد في سلسة التحديات لأن الإعراب فرع المعنى، والشعر خصوصا مليء بالاستعارات والصور الخيالية والتقديم والتأخير، حتى إن القارئ العادي قد يعجز عن فهم مراد الشاعر بدون شرح، فما بالك ببرنامج حاسوبي؟!

٤- فإذا افترضنا أن لدينا شجرة احتمالات، وفكرنا أن ندرس وزن كل احتمال ونأخذ المسارات التي تجعل معظم أبيات القصيدة موزونة على نفس البحر، فقد يستغرق هذا وقتا طويلا جدا لا لزوم له، ليكتشف المستخدم في النهاية أن إضافة بضع تشكيلات هنا أو هناك هو الحل الأسهل والأسرع!

لكل هذا سنلتزم في هذا البرنامج بقواعد التشكيل البسيطة المذكورة.. في الغالب سيضيف المستخدم تشكيلا واحدا على كل كلمة، ومع بعض التسهيلات الأخرى، لن يحتاج إلى إضافة أي تشكيلات على بعض الكلمات.

ملحوظة: في البرنامج القديم، استفدت من المحلل الصرفي في استنتاج كل التشكيلات المختلفة للكلمة، ثم حذفت الحركات التي لا تؤدي إلى وزن عروضي مختلف لأستبعد بعض الصيغ التي لن تؤثر في النتيجة، ومن ثم إذا ضغط المستخدم الكلمة بزر الفأرة الأيمن أعرض له ثلاثة أو أربعة احتمالات لتشكيل الكلمة (تختلف عن بعضها بوجود شدة أو سكون على بعض الحروف)، بحيث يمكنه اختيار التشكل الذي يناسبه.. لكن هذا بالطبع لا يشمل إعراب نهاية الكلمة، وعليه أن يضيف أي تنوين أو سكون على الحرف الأخير إن لزم الأمر.. لكن لن نتطرق إلى شرح هذا هنا حتى لا يتضاعف حجم الكتاب.

والآن، دعونا نبدأ العمل في برمجة الكتابة العروضية بلغة الرينج على بركة الله.

معالجة الكلمات الخاصة إملائيا:

قبل أن نبدأ في تطبيق قواعد الكتابة العروضية، هناك حالات خاصة يجب أن نعالجها أولا.. هذه الحالات تنقسم إلى مجموعتين:

المجموعة الأولى:

تسهيلات صغيرة سنضيفها لإراحة المستخدم لتقليل عدد التشكيلات التي يكتبها، مثل:

- إضافة سكون في نهاية الكلمة التي تتكون من حرفين (مثل "من" و"عن").. هذا لن يشمل أي حرف عطف أو جر يضاف للكلمة، لأن هذا قد يدخلنا في احتمالات كثيرة في تحليل الكلمة.. لهذا يجب أن يضع المستخدم السكون بنفسه على "فمن" (فقد تكون فَمَنَّ الله عليه) و "وعن" (فقد تكون وَعَنَّ له أن يفعل كذا).. إلخ.. وسنستثني أيضا بعض الكلمات الثنائية التي تكون متحركة الآخر أو تحتمل التحريك والتسكين لهذا سنتركها بدون تدخل، مثل:

Twos = ["لم"، "أب"، "دم"، "أخ"، "حم"، "يك"، "مع"]

وسنستثنى الكلمة "طه" أيضا لأنّ لها وضعا خاصا فهي تستخدم كاسم لبعض الأشخاص وتنطق طاها، لهذا لم نُضِفها للمصفوفة السابقة، وسنعالجها مع المجموعة الثانية.

- إضافة فتحة إلى نهاية "هو" و"هي" وبعض صيغهما التي يدخل عليها الواو أو الفاء أو اللام، فبدون هذه الفتحة ستعتبر الواو والياء ساكنتين:

HoaHea = ["أهي"، "أهو"، "ففي"، "ففهو"، "وهي"، "وهو"، "هي"، "هو"]

وإن احتاج المستخدم لتسكين نهاية "هو" أو "هي" في القافية مثلا، فسيكون مضطرا لإضافة سكون عليها.. هذا ليس النطق الدقيق في هذه الحالة، لكنها حالة نادرة جدا، ولن تزعج معظم المستخدمين.

- الكلمات المئوية مثل (مائة، المائة، مائتان، مائتي، ثلاثمائة..... إلخ)، تنتمي إلى المجموعة الثانية من الكلمات الخاصة، لأن بها ألفا زائدة يجب حذفها، لكن بملاحظة بسيطة، سنكتشف أنه أيا كانت السوابق واللواحق التي تدخل عليها، فلا

توجد كلمة عربية أخرى يمكن أن تحتوي على الحروف "مائة" أو "مائت"، فنحن حتى لا نستخدم اسم الفاعل من الفعل "مات" وإنما نقول "ميت" بدلا من "مائت".. لهذا من الآمن تماما أن نحول الحروف "مائة" و "مائت" إلى "مئة" و "مئت" إن وجدناها في أي كلمة.

ملحوظة:
كثيرون وأنا منهم نكتب "مئة" بشكل طبيعي، فلا توجد أي ضرورة لإضافة الألف.. كان عرب الجاهلية يكتبون الحروف بلا نقاط ولا همزات، فكان من الصعب عليهم التفريق بين مئة و منه بدون وضع الألف، (نفس الحال في أولئك وإليك بدون النقاط والهمزات، لهذا أضافوا الواو لتمييزهما)، لكن بعد وضع النقاط على الحروف وإضافة التشكيل، لم يعد لذلك أي هدف سوى تعقيد الكود في برنامجنا ☺.

تعال نكتب الكود الذي يعالج هذه المجموعة الآن.. تذكر أننا في الفصل السابق قد قطّعنا القصيدة إلى سطور، وقطعنا السطور إلى كلمات.. مهمتنا إذن أن نكتب دالة (ولنُسمِّها FixSpWords) لتصحيح الكلمات الخاصة.. هذه الدالة ستستقبل معاملا واحدا فقط (وليكن اسمه aWord)، يحمل الكلمة التي نتعامل معها.. هذا هو الشكل العام لهذه الدالة:

```
Func FixSpWords(aWord) {
  // كود معالجة المجموعة الأولى
  // كود معالجة المجموعة الثانية
}
```

والآن لننظر إلى كود معالجة المجموعة الأولى من الكلمات الخاصة، وسنبدأ بحذف ألف مائة ومائت:

```
pos = aWord.IndexOf("مائ", 1)
if pos > 0 and pos < aWord.Length - 2
  hrf = aWord[pos + 3]
  if hrf = "ة" or hrf = "ت"
    return Word.RemoveAt(pos + 1, 1)
  end
end
```

في هذا الكود استخدمنا الوسيلة IndexOf للبحث عن الحروف "ماىٔ" في النص، ووضع نتيجة البحث في المتغير pos، الذي يجب يكون أكبر من صفر (أي أننا وجدنا ما نبحث عنه)، ويجب كذلك أن يكون هناك حرف واحد على الأقل بعد الحروف "ماىٔ"، وهذا هو سبب الشرط:

pos < aWord.Length – 2

تذكر أن pos سيشير إلى موضع الحرف "م" في النص، وأن بعده الحرفين "اىٔ"، لهذا لو كانت الحروف "ماىٔ" في نهاية النص، فسيكون موضع الميم أصغر من طول النص بحرفين، لذا نحتاج أن يكون pos أصغر من (طول النص – ٢).

ولكن لماذا نحتاج لهذا التأكيد؟

السبب أننا سنتأكد أن الحرف التالي لـ "ماىٔ" هو "ة" أو "ت"، فهناك كلمات أخرى تحتوي على "ماىٔ" مثل ("مائه" بوضع الضمير هاء في نهاية الكلمة ماء، و"مائز"، و"مائن" من المين أي الكذب)، وسيحدث خطأ لو حاولنا قراءة حرف في موضع ليس موجودا في النص.

إذن فيجب أن نفحص الحرف الموجود في الموضع الثالث بعد الميم:

hrf = aWord[pos + 3]

ونتأكد أنه تاء مربوطة أو مفتوحة، فإن تحقق هذا، فسنستخدم الوسيلة RemoveAt لحذف الحرف التالي للميم (وهو الألف) ونعيد الكلمة المعدلة باعتبارها ناتج الدالة FixSpWords، فلم تعد هناك ضرورة لفحص أي حالات أخرى بعد أن عثرنا على ضالتنا:

```
if hrf = "ة" or hrf = "ت"
  return Word.RemoveAt(pos + 1, 1)
end
```

تذكر أن الدالة aString.RemoveAt تستقبل معاملين: موضع بداية الحذف، وعدد الحروف التي تريد حذفها بدءا من هذا الموضع.

أما إذا لم نعثر على هذه الحالة (ولم تصل رينج إلى السطر الذي فيه الأمر return) فهذا معناه أنها ستواصل تنفيذ الكود التالي.. هذا هو السبب أننا لا نحتاج إلى استخدام

المقطع else في جمل الشرط التي تحتوي على return، لأن الأمر return ينهي تنفيذ الدالة في الحال، وبالتالي كل ما يليه يعتبر كأنه موجود في مقطع else لأنه لن يُنفَّذ إلا إذا فشل الشرط!

الخطوة التالية هي أن نضع الفتحة على "هو" و"هي" ومشتقاتهما.. لفعل هذا، سنبحث في المصفوفة HoaHea عن الكلمة الحالية، فإن كانت موجودة (ناتج البحث أكبر من صفر)، فسنضيف فتحة إلى نهاية الكلمة ونعيدها كناتج للدالة FixSpWords:

```
if find(HoaHea, aWord.Text) > 0
  return aWord + Letters.Fat7ah
end
```

لاحظ أن الدالة find تبحث في المصفوفة عن نص عادي، وأن المتغير aWord يحمل نصا من النوع aString، لهذا حولناه إلى نص عادي باستخدام الخاصية Text.. لقد أضفت هذه الخاصية للفئة aString لأنها أكثر اختصارا من الوسيلة ()ToString، وكلاهما يعيد النص الذي تحمله الفئة aString.

نصل الآن إلى الكلمات الثنائية التي نريد إضافة سكون عليها.. يجب أولا أن نتأكد أن الكلمة لا تنتهي بعلامات تشكيل، (فلو أضاف المستخدم تشكيلا على الكلمة فلن نغيّره).. المشكلة أن التشكيل قد يكون علامة واحدة مثل الفتحة، أو علامتين مثل الشدة وتنوين الضم (كما في "ذمٌّ") لهذا نحتاج لدالة تخبرنا بعدد علامات التشكيل، سنسميها LastTashkeelLength وفكرتها بسيطة:

سنستخدم متغيرا اسمه n نضع فيه عدد علامات التشكيل، وستبدأ قيمته بصفر، وسنكتب حلقة تكرار عكسية تمر عبر حروف الكلمة من آخر حرف إلى أول حرف (سنستخدم الخطوة -١ في حلقة التكرار For Next)، ونرسل كل حرف إلى الدالة IsTashkeel، فإن أعادت True نزيد قيمة n بمقدار واحد، وإن أعادت False نكون قد وصلنا إلى آخر حرف أبجدي في الكلمة (ليس علامة تشكيل)، وهنا علينا مغادر الدالة كلها وإعادة قيمة المتغير n:

```
Func LastTashkeelLength(aWord)
   L = len(aWord)
   n = 0
   For i = L To 1 Step -1
      if IsTashkeel(aWord[i])
         n++
      else
         return n
      end
  next
EndFunc
```

الآن يمكننا أن نعود إلى الدالة FixSpWords ونستخدم الدالة التي كتبناها:

```
n = LastTashkeelLength(aWord)
```

وسنعرف متغيرا اسمه word سنضع فيه الكلمة بعد إزالة التشكيل منها، لأن كل المقارنات التي سنجريها بعد ذلك ستكون بدون تشكيل، وسنضع طولها في متغير:

```
word = aWord.RemoveTashkeel( )
wordLength = word.Length
```

وستجد كود الدالة RemoveTashkeel في الفئة aString وفكرتها بسيطة لا تحتاج لشرح.

تذكر أن مهمتنا الآن معالجة الكلمات المكونة من حرفين، لكن علينا أولا استبعاد كلمة "طه" وأي كلمة تنتهي ب همزة مد "آ":

```
If word = "طه" Return new aString("طاها") End
If not word.EndsWith("آ")
  // ……
End
```

سنكتب كود معالجة الكلمات الثنائية في موضع التعليق في الكود السابق:

إذا كانت الكلمة الأصلية لا تنتهي بعلامة تشكيل (n = 0)، والكلمة بدون تشكيل تتكون من حرفين فقط، فسنبدأ في معالجة الكلمات الثنائية، كالتالي:

- نبحث في مصفوفة الكلمات الثنائية المستثناة Twos عن الكلمة بدون تشكيل، ولو وجدناها نعيدها كما هي كناتج للدالة FixSpWords.
- وإلا، لو كانت الكلمة "هو" أو "هي" نضيف فتحة إلى نهايتها.. قد يحدث هذا لأن

المستخدم كتب "هُو" فلم نعثر عليها في المصفوفة HoaHea بسبب الضمّة، ولكننا سنعثر عليها هنا لأننا تخلصنا من التشكيل.

- فإن لم تحدث الحالتان السابقتان، فنحن الآن نتعامل مع كلمة ثنائية من التي نريد أن نضيف إليها علامة السكون، بشرط ألا تكون منتهية بحرف علة ("ا"، "ى"، "ي"، "و").

أما إذا كانت الكلمة (بدون تشكيل) تتكون من حرف واحد، أو من ثلاثة حروف (لكنها ليست إحدى مشتقات هو وهي، فقد يكون المستخدم كتبها بالتشكيل أيضا)، فسنعيد الكلمة الأصلية كما هي.. هذا هو الكود:

```
if word = "طه" return new aString("طاها") end
if word.RightStr(1) != "آ"
  if n = 0 and wordLength = 2
    if find(Twos, word.Text) > 0
      return aWord
    elseif word = "هو" or word = "هي"
      return word + Letters.Fat7ah
    elseIf not word.EndsWithAny(["ا"، "ى"، "ي"، "و"])
      return word + Letters.Sokon
    end
  elseif wordLength = 1 or
      (wordLength = 3 and find(HoaHea, word.Text) > 0)
    return aWord
  end
end
```

وهكذا نكون قد انتهينا بحمد الله من معالجة المجموعة الأولى من الكلمات الخاصة.

المجموعة الثانية من الكلمات الخاصة:

هي كلمات تشذّ عن قواعد الإملاء المتبعة حاليا، بسبب وجود حروف زائدة (مثل أولئك) أو ناقصة (مثل هذا، الذي، إله).. ولتمثيل هذه الكلمات برمجيا، سنضع بيانات كل كلمة في مصفوفة كما في هذا المثال:

```
["لكنْ"، "لاكِنْ", False]
```

هذه المصفوفة تحتوي على المعلومات التالية بالترتيب:

١. قيمة منطقية (True/False) تخبرنا هل يمكن أن تضاف لواحق لنهاية الكلمة أم لا.. في المثال السابق لا يمكن إضافة أي حرف بعد "لكنْ" (ساكنة النون).. وقد اخترت وضع هذه القيمة في بداية المصفوفة لأن محرر رينج سيعرض النصوص بشكل مربك لو كتبت نصا عربيا أولا ثم أتبعته بنصوص إنجليزية.. ولولا هذا، كنت سأفضل أن أضيف هذه المعلومة في نهاية المصفوفة.

٢. الكتابة الإملائية المتعارف عليها للكلمة (مثل "لكنْ").. لاحظ أن عرض كلمات عربية في نص إنجليزي يجعل الأمور مربكة، لهذا يبدو لك العنصر "لكنْ" كأنه آخر عنصر في المصفوفة، لكنه في الحقيقة مكتوب في الموضع الثاني.

٣. الكتابة الإملائية القياسية للكلمة (مثل "لاكنْ").

والفكرة ببساطة أن نفحص كل كلمة في النص، فإن كانت إحدى الكلمات الخاصة (مثل "لكنْ") فسنضع بدلا منها الكتابة القياسية لها (مثل لاكنْ).

هذه هي طريقة التفكير البسيطة التي سنبدأ بها، لكننا بالطبع سنطورها أكثر لمراعاة كل الحالات المحتملة، كما سنرى بعد قليل.

وأول هذه الحالات، هي وجود بعض الصيغ الشبيهة للكلمات الخاصة.. على سبيل المثال: (الله، اللهم)، (إله، الإله).. وسنعالج هذه التشابهات بطريقة خاصة:

فحينما كتبت الكود أول مرة، كنت أتعامل مع كل هذه التشابهات كأنها كلمات مستقلة، وهذا جعل الكود يستغرق وقتا ملموسا لتنفيذه، وهذا بالتأكيد يقلل من كفاءة البرنامج.. لهذا أعدت التفكير في الأمر، وقررت تقسيم الكلمات الخاصة إلى مجموعات، بحيث تحتوي كلمات كل منها على بعض الحروف المشتركة، وهكذا أستطيع البحث في الكلمة عن هذه الحروف أولا، فإن وجدتها أبدأ في فحص عناصر المجموعة، وقد قلل هذا عمليات البحث بشكل كبير وجعل تنفيذ الكود أسرع بكثير.. وهذه المجموعات هي:

```
_sp1 = ["لذ",
   [False, "الذي"، "اللذي"],
   [False, "الذى"، "اللذي"],
   [False, "الذين"، "اللذين"]
]
```

```
_sp2 = ["لت",
   [False, "الّلتي" ،"التي"],
   [False, "الّلتي" ،"التى"]
]
_sp3 = ["لله",
   [False, "الّلاه" ،"الله"],
   [False, "الّلاهمّ" ،"اللهم"]
]
_sp4 = ["أول",
   [False, "أُلائك" ،"أولئك"],
   [False, "أُلائكمْ" ،"أولئكم"],
   [False, "أُلاء" ،"أولاء"],
   [False, "أُلات" ،"أولات"],
   [False, "أُلو" ،"أولو"],
   [False, "أُلي" ،"أولي"]
]
_sp5 = ["هذ",
   [False, "هاذا" ،"هذا"],
   [False, "هاذِهِ" ،"هذه"],
   [False, "هاذان" ،"هذان"],
   [False, "هَاذَيْن" ،"هذين"],
   [False, "هَاذِي" ،"هذي"]
]
_sp6 = ["ذلك",
   [False, "ذَالِك" ،"ذلك"],
   [False, "ذَالِكما" ،"ذلكما"],
   [False, "ذَالِكمْ" ،"ذلكم"],
   [False, "ذَالِكنّ" ،"ذلكن"]
]
_sp7 = ["هأن",
   [False, "ها أنا" ،"هأنا"],
   [False, "ها أَنَذا" ،"هأنذا"]
]
_sp8 = ["لكن",
    [False, "لاكِنّا" ،"لكنا"],
    [False, "لاكِنّني" ،"لكنني"],
    [False, "لاكِنْ" ،"لكن"],
    [True, "لاكِنّ" ،"لكن"]
]
```

```
_sp9 = [""،
  [False, "طه"، "طاها"],
  [False, "عمرو"، "عَمْر"],
  [False, "هؤلاء"، "هاؤلاء"]
]
_sp10 = [""،
  [False, "داود"، "دَاوُود"],
  [True, "طاوس"، "طَاوُوس"],
  [False, "الطاوس"، "اطْطَاوُوس"],
  [True, "إله"، "إلاه"],
  [False, "الإله"، "الْإلاه"],
  [False, "رحمن"، "رَحْمان"],
  [False, "الرحمن"، "ارْرحْمان"]
]
```

لاحظ أننا لن نستخدم أسماء هذه المجموعات في الكود، لهذا لم أعطها أسماء واضحة.. كل ما سنفعله بهذه الأسماء هو وضعها كعناصر داخل مجموعة الكلمات الخاصة:

```
SpWords = [ _sp1, _sp2, _sp3, _sp4, _sp5,  _sp6,  _sp7,
            _sp8, _sp9,  _sp10 ]
```

مع العلم أننا نستطيع تعريف المجموعات مباشرة داخل المصفوفة SpWords، لكن مرة أخرى: وجود كلمات عربية سيجعل محرر رينج يعرض النصوص بشكل فوضوي مربك، لهذا استخدمت هذا الحل، كما أن هذا التقسيم يجعل شرح الكود أسهل.

وبالنظر لمجموعات الكلمات الخاصة يمكنك ملاحظة ما يلي:

- أول عنصر في المجموعة نص يمثل الحروف المشتركة بين كلمات المجموعة.
- المجموعتان التاسعة والعاشرة تحتويان على كلمات متنوعة لا توجد بينها حروف مشتركة، لهذا جعلنا أول عنصر في كل منهما نصا فارغا "".. وتختلف المجموعة العاشرة عن التاسعة في أن كلمات المجموعة العاشرة يمكن أن تلحق بها ياء النسب كما سأوضّح لاحقا.
- أراعي أن بعض المستخدمين (خاصة المصريين) قد يكتبون الياء المتطرفة بدون نقطتين تحتها (يستخدمون الألف اللينة ى بدلا من الياء ي) لهذا استخدمت

الطريقتين كما في "الذي" و"الذى"، لكن لم أفعل هذا مع اسم الإشارة "هذي" لأن "هذى" هو فعل ماضٍ (هذى يهذي هذيانا)، وفي هذه الحالة يجب استخدام "هذي" لو كان المقصود اسم الإشارة، وإلا فعلى المستخدم أن يضيف الألف بنفسه: "هاذى".

جميل.. تبدو الأمور سهلة حتى الآن.. لكن... وآه مما يأتي بعد لكن!

في الوضع البسيط، لو وجدنا أي كلمة خاصة، فسنستبدل بها الكتابة الصحيحة لها، لكن الكلمة قد يسبقها بعض حروف الجر والعطف والاستفهام أو تواليف منها، مثل: (والله، وللرحمن، أفلهذا، وكالذين).. كما أن الكلمة قد يلحق بها بعض الضمائر، مثل: (إلهنا، طاوسكم، لكنهم).. وقد يحدث الأمران معا مثل: (فإلهنا، أفطاوسكم، ولكنهم).. لهذا يجب أن نضيف السوابق prefixes واللواحق Suffixes في الاعتبار:

```
prefixes = [
   ["و"، "ف"، "أ"، "ب"، "ك"، "ل"],
   ["فو"، "وأ"، "وب"، "وك"، "ول"، "فأ"، "فب"، "فك"، "فل"، "أو"، "أف"، "أب"، "أك"، "أل"],
   ["أفب"، "أفك"، "أفل"، "أوب"، "أوك"، "أول"]
]
Suffixes = [
   ["ي"، "هُ"، "ك"],
   ["نا"، "همْ"، "هنّ"، "كمْ"، "كنّ"],
   ["هما"، "كما"]
]
AbsSuffixes = [
   ["ي"، "ه"، "ك"],
   ["نا"، "هم"، "هن"، "كم"، "كن"],
   ["هما"، "كما"]
]
```

لاحظ أنني جمعت السوابق واللواحق في مصفوفات فرعية على حسب طولها لتسريع عملية البحث عنها قليلا كما سنرى بعد قليل.. لكن لا يوجد ما يمنع من كتابتها معا كمصفوفة واحدة مباشرة.

لاحظ أيضا أن AbsSuffixes تحتوي على نفس عناصر المصفوفة Suffixes لكن

بعد حذف التشكيل منها.. في هذا البرنامج، سأستخدم دائما البادئة abs (اختصارا للكلمة absolute بمعنى مطلق) للإشارة إلى الكلمات المجردة من التشكيل.

نقطة أخرى: الكلمات المعرفة بأل لا تدخل عليها اللواحق السابقة، لهذا ستجد مصفوفاتها جميعا تبدأ بالقيمة False، لكنّ هناك نوعا آخر من اللواحق يمكن أن يدخل عليها وهو ياء النسب (مثل "الأمر الإلهيّ" و"الأوامر الإلهية") ومثنياتها وجموعها.. وهذه اللواحق تدخل أيضا على صيغة النكرة من هذه الكلمات.. لهذا سنعامل هذا النوع من اللواحق معاملة خاصة.. ولو نظرت إلى المصفوفة SpWords، فستلاحظ أن هذه الحالة تصلح فقط مع آخر ٧ كلمات (داود، طاوس، الطاوس، إله، الإله، رحمن، الرحمن).. هذا هو السبب في أنني وضعت هذه الكلمات في المجموعة العاشرة لأطبق عليها هذه الحالة.

الآن، علينا تعريف لواحق النسب:

```
RelativeSuffixes = [
  ["يّ"], ["ىّ"]
  ["يّة"، "يّا"، "يُّو"، "يّي"],
  ["يّان"، "يّون"، "يّين"، "يّتا"، "يّتَيْ"، "يّات"],
  ["يّتان"، "يّتَيْن"]
]
AbsRelativeSuffixes = [
  ["ي"], "ى"
  ["ية"، "يا"، "يو"، "يي"],
  ["يان"، "يون"، "يين"، "يتا"، "يتي"، "يات"],
  ["يتان"، "يتين"]
]
```

الحقيقة أن الأمور يمكن أن تتعقد أكثر من هذا، لأننا بعد ضمائر النسب، يمكن أن نضيف ضمائر الملكية والمخاطب، كأن تقول "طاوسيّتي" (كناية عن الخيلاء مثلا)، لكن مثل هذا الاستخدام سيكون نادرا جدا، وفي معظم الأحيان لن تستوعبه أوزان الشعر (مثل "طاوسّيتاكما") فلا داعي إذن لأن نعقد الأمور على أنفسنا!

لدينا الآن فكرة واضحة عما نريد فعله.. نريد إذن أن نكتب الكود الذي ينفذ هذا، وهو كود سيحتاج منا بعض التفكير والإبداع.. فلنبدأ على بركة الله:

فـي البدايـة، هنـاك معلومتـان سنضـعهما فـي متغيـرين، لأننـا سنسـتخدمهما عـدة مـرات داخل حلقات التكرار:

المعلومـة الأولـى هـي عدد مجموعات الكلمات الخاصـة (طـول المصـفوفة SpWords) وقد حفظناها في المتغير spGroupsCount:

spGroupsCount = len(SpWords)

بشكل عام، أنصحك ألا تستخدم الدالة Len لمعرفة طول المصفوفة في سطر تعريف حلقـة التكـرار For، حتـى لا يتكـرر اسـتدعاء الدالـة Len بعـدد مـرات دورات حلقـة التكرار.. واستخدم بـدلا مـن هذا متغيـرا يحمـل طـول المصـفوفة، فهذا يجعلهـا أسـرع، خاصـة إذا كنـت تتعامـل مـع مصـفوفات طويلـة جدا.. وفـي مثـل هذه الحـالات تجنـب أيضـا استخدام For in، لأنها تعيد حساب طول المصفوفة في كل لفّة.. السبب في هذا هو أن طول المصفوفة يمكن أن تتغير داخل حلقة التكرار بإضافة أو حذف بعض العناصر، وهو أمر قليل الحدوث في البرمجة، لهذا إن كنت متأكدا أنك لن تغير عدد عناصـر المصـفوفة، وكانـت هذه المصـفوفة طويلـة جدا، أو سيعاد تنفيذ حلقـة التكرار لعدد كبير من المرات، فاستخدم الحلقة For بدلا من For In، واستخدم متغيرا يحمل طـول المصـفوفة.. وفـي حالتنـا هذه، نحـن نتعامـل مـع مصـفوفات قصـيرة، لكننـا سنكرر استدعاء الدالـة FixSpWords لكل كلمـة فـي القصـيدة.. لـو أننـا نتعامـل مـع قصـيدة تتكـون مـن ١٠ أبيـات فقـط، وكـل بيـت يتكـون مـن ١٠ كلمـات فقـط، فهـذا يعنـي أننـا سنكرر حلقـة التكرار ١٠٠ مرة، وكل حلقـة ستدور ١٠ مـرات، وداخلهـا حلقـة داخليـة لمجموعة الكلمات الخاصة (٥ كلمات في المتوسط)، أي أن بعض سطور الكود ستنفذ حوالي ٥ آلاف مرة!.. لاحظ أنـه بدون تقسيم الكلمـات الخاصـة إلـى مجموعـات كانـت بعض سطور الكود ستنفذ ٤٠ ألف مرة لمعالجة قصيدة قصيرة طولها ١٠ أبيات، وهذا فـي خطـوة واحـدة تمهيديـة قبـل بـدء تحليـل القصـيدة!.. طبعـا علـى الأجهـزة الحديثـة سيستغرق كل هذا بضع ثوانٍ، لكن التأثير الإجمالي في معالجة القصيدة قد يصل إلى دقيقة أو اثنتين، وهو وقت طويل بالنسبة للمستخدمين اليوم.. كل تحسين سيحدث فارقا في النهايـة، لهذا فإن مجرد تقسيم الكلمـات الخاصـة إلـى مجموعـات تشـترك فـي بعض

الحروف وفر أكثر من ٨٥% من وقت تنفيذ هذه الدالة، والتحسينات الصغيرة الأخرى مثل حفظ بعض القيم خارج حلقات التكرار وفر ٥% إضافية!.. أي أننا نستطيع بقليل من التفكير البرمجي السديد تسريع البرنامج بنسبة ٩٠%!

المعلومة الثانية التي سنحتفظ بها هي تشكيل آخر حرف في الكلمة، وبما أننا نعرف طول علامات التشكيل (وضعناه سابقا في المتغير n)، يمكننا أن نحصل على علامات التشكيل نفسها (باستخدام الوسيلة aString.Right التي تعيد إلينا العدد الذي نريده من الحروف من نهاية النص) ونحفظها في متغير اسمه lastTashk:

```
lastTashk = aWord.Right(n)
```

والآن، فلنكتب حلقة التكرار التي ستمر عبر مجموعات الكلمات الخاصة:

```
For gpIdx = 1 to spGroupsCount
  SpGroup = SpWords[gpIdx]
  if gpIdx < 9
    if word.IndexOf(SpGroup[1], 1) = 0 Loop end
    if gpIdx = 6
      if aWord.IndexOfAny(["ذْ", "لَ"], 1) > 0 Return aWord End
    ElseIf gpIdx = 8
      k = aWord.Right(2)
      if k = "كَ" or k = "كِ"
          pos = aWord.RemoveLast(2).IndexOfAny(NotLaken, 1)
      else
          pos = aWord.IndexOfAny(NotLaken, 1)
      end
      if pos > 0 return aWord end
    end
  end

  // كود معالجة كل مجموعة

next
return aWord
```

لاحظ ما يلي:

- كما نفعل دائما، عندما نعثر على كلمة خاصة نعيد الكلمة المقابلة لها كنتيجة للدالة FixSpWords، أي أننا سنخرج من الدالة مباشرة.. وهذا معناه أننا إذا

وصلنا إلى السطر التالي للكلمة Next فهذا معناه أننا لم نعثر على أي كلمات خاصة، ولهذا سنعيد الكلمة الأصلية aWord كما هي.

- في بداية كل لفة في حلقة التكرار، سننسخ المجموعة الحالية في المصفوفة إلى متغير، ليسهل علينا إعادة استخدامه.. هذا سيجعل الكود أوضح لكن انتبه أن عملية النسخ نفسها قد تضيع القليل من الوقت خاصة مع تكرارها آلاف المرات كما شرحنا.. لكل شيء ثمن كما تعلم:

```
SpGroup = SpWords[gpIdx]
```

- لن نفحص الصيغ الخاصة بالمجموعة الحالية إلا إذا كانت الكلمة تحتوي على الحروف المشتركة بين هذه الصيغ (مثل "لكن").. تذكر أننا وضعنا هذه الحروف في أول عنصر في المجموعة، ما عدا المجموعتين التاسعة والعاشرة فقد وضعنا نصا فارغا في أول عنصر فيها.. لهذا سيكون الشرط كالتالي:

```
if gpIdx < 9
  if word.IndexOf(SpGroup[1], 1) = 0 Loop end
```

في هذا الشرط إذا لم تكن الحروف المشتركة بين كلمت المجموعى موجودى في الكلمة التي نعالجها، فسننفذ الأمر Loop لتجاهل هذه المجموعة.. هذا الأمر يجعل رينج تقفز مباشرة إلى السطر Next لتنتقل إلى اللفة التالية في حلقة التكرار، وهذا يعني أننا تجاوزنا تنفيذ باقي كود حلقة التكرار في اللفة الحالية.

- هناك حالتان خاصتان أيضا تقعان في المجموعات الأقل من ٨، وهما تخصان "ذلك" (المجموعة رقم ٦)، و "لكن" (المجموعة رقم ٨).. فهناك بعض الكلمات الشبيهة ب "ذلك" (مثل "ذلّك" من الذلّ، و"فذْلك" من الفذلكة).. لهذا لو جدنا حرف ذال ساكنا ("ذْ") أو لاما مشددة ("لّ") في الكلمة التي نفحصها، فهذا معناه أنها ليست اسم الإشارة "ذلك" وسنعيد الكلمة كما هي ونخرج من الدالة:

```
if gpIdx = 6 and aWord.IndexOfAny(["ذْ", "لّ"], 1) > 0
    return aWord
end
```

تذكر أن الوسيلة IndexOfAny تستقبل معاملين: مصفوفة النصوص المراد البحث عنها، وموضع بداية البحث، وتعيد موضع بداية أول نص عثرت عليه.

هنا قد تسألني سؤالا:

لماذا نخرج من الدالة كلها لمجرد أن الكلمة تحتوي على "ذْ" أو "لّ"؟.. ألا يجب أن نفحص باقي الحالات أولا؟.. اللام المشددة مثلا وارد أن تظهر في كلمات خاصة أخرى.. فلماذا لم نستخدم Loop بدلا من Return؟

هنا يجب أن ألفت انتباهك إلى أن هذه الحالة تعتبر ElseIf للشرط:

if word.IndexOf(SpGroup[1], 1) = 0 Loop End

فالأوامر التي تجعل رينج تقفز من موضع إلى آخر مباشرة (مثل Retun وLoop وExit) تمنع تنفيذ السطور التالية لها إذا كان الشرط صحيحا، أي أن وصولنا إلى السطور التالية لها، يجعلنا متأكدين أن الشرط كان خاطئا.. فإذا كان الشرط السابق خاطئا، فهذا معناه أننا وجدنا الحروف المميزة للمجموعة في الكلمة التي نعالجها.. إذن فنحن هنا لم نعثر فحسب على "ذْ" و "لّ" في الكلمة aWord (المشكّلة)، بل عثرنا قبل هذا أيضا على "ذلك" في الكلمة Word (غير المشكلة).. لو جمعت هاتين المعلومتين معا، فستعرف أننا عثرنا على "ذْلك" أو "ذلّك"، ولا توجد كلمات خاصة أخرى يمكن أن تحتوي على هذه الحروف، لهذا سنوقف الفحص ونخرج من الدالة كلها، فهذه بالتأكيد كلمة عادية.

نفس هذا الأمر سينطبق على "لكن" إلا أن هذه الكلمة تحتاج لدقة أكبر في معاملتها، فمن الممكن أن تلتبس مع كلمات أخرى مثل "لَكُنَّ" (لك + نون النسوة) أو "لُكْنَ" (من الفعل الماضي لاك يلوك أي مضغ، مضافا إليه نون النسوة).. ولاستبعاد مثل هذه الاحتمالات سنسمح للمستخدم بوضع أي تشكيل على اللام أو الكاف للفت انتباه البرنامج إلى أنه لا يقصد لكن، مع ملاحظة أن اللام في "لَكِن" مفتوحة والكاف مكسورة، لهذا لو وضع أي تشكيل غير هذين على أي من هذين الحرفين (مثل "لُكن أو لكُن") فلن نعتبرها "لكن".. وحتى لا نفاجأ بأي حالات أخرى لكلمات شبيهة ب "لكن" لا نتوقعها، أو يفاجئنا المستخدم بتشكيل خاطئ، جمعت كل التشكيلات غير الممكنة لحروف "لكن" في هذه المصفوفة:

NotLaken = ["نُ","نَ","كّ","كُ","كَ","كْ","نّ","نُ","لِ","لْ"]

وكل ما سنفعله الآن هو التأكد أن الكلمة التي تحتوي على "لكن" بدون تشكيل، لا تحتوي على أي عنصر من عناصر المصفوفة NotLaken.. مع احتراس صغير في حالة اتصال الضمير "ك" بنهاية "لكن"، فقد يكتب المستخدم "لكنكَ" أو "لكنكِ".. لهذا لو وجدنا أحد هذين الحرفين في نهاية الكلمة، فسنستبعدهما من عملية البحث.. يمكنك الآن فهم وظيفة هذا الجزء من الكود:

```
k = aWord.Right(2)
if k = "كَ" or k = "كِ"
  pos = aWord.RemoveLast(2).IndexOfAny(NotLaken, 1)
else
  pos = aWord.IndexOfAny(NotLaken, 1)
end
if pos > 0 return aWord end
```

ملحوظة عن صيغ "لكن":

لو أعدت النظر إلى تعريف المجموعة الثامنة:

```
_sp8 = ["لكن",
  [False, "لكنا", "لاكِنّا"],
  [False, "لكنني", "لاكِنّني"],
  [False, "لكن", "لاكِنْ"],
  [True, "لكن", "لاكِنّ"]
]
```

فلا ريب أنك ستندهش من صيغ "لكن".. والسبب أن هناك اختزالا يحدث عند اتصال لكن بالضمير "نا"، فيمكن أن تكون "لكنّنا" (وهذه تغطيها اللواحق) أو "لكنّا" وهذه لا تغطيها اللواحق بسبب دمج النونين، لهذا أضفناها كحالة خاصة.. ونفس الأمر مع "لكنّي" (تغطيها اللواحق) و"لكنّني" التي أضفناها لأن اللواحق لا تغطيها بسبب وجود نون زائدة للوقاية عند الإضافة لياء المتكلم.

ويتبقى آخر عنصرين وسيشعرانك بالغرابة للحظة، فكلاهما "لكن" بدون أي تشكيل!

لو أمعنت النظر فستجد "لكن" الأولى لا تقبل اللواحق، لهذا سنفترض أنها "لكنْ" ساكنة النون فهي لا تقبل أي ضمائر، وهذا سيغطي حالات مثل ("لكنْ"، "ولكنْ")، فإن فشل هذا الاحتمال، فهذا معناه وجود لواحق وهذا يحدث فقط مع "لكنّ" مشددة النون، وهذا

سيغطي أي احتمالات متبقية مثل ("لكنّا"، ولكنّك، ولكنّهم... إلخ).
هذا يكشف لك أهمية تنظيم التفكير وترتيب الحالات بما يتناسب مع آلية الكود الذي كتبناه.. ولكن ماذا سيحدث لو كتب المستخدم "لكنّ"، ونحن نجري عمليات المقارنة هنا على الكلمات بدون تشكيل أي أن الشدة ستضيع، وهذا يعني أن البرنامج سيحولها إلى "لاكنْ" بنون ساكنة على عكس إرادة المستخدم؟
لا تقلق، وتذكر أننا احتفظنا بتشكيل الحرف الأخير في المتغير lastTashk، وسترى كيف سنستخدمه بعد قليل للمحافظة على تشكيل آخر حرف في الكلمة.

الآن نحن جاهزون لكتابة الكود الخاص بالمقطع:

كود معالجة كل مجموعة //

هذا الكود سيكون حلقة تكرار تمر عبر الكلمات الموجودة في المجموعة الخاصة الحالية، لتبحث عنها في الكلمة، وإن وجدتها تعوض عنها بالكتابة الإملائية الصحيحة.. سيكون الكود كالتالي:

```
spCount = len(SpGroup)
For idx = 2 to spCount
   spInfo = SpGroup[idx]
   canHasSuffex = spInfo[1]
   spWord = new aString(spInfo[2])
   spWordLength = spWord.Length
   newWord =  new aString(spInfo[3])
   pos = word.IndexOf(spWord, 1)
   if pos > 0
      عثرنا على الكلمة الخاصة //
   Elseif spWord.left(1) = "ا"
      لم نعثر على الكلمة الخاصة لكنها تبدأ بحرف الألف ولها وضع خاص //
   End
Next
if gpIdx < 9 return aWord End
```

لاحظ أن العداد idx يبدأ من ٢ وليس ١، لأننا نضع في الخانة الأولى الحروف المشتركة بين كلمات المجموعة.

وبعد انتهاء حلقة التكرار ووصولنا إلى السطر التالي للأمر Next، أضفنا شرطا

عجيبا: إذا كان عداد حلقة التكرار الخارجية (عداد المجموعات الخاصة) أقل من ٩، فسنعيد الكلمة التي نفحصها كما هي ونغادر الدالة!

أعرف أنك ستصيح في دهشة: ولكن هذا سيمنع فحص المجموعتين التاسعة والعاشرة!

ولكني سأجيبك في ثقة: لا.. لن يحدث هذا أبدا!

السبب أننا لا ندخل حلقة التكرار الداخلية (كلمات المجموعة) إلا إذا كانت الكلمة تحتوي على الحروف المشتركة بكل مجموعة (مثل وجود "هذ" في الكلمة "هذا").. وبتأمل بسيط، ستجد أن الحروف الخاصة بالمجموعات الثماني الأولى مميزة ولا يوجد أي تداخل بينها، وبالتالي إذا وجدنا "هذ" في الكلمة لكن لم تنطبق عليها أي من كلمات المجموعة (كما لو كنا نفحص الكلمة يهذي مثلا)، فهذا معناه أن هذه ليست كلمة خاصة من هذه المجموعة، وفي نفس الوقت فإن احتواءها على الحروف المشتركة يضمن لنا كذلك أنها لا توجد في المجموعات الأخرى.

إذن فهذا الشرط سيختصر تنفيذ مقارنات كثيرة لا لزوم لها في مثل هذه الحالات.

الآن، فلندخل إلى كود حلقة التكرار:

في البداية عرّفنا بعض المتغيرات لنضع فيها بعض القيم التي سنكرر استخدامها في الكود.. فوضعنا في المتغير spInfo معلومات الكلمة الخاصة الحالية، الموجودة في المجموعة SpGroup في الخانة رقم idx (قيمة عداد حلقة التكرار في اللفة الحالية).

الآن spInfo يحتوي على مصفوفة من ثلاث خانات:

- الأولى spInfo[1] فيها True أو False وهي تخبرنا إن كان من الممكن إضافة لاحقة إلى الكلمة الخاصة أم لا، وقد وضعناها في المتغير canHasSuffex.
- الخانة الثانية spInfo[2] تحتوي على الكلمة الخاصة إملائيا وسنضعها في متغير اسمه spWord، ونخزن طولها في متغير اسمه spWordLength.
- الخانة الثالثة spInfo[3] تحمل الكلمة الصحيحة المقابلة للكلمة الخاصة، وقد وضعناها في متغير اسمه newWord.

بهذه الطريقة سيصير الكود أسهل، فنحن الآن نتعامل مع متغيرات لها أسماء واضحة،

بدلا من التعامل مع أرقام خانات لا نفهم معناها.

والآن، كل ما سنفعله هو البحث عن الكلمة الخاصة spWord داخل نص الكلمة (غير المشكلة) word، ونضع نتيجة البحث في المتغير pos:

pos = word.IndexOf(spWord, 1)

فإن كانت الكلمة الخاصة جزءا من الكلمة، فستكون قيمة المتغير pos أكبر من صفر، وإن لم يتم العثور عليها، فستكون قيمته صفرا.. لهذا كتبنا جملة شرط للتعامل مع هاتين الحالتين.

لكن السؤال الذي يثور هنا: هل هناك ما يجب أن نفعله لو لم نعثر على الكلمة الخاصة؟.. ما دمنا لم نجدها، فهذا يعني أنها كلمة عادية والمفروض ألا نفعل شيئا.

هذا صحيح، لكن...

هناك حالتان خاصتان تغيران شكل الكلمة الخاصة، هما دخول الهمزة واللام على كلمة تبدأ بألف.. فمثلا إذا دخلت اللام على كلمة "الإله" فستحولها إلى للإله (بحذف الألف)، وهناك حالة أكثر خصوصية من هذه الحالة، إذا دخلت اللام على لفظ الجلالة "الله" حيث نكتبه "لله" فكأننا حذفنا الألف واللام!

أما إذا دخلت الهمزة على كلمة تبدأ بألف، فتدمجان معا في همزة مد "آ"، كما في قوله تعالى (آلآنَ وَقَدْ عَصَيْتَ قَبْلُ وَكُنْتَ مِنَ الْمُفْسِدِينَ).

لهذا السبب لو كنا نفحص كلمة مثل "للرحمن" وبحثنا فيها عن الكلمة الخاصة "الرحمن" فلن نجدها!.. نحتاج إذن لمعالجة هاتين الحالتين الخاصتين في شرط منفرد، ولهذا كتبنا مقطع ElseIf والشرط الخاص به هو أن تبدأ الكلمة الخاصة بحرف الألف.. ولجعل الأمور أكثر تنظيما، نلاحظ أن حرف الألف يوجد في كلمات المجموعات الخاصة sp1_ و sp2_ و sp3_ وبعض كلمات المجموعة الخاصة sp10_.. هذا الترتيب سيجعلنا نكتب شرطا يقلل عدد المقارنات النصية، فنحن متأكدون أن الكلمة تبدأ بألف لو كان رقم المجموعة الخاصة أقل من ٤، أما المجموعة رقم ١٠ فبعض كلماتها لا تبدأ بألف، لذا يجب أن نتأكد أن أول حرف من كلماتها هو "ا" (ولاحظ أهمية وجود قوسين حول هذا الشرط المركب):

```
if pos > 0
   عثرنا على الكلمة الخاصة //
ElseIf gpIdx < 4 or (gpIdx = 10 and spWord.left(1) = "ا")
   لم نعثر على الكلمة الخاصة لكنها تبدأ بحرف الألف ولها وضع خاص //
End #If
```

الآن فلننظر ماذا سنفعل إن كانت الكلمة التي نفحصها تحتوي على كلمة خاصة:

بشكل عام سنحاول تقسيم الكلمة إلى ثلاثة أجزاء:

- بادئة سنضعها في متغير اسمه pfx (اختصارا للكلمة prefix).. ومن الممكن ألا توجد بادئة وهنا سيحتوي هذا المتغير على نص فارغ.
- الكلمة الخاصة، وسنضع بدلا منها التصحيح.
- لاحقة سنضعها في متغير اسمه sfx (اختصارا للكلمة Suffex)، لكن لن نقبل وجودها إلا إذا كان للمتغير canHasSuffex القيمة True، أو إذا كنا نتعامل مع المجموعة الأخيرة من الكلمات الخاصة (لأنها تقبل الصيغ المختلفة لياء النسب).

فلننظر إذن إلى الكود الذي سنكتبه في الموضع:

```
عثرنا على الكلمة الخاصة //
```

في البداية سنضع طول البادئة في متغير.. طول البادئة أصغر من موضع الكلمة الخاصة بـ ١.. مع ملاحظة أن أطول بادئة مسموح بها تتكون من ٣ حروف، لهذا لو وجدنا بادئة أطول فسنستخدم الأمر Loop لتجاوز اللفة الحالية من حلقة التكرار، والقفز مباشرة لتجربة الكلمة الخاصة التالية:

```
pfxLen = pos - 1
If pfxLen > 3 Loop End
```

فإذا لم يتحقق الشرط السابق، فسنضع هذه البادئة في متغير:

```
pfx = Word.Left(pfxLen)
```

وعلينا أن نتأكد أن البادئة التي حصلنا عليها (إن كان طولها أكبر من صفر) هي إحدى البوادئ المسموح بها، والتي وضعناها في المصفوفة prefixes في مصفوفات فرعية على حسب أطوالها.. فالمصفوفة الفرعية الأولى تحتوي على البوادئ التي طولها ١، والثانية تحتوي على البوادئ التي طولها ٢، والثالثة تحتوي على البوادئ التي طولها ٣.. هذا سيختصر عمليات البحث، فكل ما سنفعله، فهو أن نبحث في الخانة

التي رقمها يساوي طول البادئة [prefixes[pfxLen، فإن لم نجد البادئة، فسنستخدم الأمر Loop لتجاوز اللفة الحالية من حلقة التكرار، والقفز مباشرة لتجربة الكلمة الخاصة التالية:

```
if pfxLen > 0 And find(prefixes[pfxLen], pfx.Text) = 0 Loop End
```

والآن سنحسب طول اللاحقة، وهو يساوي:

طول الكلمة التي نفحصها – طول الكلمة الخاصة – طول البادئة

فإن كان طول اللاحقة صفرا، فلن نفعل شيئا آخر، وعلينا تكوين الكلمة الصحيحة من البادئة مضافا إليها الشكل الصحيح للكلمة الذي حفظناه في المتغير newWord، وسنجعلها القيمة العائدة من الدالة FixSpWords.. إما إذا كان طول اللاحقة أكبر من صفر لكن بشرط أن يكون أصغر من ٥ (لأن أطول لاحقة ضمن لواحق ياء النسب تتكون من ٤ حروف)، فسنحتاج لبعض الكود لمعالجة هذه الحالة:

```
sfxLen = wordLength - spWordLength – pfxLen
if sfxLen = 0
  w = pfx + newWord
  return AddLastTashkeel(w, lastTashk)
Elseif sfxLen < 5
  التعامل مع اللاحقة //
end
```

قبل أن نكمل، دعنا نفهم ما هي الدالة AddLastTashkeel:

كما قلنا سابقا، سنحافظ على التشكيل الذي يضعه المستخدم على آخر الكلمة فقد يضيف سكونا أو شدة أو تنوينا.. وقد احتفظنا بهذا التشكيل في المتغير lastTashk، لهذا يجب أن نستدعي الدالة AddlastTashkeel ونرسل إليها الكلمة الصحيحة والتشكيل الأخير الذي نريد وضعه عليها.. هذه الدالة ستفعل ما يلي:

- إن كان التشكيل الأخير الذي أرسلناه إليها نصا فارغا فستعيد الكلمة بدون تغيير.. ونفس الحال لو كانت الكلمة تنتهي بحرف الألف، لأننا لا نضع عليه أي تشكيل.
- إن كانت الكلمة تنتهي بتشكيل فستزيل آخر حرف منها وتضيف التشكيل المطلوب.
- غير هذا، ستضيف التشكيل المرسل إليها إلى نهاية الكلمة.

هذا هو كود هذه الدالة:

```
Func AddlastTashkeel(word, lastTashk)
  if lastTashk = "" or word.Right(1) = "ا"
    return word
  elseif IsTashkeel(word.RightStr(1))
    return word.RemoveLast(1) + lastTashk
  else
    return word + lastTashk
  end
EndFunc
```

والآن، فلنعد إلى الكود الأصلي.. سنكتب الآن كود المقطع:

```
// التعامل مع اللاحقة
```

سنضع اللاحقة في متغير:

```
sfx = Word.RightStr(sfxLen)
```

ثم سنتعامل معها في حالتين: إما أنها لاحقة عادية، وإما أنها لاحقة نسب وهي تصلح فقط مع المجموعة العاشرة (المجموعة الأخيرة):

```
if canHasSuffex = True and sfxLen < 4
  // الحالة الأولى
end
if gpIdx = spGroupsCount
  // الحالة الثانية
end
```

لاحظ أن بعض الكلمات في المجموعة العاشرة (مثل "إله") تقبل اللواحق العادية (مثل "إلهكم") وتقبل لواحق النسب (مثل "إلهيّة")، وهذا معناه أننا يجب أن نجرب الحالتين تباعا بشرطين منفصلين ولن نستخدم ElseIf لأن رينج كما تعلم لو نفذت شرط If فلن تنفذ شرط ElseIf.. ولا تقلق من فحص الحالة الثانية بلا داع، فإن نجحت الحالة الأولى، فسنستخدم الأمر Return لإعادة النتيجة ومغادرة الدالة في الحال، لهذا مثلا لو كنا نفحص الكلمة "إلهي" فسنعثر عليها من الحالة الأولى (ياء الملكية) ولن نفحص الحالة الثانية (ياء النسب).. وهذا لن يسبب أي التباس في البرنامج، فلو كتب المستخدم "إلهيّ" بشدة على ياء النسب، فسنحتفظ بالشدة وستضيفها الدالة AddlastTashkeel وسنحصل على النتيجة المتوقعة، رغم أننا عالجنا الكلمة على

أنها ياء الملكية ☺.. فالمهم أن يحصل المستخدم على النتيجة الصحيحة.

والآن فلننظر إلى كود الحالة الأولى:

يجب أولا أن نتأكد أن اللاحقة هي إحدى اللواحق التي وضعناها (بدون تشكيل) في المصفوفة AbsSuffixes.. ونحن هنا أيضا نقسم اللواحق إلى مصفوفات فرعية على حسب أطوالها، لهذا سنبحث في المصفوفة الفرعية الموجودة في الخانة التي رقمها هو طول اللاحقة، فإن وجدناها فسنجمع البادئة مع التصحيح مع اللاحقة (سنأخذها بالتشكيل هذه المرة من المصفوفة Suffixes) لتكوين ناتج الدالة:

```
j = find(AbsSuffixes[sfxLen], sfx)
If j > 0
  w = pfx + newWord + Suffixes[sfxLen][j]
  return AddlastTashkeel(w, lastTashk)
End
```

وأخيرا هذا هو كود الحالة الثانية، وهو مشابه لكود الحالة الأولى، لكننا سنتعامل هنا مع المصفوفتين AbsRelativeSuffixes و RelativeSuffixes:

```
j = find(AbsRelativeSuffixes[sfxLen], sfx)
If j > 0
   w = pfx + newWord + RelativeSuffixes[sfxLen][j]
   return AddlastTashkeel(w, lastTashk)
End
```

آخر مهمة أمامنا الآن هي كتابة كود المقطع:

```
// لم نعثر على الكلمة الخاصة لكنها تبدأ بحرف الألف ولها وضع خاص
```

في البداية، سنكتب دالة لفحص حالتي المد واللام وليكن اسمها MaddOrLam، وسنرسل إليها الكلمة الأصلية (غير المشكلة)، والكلمة الخاصة.. هذه هي:

```
Func MaddOrLam(word, spWord)
  madd = spWord.SetCharAt(1, "آ")
  pos = word.IndexOf(madd, 1)
  if pos return [pos, 1] end

  if spWord = "الله"
    pos = word.IndexOf("له", 1)
    if pos return [pos - 1, 2] end
  end
```

```
  pos = word.IndexOf(spWord.RemoveFirst(1), 1)   // لام
  return [pos - 1, 1]
EndFunc
```

لاحظ أن الترتيب هنا مهم، بسبب احتمال تداخل الحالات لكثرة وتنوع البوادئ.. لهذا يجب أن نفعل هذا بالترتيب:

- فحص حالة ألف المد.. لفعل هذا سنستخدم الدالة aString.SetCharAt لتغيير أول حرف في الكلمة الخاصة إلى "آ" (تذكر أننا تأكدنا من قبل أن أول حرف من الكلمة هو "ا")، ثم نبحث عن هذه الصيغة الجديدة في الكلمة التي نفحصها، فإن وجدناها فسنعيد النتيجة ونخرج من الدالة مباشرة.
- فحص حالة دخول اللام على لفظ الجلالة.. لفعل هذا سنبحث عن "له" في الكلمة، فإن وجدناها فسنعيد النتيجة ونخرج من الدالة مباشرة.
- وأخيرا فحص حالة اللام، حيث سنستخدم الدالة aString.RemoveFirst لحذف أول حرف من الكلمة ("ا")، والبحث عن هذه الصيغة الجديدة، ونعيد النتيجة أيا كان ناتج البحث.

وسنجعل هذه الدالة تعيد مصفوفة تتكون من خانتين:

- الخانة الأولى تحتوي على الموضع الذي عثرنا فيه على الكلمة الخاصة، وسيكون هو نتيجة البحث pos في حالة ألف المد، لكن سنطرح منه ١ في حالتي اللام، لأننا نبحث عن الكلمة الخاصة بدون اللام، ونحتاج في الكود التالي إلى أخذها في الاعتبار، لأن هذا سيخبرنا بطول البادئة.. فمثلا، سنحصل على الموضع ٢ إذا دخلت البادئة "ول" على لفظ الجلالة ("ولله").
- والخانة الثانية سنضع فيها عدد الحروف التي حذفناها من الكلمة الخاصة بسبب هذه الحالة الخاصة.. في حالة لفظ الجلالة حذفنا الحرفين "ال"، لكن في حالتَيْ ألف المد واللام حذفنا الحرف "ا" فقط.

الآن يمكننا أن نستخدم هذه الدالة في كتابة كود هذه الحالات الخاصة:

في البداية سنستدعي الدالة ونضع النتيجة التي تعيدها في مصفوفة اسمها result، ونضع أول خانة في هذه المصفوفة في متغير اسمه pos:

```
result = LamOrMadd(word, SpWord)
pos = Result[1]
```

والآن إذا كان pos أصغر من ١ أو كان أكبر من ٣ (أطول بادئة نتعامل معها تتكون من ٣ حروف)، فسنستخدم الأمر Loop لتجاوز اللفة الحالية من حلقة التكرار والانتقال لفحص الكلمة الخاصة التالية مباشرة:

```
if pos < 1 or pos > 3 Loop End
```

سنتأكد أيضا أن آخر حرف في البادئة (الموجود عند الموضع pos)، هو ألف مد أو لام، وإلا فسنتجاوز اللفة الحالية:

```
first = word[pos]
if (first != "ل" and first != "آ") Loop end
```

فإن وصلنا إلى هنا فسنضع البادئة في متغير اسمه pfx:

```
pfx = Word.Left(pos)
```

وعلينا أن نتأكد أن هذه البادئة موجودة فعلا في مصفوفة البوادئ، لكن لاحظ أن علينا تغيير همزة المد "آ" في البادئة إلى همزة عادية قبل البحث.. لفعل هذا سنستخدم الدالة aString.RemoveLastStr لحذف حرف من نهائية البادئة ونضيف إليها "أ".. وإذا لم نجد البادئة، فسنتجاوز اللفة الحالية:

```
if first = "آ"
  w2 = pfx.RemoveLastStr(1) + "أ"
  i = find(prefixes[pos], w2)
else
  i = find(prefixes[pos], pfx.Text)
end

if i =  0 Loop End
```

فإذا وصلنا هنا، فهذه فعلا هي الحالة الخاصة التي نتعامل معها.

لاحظ أن الكلمات الخاصة التي تبدأ بالألف لا تلحق بها لواحق عادية، لكن يمكن أن تلحق بها ياء النسب إن كنا نتعامل مع المجموعة العاشرة (الأخيرة).. لهذا سنحسب طول اللاحقة، مع الأخذ في الاعتبار طرح الحروف التي حذفناها من الكلمة الخاصة (الموجودة في الخانة الثانية من المصفوفة Result) من طول الكلمة الخاصة:

```
sfxLen = wordLength - (spwordLength - Result[2]) - pos
```

فإن كان طول اللاحقة صفرا، فسنجعل الدالة FixSpWords تعيد نتيجة تتكون من البادئة مضافا إليها تصحيح الكلمة الخاصة newWord لكن بعد حذف أول حرف منها ("ا").. ولن نحذف أي شيء آخر من تصحيح لفظ الجلالة، فكتابته العروضية الصحيحة "اللاه" وعند إضافة اللام نحذف الألف فقط مثل الكلمات العادية "للاه".

```
if sfxLen = 0
  w = pfx + newWord.RemoveFirst(1)
  return AddlastTashkeel(w, lastTashk)
elseif gpIdx = spGroupsCount and sfxLen < 5
  sfx = Word.RightStr(sfxLen)
  j = find(AbsRelativeSuffixes[sfxLen], sfx)
  w = pfx + newWord.RemoveFirst(1) + RelativeSuffixes[sfxLen][j]
  if j > 0 return AddlastTashkeel(w, lastTashk) end
end
```

وبالنسبة للمقطع Elself فنحتاجه إن كانت هناك لاحقة، ويجب أن نتأكد أننا نتعامل مع المجموعة العاشرة (لواحق النسب)، وأن طول اللاحقة أصغر من ٥ (لأن أطول لاحقة من لواحق النسب تتكون من ٤ حروف).. كود هذا المقطع واضح ولا جديد فيه بالنسبة لما شرحناه من قبل عندما فحصنا حالة ياء النسب مع الكلمات العادية، لكننا هنا فقط نحذف الحرف الأول من تصحيح الكلمة الخاصة ("ا").

وهكذا نكون قد أتممنا معالجة الحالات الخاصة بحمد الله، ويمكنك رؤية الكود كاملا في الملف SpWords.ring، كما أن الملف SpWords.tests.ring في المجلد tests يحتوي على ١٥٨ اختبارا لهذه الدالة لصيغ مختلفة من الحالات الخاصة والبوادئ واللواحق.

-٥-

الكتابة العروضية

في الفصل السابق، صححنا كتابة الكلمات الخاصة إملائيا، وسنكمل في هذا الفصل تحويل الكتابة الإملائية إلى كتابة عروضية.. دعنا إذن نكتب الدالة التي تفعل هذا.

الدالة DoAroodWrite:

تستقبل هذه الدالة معاملين، أولهما يحمل الكلمة التي نريد معالجتها (وهي نص من النوع aString)، والثاني يمثل رقم هذه الكلمة في الجملة.

Func DoAroodWrite(word, wordNum)

وستعيد هذه الدالة الكتابة العروضية للكلمة (وهي نص من النوع aString).

وكل ما سنفعله في هذه الدالة هو تطبيق مجموعة من القواعد لتحويل الكتابة الإملائية إلى كتابة عروضية.. وهي:

١- وضع فتحة على واو العطف:

إذا كانت الكلمة تتكون من حرف الواو فقط (واو العطف)، فسنضع عليها فتحة لأنها واو متحركة (ليست واو مد أو لين):

```
If word = "و"
  word.Append(ex.Fat7ah)
  return word
end
```

٢-٣- إشباع الهاء المضمومة والمكسورة:

عند اتصال ضمير الهاء بنهاية الكلمة (مثل "قلبه") يجب أن نقرر إذا كان سيتم إشباعه أم لا.. إشباع الحرف في النطق يعني مد حركته، بتحويل الضمة إلى واو (قلبُهُ -> قلبهو)، والكسرة إلى ياء (قلبِهِ -> قلبهي).. والقاعدة السائدة أن ضمير الهاء يشبع إذا لم يسبقه حرف ساكن ولم تبدأ الكلمة التالية له بألـ.. ولكن المشكلة التي ستواجهنا هي أننا لا نستطيع أن نعرف إن كانت الهاء ضميرا أم أصلية في الكلمة (مثل وجه)، كما أن هناك حالات يتم فيها إشباع الهاء رغم وجود حرف ساكن قبلها، كما في قراءة "فيه" في قوله تعالى: (ويخلد فيه مهانا).. لهذا وجدت أن أفضل حل لهذه المشكلة، هي أن نترك المستخدم قرار إشباع الهاء بوضع ضمة أو كسرة عليها.. إذن فكل المطلوب، هو أن نحول "هُ" في نهاية الكلمة إلى "هو" و "هِ" في نهاية الكلمة إلى "هي":

```
CanHaveTanween = True
If word[word.Length -1] = ex.Haa and
      ( Haa_Dmh_EdgEx.Replace(word, Haa_WawRep) or
        Haa_Ksrh_EdgEx.Replace(word, Haa_YaaRep) )
   CanHaveTanween = False
End
```

لاحظ ما يلي:

- المتغير CanHaveTanween هو مؤشر يخبرنا إن كان من الممكن أن تنتهي الكلمة بتنوين أم لا (سأخبرك بفائدته لاحقا).. في البداية سنجعل قيمته True، لكن إذا واجهنا حالة تمنع وجود تنوين في النهاية فسنجعل قيمته False.. انتهاء الكلمة بهاء مضمومة أو مكسورة هو بالتأكيد أحد هذه الحالات، فلا يوجد أي تنوين هنا.
- تأكدنا أولا أن الحرف قبل الأخير في الكلمة word[word.Length -1] هو حرف الهاء، حتى لا نضيع الوقت في فحص حالتين غير ممكنتين.
- تذكر أن المعامل المنطقي and لا يفحص الشرط الثاني إذا كان الشرط الأول

خاطئا، وأن المعامل or لا يفحص الشرط الثاني إذا كان الشرط الأول صحيحا.

- أذكرك مرة أخرى أن وضع الأقواس حول جمل الشرط يغير معناها.. هنا تم وضع الشرطين المربوطين بالمعامل Or بين قوسين، وهذا معناه أن نتيجتهما النهائية هي الشرط الثاني للمعامل And.

كل هذا جميل وبسيط.. لكنه لا يجيب عن أهم سؤال هنا: ما معنى:

Haa_Dmh_EdgEx.Replace(word, Haa_WawRep)

مبدئيا، سنتفق على أن Ex هي اختصار للكلمة الإنجليزية "تعبير" Expression، وأن Rep هي اختصار الكلمة الإنجليزية "استبدال" Replacement.. سأستخدم هاتين اللاحقتين في كل التعبيرات القادمة.

إذن فالجملة السابقة تقول إننا سنطبق الدالة Replace على الهاء التي تليها ضمة تليها حافة الكلمة edge (وهو ما عبرت عنه بالاسم Haa_Dmh_Edg)، فإن وجدناها في الكلمة word فسنضع بدلا منها هاء تليها واو Haa_waw.

إذن فالشرطة المنخفضة _ هنا تفصل كل حرفين في التسمية الإنجليزية للمتغير، والمقطع Edg يعبر عن حافة الكلمة، فإن جاء في البداية كان هذا معناه أن الكلمة يجب أن تبدأ بهذه الصيغة مثل Edg_AL للتعبير عن أن الكلمة تبدأ بـ "ال"، وإن جاءت في النهاية فيجب أن تنتهي الكلمة بهذه الصيغة كما في مثالنا هذا.. يمكنك الآن تطبيق نفس القواعد على:

Haa_Ksrh_EdgEx.Replace(word, Haa_YaaRep)

والتي تستبدل الهاء التي تليها كسرة في نهاية الكلمة، بهاء تليها ياء.

جميل.. لكننا ما زلنا لم نفهم ما هي تلك الصيغ التي تنتهي بـ EX و Rep.

في الواقع المتغيرات التي تنتهي بـ Rep هي نصوص عادية تماما، وهذا هو تعريفها بمنتهى البساطة:

Haa_WawRep = "هو"
Haa_YaaRep = "هي"

لكني فضلت استخدام متغيرات بأسماء إنجليزية لأن إرسال النصوص العربية كمعاملات للدوال يسبب فوضى في محرر الكود في لغة رينج، ويجعل من الصعب

عليك قراءة الجملة بالترتيب الصحيح.. كما أن وضع القيم في متغيرات (ربما يعاد استخدامها أكثر من مرة) يجعل من السهل علينا تعديلها إن استلزم الأمر، بإجراء تغيير في موضع واحد فقط في البرنامج.

لكن الأمر ليس بنفس البساطة عندما نتحدث عن المتغيرات التي تنتهي بـ Ex مثل Haa_Dmh_EdgEx، فهذه المتغيرات كائنات Objects تم تعريفها من فئة اسمها Expression (هذا هو إذن مصدر الاختصار Ex)، وهي فئة كتبتها لمعالجة التعبيرات النمطية Regular Expressions، وستجدها في مجلد البرنامج.. ولن أشرح كود هذه الفئة في هذا الكتاب فهو بسيط، ويمكنك أن تفهمه بنفسك بعد أن ترى كيف نستخدم هذه الفئة هنا.

التعبيرات النمطية Regular Expressions:
التعبير النمطي هو صيغة نصية تستخدم للبحث في النصوص، وهي أشبه بلغة برمجة صغيرة تتيح لك وضع مجموعة من الشروط للبحث في النص.. وتحتوي معظم لغات البرمجة (بما فيها رينج) على فئات جاهزة للتعامل مع التعبيرات النمطية، وهناك صيغة قياسية لهذه التعبيرات تلتزم بها معظم لغات البرمجة، لكني فضلت هنا أن أستخدم صيغة خاصة بي وأبني فئة مختصرة لمعالجة الكتابة الإملائية، فهذا أتاح لي إضافة ميزات جديدة، كما أنها فرصة لنزيد من مهارتنا البرمجية.

وسأؤجل قليلا شرح كيف نستخدم الفئة Expression لتعريف صيغ البحث مثل Haa_Dmh_EdgEx وغيرها.. وسأشرح هنا فقط قواعد تحويل الكتابة الإملائية إلى كتابة عروضية، فهذا سيجعلنا نركز أكثر في المنطق البرمجي دون أن نشغل أنفسنا بتفاصيل الكود، ثم سأعود لشرح التعبيرات النمطية في نهاية الفصل.

٤-٥- معالجة التاء المربوطة "ة":

سنحول التاء المربوطة إلى هاء إذا تلتها سكون، وإلا فسنحولها إلى تاء مفتوحة:

```
if Taa5_SokonEx.Replace(word, Haa_SokonRep)
  CanHaveTanween = False
else
  Taa5Ex.Replace(word, ex.Taa)
end
```

لاحظ أنني أعبر عن التاء المربوطة بـ Taa5 باعتبار أن الرقم ٥ يشبه الهاء.. ولم أهتم هنا بالتأكد من وجود التاء المربوطة في نهاية الكلمة، لأن هذا مكانها الطبيعي، وكتابتها في منتصف الكلمة ليس أمرا متوقعا من المستخدم (لن يكتب مثلا "مكتب" بدلا من "مكتب"، لكن حتى إن فعل فسنصححها له ☺).

ومن الواضح طبعا أن انتهاء الكلمة بتاء مربوطة عليها سكون يعني أن الكلمة غير منونة، لهذا وضعنا False في CanHaveTanween في تلك الحالة.

معالجة حالات الألف:

فيما يلي سنبدأ في معالجة القواعد الإملائية التي تتضمن حرف الألف "ا"، ولأنها حالات كثيرة وفحصها يحتاج بعض الوقت، فسيكون من الأذكى أن نتجاهل فحصها إلا إذا كانت الكلمة تحتوي على هذا الحرف:

```
if word.IndexOf(ex.Aleph, 1)
  // حالات حرف الألف
end
```

فانتبه أن الكود التالي سيوضع بدلا من التعليق في المقطع السابق.

٦- حذف ألف واو الجماعة:

قاعدة بسيطة، مع ملاحظة أن واو الجماعة لا تنون:

```
if WawGama3ahEx.Replace(word, WawGama3ahRep)
  CanHaveTanween = False
end
```

٧- إضافة فتحة بعد الواو التي تليها ألف في نهاية الكلمة:

بما أننا حذفنا ألف واو الجماعة في الخطوة السابقة، فالألف المتبقية هنا هي ألف المثنى، مثل "نَمَوَا" وهي صيغة المثنى من الفعل "نما ينمو"، وفي هذه الحالة الواو ليست ساكنة، وسنضع عليها فتحة:

```
if W_AEx.Replace(word, W_Fat7a_ARep)
  CanHaveTanween = False
end
```

وسأوضح عند كتابة التعبير النمطي لواو الجماعة وواو المثنى كيف نفرق بينهما.

٨- حذف الألف التي عليها سكون:

في حالات نادرة، لا تنطق الألف في نهاية الكلمة، مثل ألف واو الجماعة ولكنها كانت سهلة الرصد وعالجناها بالفعل، ومثل "أنا" التي يمكن أن تنطق ألفها أو تحذف على حسب إرادة الشاعر ومناسبة الوزن، لهذا سنسمح للمستخدم بتمييزها بوضع سكون عليها.. فلو كتب مثلا "أناْ" فسنضع فتحة بدلا من الألف والسكون لتصير "أنَ"، لكن لو كتب "أنا" فلن يحدث أي تغيير:

```
A_SokonEx.Replace(word, ex.Fat7ah)
```

٩- معالجة حرف شمسي مشدد بعد أل:

أنت تعرف أن كلمة "الشمس" تنطق "أشْ شَمس"، وينطبق هذا على باقي الحروف الشمسية.. في حالتنا هذه، سنعالج الصيغة التي تبدأ بـ "ال" التعريف، سواء كانت على اللام سكون أم لا، لكن المعيار الهام هنا هو أن يليها حرف شمسي عليه شدة.. هذا المعيار سيفرق بين الكلمتين "التَهم" من الالتهام و "التّهم" جمع "التهمة".. تذكر أننا نتيح تسهيلات للمستخدم لكي لا يضع علامات التشكيل، لكن كما ترى، هناك بعض مواضع التباس في اللغة، لهذا عليه أن يمنحنا الحد الأدنى من علامات التشكيل لمنع هذا الالتباس.

لاحظ أيضا أن الألف في "ال" هي ألف وصل، أي أنها تنطق فقط في بداية الكلام وتحذف في وسط الكلام، لهذا سنحولها إلى ألف مهموزة "أ" إن كانت هذه أول كلمة

في الجملة، وسنتركها كما هي لو كانت الكلمة في أي موضع آخر، حيث سنحذف الألف فيما بعد لكن تركها مهم لفعل أشياء أخرى كما سنرى لاحقا.

```
_A = Select(wordNum = 1, AHmz_FkkShdhRep, A_FkkShdhRep)
if Edg_Al_shmsy_ShdhEx.Replace(word, _A)
  CanHaveTanween = False
ElseIf …..
 // ….
End
```

لاحظ أن الاسم FkkShdh يعني "فك الشدة".. وكما ذكرنا سابقا، فك الحرف المشددة يكون بتحويله إلى حرفين متشابهين أولهما عليه سكون.

لاحظ أن حالات "ال" التالية لا يمكن أن تحدث معا، لهذا سنستخدم عدة جمل ElseIf لفحص كل حالة إن فشلت الحالة السابقة لها.. تذكر أن الكلمة المعرفة بأل لا يمكن وضع تنوين على آخرها (التنوين يكون فقط للاسم النكرة)، لهذا عند نجاح كل حالة سنضع False في المتغير CanHaveTanween.

دالة الاختيار Select:

كتبت هذه الدالة لاختصار مقطع الكود الذي نختار فيه إحدى قيمتين لوضعها في متغير على حسب نتيجة فحص شرط معين.. هذا المقطع يتكون من ٦ سطور، لكن باستخدام هذه الدالة سنكتبه في سطر واحد فقط!

وهي دالة بسيطة للغاية يستقبل معاملها الأول الشرط، فلو كان الشرط صحيحا تعيد القيمة المرسلة للمعامل الثاني، أما لو كان الشرط خاطئا فتعيد القيمة المرسلة للمعامل الثالث، هذا هو كودها:

```
Func Select(cond, ifTrue, ifFalse)
  If cond
    return ifTrue
  else
    return ifFalse
  end
EndFunc
```

١٠- معالجة حرف شمسي غير مشدد بعد "ال":

هذه الحالة تشبه الحالة السابقة، حيث تبدأ الكلمة بـ "ال" يليها حرف شمسي (لهذا سنستخدم نفس النص البديل الذي وضعناه في المتغير A_ سابقا)، لكن الاختلاف هنا هو عدم وجود شدة على الحرف الشمسي كنوع من التسهيل، وكما قلنا هذا سيجعل هناك التباسا مع كلمات مثل ("التمس"، "التبس"، "التهم")، لهذا سيكون ضماننا هنا هو استبعاد التاء من الحروف الشمسية، أي أن المستخدم لو كتب "التهمة" فسنعتبر اللام هنا أصلية وليست "ال" التي يليها حرف شمسي، وسيكون مجبرا على إضافة الشدة (وهذه هي فائدة الحالة السابعة).. غير هذا، يستطيع المستخدم أن يكتب "الديك" مثلا، ليحصل على "أدْديك".

```
ElseIf Edg_AL_ShmsyNotTaaEx.Replace(word, _A)
   CanHaveTanween = False
```

١١- معالجة حرف شمسي بعد "ال" ليست في البداية:

هذه الحالة مشابهة للحالة السابقة، لكن ال ليست في بداية الكلمة.. نحن نضمن هذا الآن لأننا عالجنا بالفعل "ال" التي في بداية الكلمة وغيرناها إلى صيغة أخرى.. لعلك تسأل لماذا احتجنا للقاعدة السابقة، ما دامت القاعدة الحالية هي الحالة العامة؟.. لو نظرت للحالة السابقة، فستعرف الإجابة، فالفارق هو نص الاستبدال، فهناك كان ألف الوصل في أول الكلمة، فإما أن يتحول إلى ألف مهموزة وإما أن نتركه كما هو، لكن هنا توجد حروف قبل ألف الوصل، لهذا سنحذف الألف تماما، فمثلا "والشّمس" ستصير "وشْشمس".

```
ElseIf Al_shmsy_ShdhEx.Replace(word, FkkShdhRep)
        CanHaveTanween = False
```

١٢- معالجة حرف شمسي بعد "ال" عليها سكون:

في هذه الحالة "ال" ليست في بداية الكلمة لأن الحالات السابقة عالجتها بالفعل، لكن ستتبقى حالة "ال" ليست في البداية، ويتبعها حرف شمسي لكن ليست عليه شدة (لأن الحالات السابقة عالجته).. والعلامة المميزة هنا هي وجود سكون على

"ال" مثل "والْديك" من طائر الدِّيك، فبدون وضع السكون أو الشدة سنعتبر الألف واللام حرفين أصليين مثل "والديك" أي والِدَيْك بمعنى أبويك ☺.. كما قلنا، هناك التباسات كثيرة، وعلى المستخدم أن يساعدنا في حلها.. لهذا لو كتب "والديك" فلن نغير فيها شيئا، لكن إن كتب "والْديك" أو "والدّيك" فسنحولهما إلى "ودْديك".. ونفس الشيء لو كتبهما المستخدم معا "والْدّيك"، لأن كلا القاعدتين ينتظر علامة ويضع احتمال وجود الأخرى (كما سنرى عند كتابة التعبيرات النمطية)، لذا فالقاعدة الأولى تغطي حالة وجود السكون والشدة معا.

```
elseIf AL_Sokon_ShmsyEx.Replace(word, FkkShmsyRep)
   CanHaveTanween = False
```

١٣- إضافة سكون بعد "ال" في بداية الكلمة:

بوصولنا هنا نكون عالجنا كل حالات اللام الشمسية، ولم تتبق أمامنا إلا اللام القمرية مثل "القمر".. في هذه الحالة قد تكون هناك سكون على اللام أو لا (سنضيفها في الكتابة العروضية في الحالتين).. وكما فعلنا سابقة، لو كانت هذه أول كلمة في الجملة فسنحول الألف إلى ألف مهموزة "أَلْقمر"، وإلا فسنتركها كما هي "الْقمر" لتحذف لاحقا.. وبسبب هذا الشرط الإضافي لن نستطيع أن نستخدم هنا ElseIf، ولكن سنستخدم Else وداخلها جملة شرط فرعية:

```
else
  _AlRep = Select(wordNum = 1, AHmz_L_SokonRep, Al_SokonRep)
  if Edg_Al_MaybeSokonEx.Replace(word, _AlRep)
    CanHaveTanween = False
  else
    معالجة حالات أخرى //
  end
end
```

وبوصولنا إلى هنا نكون قد انتهينا من معالجة كل حالات ال.. لاحظ أننا لن نفعل شيئا لو كتب المستخدم "والقمر" فقد يكون "ال" حرفين أصليين مثل "والهة" بكسر اللام من الوله أي العشق ومثل "والدي"، لهذا يجب أن يضع المستخدم السكون على ال حصريا.. إذا أردت تسهيل ذلك قليلا فيمكنك اعتبار أن "وال" في بداية

الكلمة هي واو العطف تليها ال التعريف، بشرط عدم وجود تشكيل على اللام، وعلى المستخدم أن يضع كسرة مثلا تحت اللام لمنعنا من إضافة السكون عليها.. سأترك لك فعل هذا بنفسك كتدريب (بعد أن تنتهي من قراءة هذا الفصل كاملا)، لكن عليك تشغيل اختبارات الكتابة الإملائية بعد كل تغيير تجريه، لأن الحالات متداخلة ومن السهل أن تتسبب في إفساد قواعد أخرى أثناء تعديلك للكود).

١٤- إضافة سكون بعد أي حرف ألف في بداية الكلمة:

سنضع سكون على أي حرف بعد الألف (لو لم تكن موجودة) بشرطين:

- أن تكون الألف في بداية الكلمة مثل "ادخل"، ""اكتب"، "امتحان".. هذا الشرط سيمنع البرنامج من إضافة سكون على كلمات مثل "ناس"، "فاس" (من الفأس.. نذكر أننا نتعامل مع شعراء قد يسهّلون الهمزات لملاءمة الأوزان والقوافي).
- ألا يكون هناك تشكيل بالفعل على الحرف التالي للألف مثل الشدة كما في ("ادّخر"، اتّخذ).

لاحظ أننا في هذه القاعدة أيضا سنحول الألف إلى ألف بهمزة مكسورة "إ" لو كانت الكلمة هي أول كلمة في الجملة مثل "إكْتب".. سأكتب كود هذه القاعدة مع كود القاعدة التالية.

١٥- تحويل الألف إلى ألف مكسور في بداية الكلمة:

بوصولنا هنا، سيتبقى فقط الألف التي يليها حرف عليه تشكيل (مثل الشدة)، وفي هذه الحالة سنحول الألف إلى ألف مكسورة الهمزة لو كانت الكلمة هي الأولى في الجملة.. هذا هو كود الحالتين ١٤ و ١٥:

```
Else
  _AlephRep = Select(wordNum = 1, E_HrfSakenRep, A_HrfSakenRep)
  if not Edg_A_HrfEx.Replace(word, _AlephRep)
      _A = Select(wordNum = 1, ex.AlephMaksor, ex.Aleph)
     Edg_AEx.Replace(word, _A)
  end
end
```

وبهذا نكون قد أنهينا جملة الشرط الأم التي تتأكد من وجود "ا" في الكلمة وعالجنا داخلها كل حالات "ال" وحالة الألف في بداية الكلمة.

معالجة حالات لل:

لم ننته من حالات ال حتى الآن، فما زال أمامنا احتمال دخول حرف الجر "ل" على "ال" فيحولها إلى "لـل" كما في "للشمس"، لهذا نحتاج لمعالجة هذه الحالات.. وسأستخدم هنا المعامل or بدلا من ElseIf لأننا لا نحتاج إلى شروط داخلية مثلما فعلنا في الكود السابق لمعالجة حالة الألف في أول كلمة في الجملة، فهنا لا توجد ألف أصلا ☺.. هذا هو الكود:

```
if word.IndexOf(ex.lam, 1) and
    (LL_Sokon_ShmsyEx.Replace(word, L_FkkShmsyRep) or
     LL_shmsy_ShdhEx.Replace(word, L_FkkShdhRep) or
     Edg_Atf_LL_shmsy.Replace(word, Atf_L_FkkShdhRep) or
     Edg_Atf_LLEx.Replace(word, Atf_LL_SokonRep) )

  CanHaveTanween = False
end
```

الشرط الأول يتأكد أن الكلمة تحتوي على حرف اللام، وإلا فلا داعي لفحص باقي الشروط، لهذا استخدمنا and ووضعنا كل شروط معالجة "لـل" بين قوسين كشرط مجمع يمثل الشرط الثاني للمعامل And.. وطبعا لو كان الشرط صحيحا (بمعالجة أي حالة من حالات "لل") فهذا معناه أن هذه كلمة معرفة بأل ولا يمكن أن تنتهي بتنوين، لهذا وضعنا False في المتغير CanHaveTanween.

لاحظ أن اللام التي تدخل على "ال" يمكن أن تكون حرف جر (لام مكسورة) أو لام القسم أو التوكيد (لام مفتوحة)، لهذا سنضع في اعتبارنا أثناء تكوين تعبير البحث احتمال أن يضيف المستخدم كسرة أو فتحة بعد اللام الأولى.

فلننظر الآن للحالات الأربع لمعالجة "لل" الموجودة في الشرط السابق.

١٦- معالجة حرف شمسي بعد "للْ":

العلامة المميزة هنا هي وجود سكون على اللام الثانية، لهذا لن نهتم إن كانت في بداية الكلمة أم لا.. فإن وجدنا بعدها حرفا شمسيا (سواء عليه شدة أو لا)، فسنحذف اللام الثانية ونفك الشدة ("للْشمس" تتحول إلى "لشْشمس"):

LL_Sokon_ShmsyEx.Replace(word, L_FkkShmsyRep)

١٧- معالجة حرف شمسي مشدد بعد "لل":

العلامة المميزة هنا هي وجود شدة على الحرف الشمسي بعد "لل".. سنحذف اللام الثانية ونفك الشدة ("للشّمس" تتحول إلى "لشْشمس"):

LL_shmsy_ShdhEx.Replace(word, L_FkkShdhRep)

١٨- معالجة حرف شمسي بعد "لل":

هذا أكبر تسهيل سنمنحه للمستخدم عند التعامل مع لل، فيمكنه ألا يضع السكون على اللام ولا الشدة على الحرف الشمسي، لكن بشرط أن تبدأ الكلمة ب "لل" أو يسبقها أحد حروف العطف (عبرت عنها في اسم المتغير ب Atf).. الحروف المسموح بها هي الواو والفاء إضافة إلى همزة الاستفهام "أ"، مع إمكانية وجود أكثر من حرف منها معا مثل "فو"، "أو"، "أف".

Edg_Atf_LL_shmsy.Replace(word, Atf_L_FkkShdhRep)

١٩- إضافة سكون بعد "لل":

الآن تتبقى الحروف القمرية بعد لل، فإذا لم تكن هناك سكون بعد لل الموجودة في أول الكلمة، أو التي تسبقها حروف العطف، فسنضيف هذه السكون:

Edg_Atf_LLEx.Replace(word, Atf_LL_SokonRep)

وهكذا نكون أنهينا حالات "لل" بحمد الله.

٢٠- إضافة سكون على الواو أو الياء إذا سبقتهما فتحة:

يمكن أن تكون الواو أو الياء ممدودة مثل ذهبوا، أو عليها سكون مثل مشَوْا.. ورغم أن كليهما يعتبر حرفا ساكنا، لكن هناك اختلافا يحدث عندما تأتي بعدها كلمة تبدأ بحرف الألف، فمثلا:

"واتبعوا الرسول" تنطق "وتْتَبعُرْ رسول" فتم حذف الواو مع الألف، وكذلك بالنسبة لياء المد مثل "قلبي الكبير" فتنطق "قلبِلْ كبير".

لكن "وآتَوا الزكاة" تنطق "وآتَوُزْ زكاة"، فتم تحريك سكون الواو إلى ضمة ولم تحذف الواو مع الألف، كذلك بالنسبة للياء اللينة مثل "ابنَي الخال" فتكسر السكون ولا نحذف الياء "ابنَبِلْ خال".. هذا الفارق الجوهري يلزمنا بوضع سكون على الواو اللينة (التي يسبقها حرف مفتوح):

Fat7ah_WOrYEx.Replace(word, Fat7ah_WOrY_SokonRep

٢١- ٢٢- معالجة الأسماء التي تبدأ بألف ومعرفة بأل:

سنعالج هنا كلمات مثل "الاستثناء"، "للاستثناء".. تذكر أننا عالجنا حالات "ال" و "ل" من قبل، لهذا ربما نكون وضعنا سكونا على اللام في خطوة سابقة.. في هذه الحالة سنحذف ألف "ال" التعريف (إن وجدت)، وسنحذف الألف الأصلية التي تبدأ بها الكلمة ونكسر اللام السابقة لها ونضع سكونا على الحرف الذي يليها إن لم يكن مشكّلا.. أي أن كلمة "الاستثناء" ستتحول إلى "لِسْتثناء".. والسبب الذي جعلنا نتخلص من ألف أل هنا، أنها لم تعد تؤثر في حال انتهاء الكلمة السابقة بساكن.. فمثلا "في الاستثناء" تنطق "في لِسْتثناء"، ولا نحذف الياء كما نفعل في حالة "في الفصل" التي تنطق "فِلْ فصل".

لاحظ أن المستخدم قد يضع السكون على الحرف التالي للألف أو لا يضعها، وهذا هو سبب استخدامنا لتعبيرين نمطيين وربطنا بينهما بالمعامل Or حتى لا يتم

فحص التعبير النمطي الثاني إذا نجح الأول، فلا يمكن أن توجد الصيغتان معا في نفس الكلمة:

```
If maybeA_L_Sokon_A_SakenEx.Replace(word,
        L_Kasrah_HrfSakenRep) Or
    maybeA_L_Sokon_A_HrfEx.Replace(word,
        L_Kasrah_HrfSakenRep)

  CanHaveTanween = False
End
```

٢٣- فك الشدة:

الآن سنفك الحرف المشدد إلى حرفين أولهما ساكن.. لقد فعلنا هذا من قبل، لكننا كنا نتعامل مع الحروف الشمسية التي عليها شدة، أما هنا، فسنعالج الحروف المشددة بشكل عام، فمثلا سنحول "أصرّ" إلى "أصرْر".

```
Hrf_ShdhEx.Replace(word, Hrf_Sokon_HrfRep)
```

لاحظ أنني استخدمت هنا نص الاستبدال Hrf_Sokon_HrfRep مع أنني سابقا كنت أستخدم المتغير FkkShdhRep.. في الحقيقة التعبيران يؤديان نفس الوظيفة، لكن هناك فارقا صغيرا بينهما، يرتبط برقم الحرف المشدد في مصفوفة المجاهيل التي نعوض عنها في التعبير الرقمي.. سأشرح هذا بالتفصيل لاحقا، ونحن نتعلم كيف نكوّن تعبيرات البحث وتعبيرات الاستبدال.. كما قلنا، نحن هنا نركز فقط على وظيفة البرنامج، ولا نشغل بالنا بتفاصيل الكود.

٢٤- حذف أي ألف وسط الكلمة بعدها حرف ساكن:

هذه القاعدة ستعالج بعض الحالات النادرة في اللغة العربية التي يتتابع فيها حرفان ساكنان في الكلمة، مثل "مادّة"، فبعد فك الشدة سنحصل على "مادْدة".. في وزن الشعر، سنعتبر الحرفين الساكنين حرفا ساكنا واحدا، فالألف هنا مجرد مد زائد وسنعتبره مجرد فتحة، لهذا سنحذف الألف لنحصل على "مدْدة".

واضح طبعا أن ترتيب القواعد مهم جدا، فلا بد أن نفك التشديد أولا في الخطوة السابقة قبل أن نطبق هذه القاعدة.. هذا هو السبب في تأخيرنا لتطبيق هذه القاعدة

وعدم وضعها مع حالات الألف.

انتبه أيضا أن هذه الحالة يجب أن تتجنب الألف في بداية الكلمة مثل "الْقمر" فهذه "ال" التعريف، وتتجنب أيضا الألف التي يتبعها حرف ساكن في نهاية الكلمة (وهذا وارد في الشعر في القوافي الساكنة مثل "نـاسْ").. لهذا سنضيف في بداية ونهاية التعبير النمطي شرط تجنب حافة الكلمة NotEdg:

```
NotEdg_A_Saken_NotEdgEx.Replace(word, HrfSakenRep)
```

٢٥-٣١- فك التنوين:

آخر شيء سنفعله هنا هو فك التنوين إلى نون ساكنة، مع وضعة فتحة أو ضمة أو كسرة على الحرف السابق للتنوين على حسب نوع التنوين.. وهنا يحين دور المتغير CanHaveTanween، فإن كانت قيمته True فسنبدأ في تطبيق حالات التنوين، وهي ٧ حالات مختلفة، كما هو واضح في الكود:

```
if CanHaveTanween
  if not ( Tnwn_AlephEx.Replace(word, Ft7h_Noon_SokonRep) or
    Aleph_ TnwnEx.Replace(word, Ft7h_Noon_SokonRep) or
    Tnwn_LeenEx.Replace(word, Ft7h_Noon_SokonRep) or
    Leen_TnwnEx.Replace(word, Ft7h_Noon_SokonRep) or
    TnwnFt7Ex.Replace(word, Ft7h_Noon_SokonRep) or
    TnwnDmEx.Replace(word, Dmh_Noon_SokonRep) )
      TnwnKsrEx.Replace(word, Ksrh_Noon_SokonRep)
  end
end
```

السبب في زيادة حالات التنوين، هو وجود خمس حالات لتنوين الفتح:

- فالكلمات العادية يضاف إليها ألف التنوين عند تنوينها بالفتح، ويمكن أن يضع المستخدم التنوين على ألف التنوين (مثل "شاعراً")، أو على الحرف السابق لها (مثل "شـاعرًا" وهذا هو الصحيح بالمناسبة).. فهاتان قاعدتان، وفي كلتيهما سنحذف ألف التنوين، وبالتالي سيكون نص الاستبدال "نَ".
- الكلمات التي تنتهي بألف لينة (مثل "هدى") يمكن وضع التنوين على الألف

اللينة (مثل "هدىً") أو على الحرف السابق لها (مثل "هدًى").. وهاتان قاعدتان أخريان، وفي كلتيهما سنحذف الألف اللينة، وسيكون نص الاستبدال "نَنْ".

- هناك كلمات لا يضاف إليها ألف تنوين (مثل "سماءً"، "مكتبةً"، "مبدأً").. وهذه هي القاعدة الخامسة.. وهنا أيضا سيكون نص الاستبدال "نَنْ".

ويتبقى بعد هذا تنوين الضم وتنوين الكسر، ولا يوجد شيء غير عادي فيهما.

أظن هذا يوضح الكود السابق.. انتبه فقط أننا استخدمنا المعامل Or لنفحص كل قاعدة إذا فشلت القاعدة السابقة لها، فلا يمكن بالطبع أن تحتوي الكلمة على تنوينين مختلفين.. استخدام Or يستلزم جملة شرط، وفي حالتنا هذه كان الشرط هو التأكد من فشل أول ٦ قواعد، وفي هذه الحالة سيكون الكود الذي ننفذه في جملة الشرط هو القاعدة السابعة والأخيرة.

المعاملات المرجعة Reference Variables:

لعلك نظرت إلى الدالة Replace التي استخدمناها لمعالجة التعبيرات النمطية مثل:

Hrf_ShdhEx.Replace(word, Hrf_Sokon_HrfRep)

وتساءلت في دهشة: أين هي النتيجة العائدة من هذه الدالة، وكيف ستغير النص الموجود في المتغير word؟

هنا يجب أن أخبرك عن الآلية التي ترسل بها رينج القيم إلى معاملات الدوال:

- فهي تتعامل مع النصوص والأرقام كقيم عادية، فتنشئ نسخة جديدة منها وترسلها إلى المعاملات، وهو ما يسمى إرسال المعاملات بالقيمة By Value.
- لكنها ترسل القوائم Lists والكائنات Objects مرجعيا.. وهذا معناه أنها لا تنسخ القائمة أو الكائن، وإنما تنسخ عنوانه في الذاكرة وترسله إلى معامل الدالة، وهو ما يسمى إرسال المعاملات بالمرجع By Reference.

في الحقيقة، لا تحمل المتغيرات في لغات البرمجة أي قيم، ولكنها تحمل العناوين التي توجد فيها هذه القيم في الذاكرة.. وفي حالة الكائنات (ومنها المصفوفات) سيكون

من الأسرع تمرير مرجع الكائن (عنوانه في الذاكرة) بدلا من إنشاء نسخة مستقلة جديدة منه تحتل مساحة كبيرة في الذاكرة.

وللتأكد من هذا، جرب هذا الكود في رينج (ستجده في الملف ByRef.Test) في المجلد tests ضمن أمثلة الكتاب:

```
Load "aString.ring"
a = 1
Test(a)
? "a = " + a

b = "abcd"
Test(b)
? "b = " + b

c = [4, 3, 2]
Test(c)
? "-------List:------"
? c
? "-------"

d = new aString("abcd")
Test(d)
? "d = " + d.Text

Func Test(x)
   x = x + 1
EndFunc
```

في هذا الكود كتبنا دالة بسيطة اسمها Test تستقبل معاملا واحدا، وكل ما تفعله هو أن تجمع ١ على قيمته.. وقد استدعينا هذه الدالة أربع مرات لنرسل إليها رقما ونصا ومصفوفة وكائنا من النوع aString.. أنت تعلم أن المعامل + يعمل مع كل هذه الأنواع بطرق مختلفة، فهو يجمع رقمين ويشبّك نصين ويضيف خانة جديدة للمصفوفة.. لو جربت هذا الكود فسترى هذه النتائج على الشاشة:

```
a = 1
b = abcd
-------List:------
4
3
2
1
-------
d = abcd1
```

تلاحظ في هذه النتائج:

- أن المتغير a الذي يحمل رقما والمتغير b الذي يحمل نصا ظل كل منهما محتفظا بنفس قيمته بلا تغيير بعد استدعاء الدالة، وهذا يعني أن رينج مررت هذين المتغيرين إلى الدالة بالقيمة By Value.
- أما المتغير c الذي يحمل قائمة، والمتغير d الذي يحمل كائنا من النوع aString فقد تغيرت قيمتاهما بعد استدعاء الدالة، فقد أضيف عنصر جديد للقائمة، وأضيف حرف إلى نهاية النص الذي يحمله المتغير d، وهذا يعني أن رينج مررت هذين المتغيرين إلى الدالة بالمرجع By Reference.. هذا إذن هو ما يحدث في الدالة Replace، فنحن نجري التعديلات مباشرة على الكائن الذي يحمله المتغير word المرسل كمعامل للدالة.

ومن المهم أن تنتبه جيدا إلى أن رينج تنسخ المتغيرات جميعا (بما فيها المصفوفات والكائنات) عند استخدام المعامل = .. فلو كتبت مثلا:

obj1 = obj2

فهذا معناه أن كلا من المتغيرين obj1 و obj2 يشير إلى كائن مختلف في الذاكرة.. جرب مثلا:

```
list1 = [1, 2, 3]
List2 = list1
list1[1] = 5
? list2[1]
```

الكود السابق سيعرض على الشاشة ١، لأن وضع ٥ في أول خانة في القائمة list1 لا يؤثر على المصفوفة List2 فهي كائن مستقل في الذاكرة.

ملحوظة أخيرة:

نظرا لأن رينج لغة مرنة، فمن الممكن أن تعيد الدالة قيمة أو تعمل كإجراء لا يعيد أي قيمة.. ولا يوجد ما يمنع أن تقوم دالة واحدة بالأمرين معا.. فلو نظرت مثلا للسطر الأخير في الدالة DoAroodWrite فسترى هذا الكود:

If TestingAroodWrite Return word End

هذا هو السطر الوحيد في هذه الدالة الذي يحتوي على الأمر Return، وكما ترى فهو موضوع في جملة شرط، وهذا يعني أنه قد يُنفَّذ أو لا، تبعا لقيمة المتغير TestingAroodWrite، وهو متغير معرف في منطقة المتغيرات العامة في بداية الملف قبل منطقة تعريف الدوال، وقيمته الافتراضية هي False، أي أن الدالة DoAroodWrite في الوضع الافتراضي لن تعيد أي قيمة، وسيتم التأثير مباشرة على المعامل word كما شرحنا.. لكن لغرض الاختبار، سيكون من المختصر أن نكتب جملا كالتالية في الملف AroodWrite.Tests.ring:

AssertEqual(3, DoAroodWrite(New aString("قلبهُ"), 1).Text, "قلبهُو")

في هذه الحالة نحتاج أن نجعل الدالة DoAroodWrite تعيد الكلمة التي عالجتها، لهذا ستجد أننا كتبنا في بداية هذا الملف:

TestingAroodWrite = true

فبدون أن تعيد الدالة نتيجة، كنا سنضطر لكتابة الكود السابق كما يلي:

```
testWord = New aString("قلبهُ")
DoAroodWrite(testWord, 1)
AssertEqual(3, testWord.Text, "قلبهُو")
```

ربما يكون الكود الأخير أوضح، لكنه سيضاعف عدد سطور الكود، وملف الاختبار يحتوي بالفعل على أكثر من ١٢٠ اختبارا لتغطية الاحتمالات المختلفة للحالات التي عالجناها في الكتابة العروضية، لهذا فضلت استخدام الصيغة الأولى.

معالجة حالات التقاء ساكنين:

الخطوة المنطقية الآن، هي أن نفهم كيف بنينا التعبيرات النمطية التي استخدمناها في الدالة DoAroodWrite، لكني أفضل أن نكمل أولا الخطوة الأخيرة في الكتابة العروضية، وهي معالجة حالات انتهاء الكلمة بحرف ساكن مثل الألف والواو والياء وأي حرف عليه سكون، وبدء الكلمة التالية لها بحرف ساكن، وهو تحديدا حرف الألف، فلا يمكن أن تبدأ الكلمة العربية بحرف ساكن غيره، وهذا هو السبب في عدم حذفنا للألف من بداية الكلمة (مثل اشْتمس)، لاحتياجنا لها في هذه المرحلة.

هنا سنكتب دالة اسمها AroodWrite، تستقبل مصفوفة من الكلمات:

```
Func AroodWrite(words)
   L = len(words)
   معالجة الكتابة العروضية //
EndFunc
```

كما ذكرنا، يتم تمرير المصفوفات إلى الدوال بالمرجع، لذا فأي تغيير سنجريه على خانات المصفوفة words سيؤثر مباشرة على المصفوفة الأصلية التي نرسلها إلى الدالة AroodWrite ولهذا لا نحتاج أن تعيد هذه الدالة أي نتيجة.

وسيكون على الدالة AroodWrite تنفيذ الدالة DoAroodWrite على كل كلمة في المصفوفة words ثم معالجة حالات التقاء الساكنين في كل كلمتين متتاليتين.. فكيف سنفعل هذا؟

أول تفكير سيراودك، هو كتابة حلقتي تكرار Loops 2:

١- حلقة التكرار الأولى تستدعي الدالة DoAroodWrite:

```
For i = 1 to L
   DoAroodWrite(words[i], i)
Next
```

تذكر أن كل خانة في المصفوفة تحمل كائنا من النوع aString وهذا يعني أنه سيمر إلى الدالة DoAroodWrite بالمرجع، حيث سيجري تعديل الكلمة إلى الكتابة العروضية الصحيحة، وسيؤثر هذا التغيير على الكائن مباشرة، وبما أن

مرجعه موجود في خانة في المصفوفة، فهذا يعني أن التغيير سيحدث مباشرة على الخانة words[i].

٢- وحلقة التكرار الثانية تعالج حالات التقاء الساكنين:

```
For i = 1 to L
  معالجة التقاء ساكنين //
Next
```

هذه طريقة صحيحة لأداء المهمة، لكن هناك طريقة أكفأ منها، تتيح لنا استخدام حلقة تكرار واحدة فقط.. فحتى نعالج التقاء الساكنين، نحتاج لمعرفة الكتابة العروضية لكلمتين فقط في كل مرة:

- معالجة حالة انتهاء الكلمة الأولى بساكن وبدء الكلمة الثانية بساكن.
- معالجة حالة انتهاء الكلمة الثانية بساكن وبدء الكلمة الثالثة بساكن.
- معالجة حالة انتهاء الكلمة الثالثة بساكن وبدء الكلمة الرابعة بساكن.

وهكذا...

وبصيغة عامة نحتاج لمعالجة حالة انتهاء الكلمة السابقة words[i-1] بساكن، وبدء الكلمة الحالية words[i] بساكن.. ونظرا لأن طرح ١ من العداد i في أول لفة سيعطينا صفرا، ولا توجد خانة في المصفوفة رقمها صفر، إذن فعلينا أن نبدأ حلقة التكرار من ٢، ونحصل على الكتابة العروضية لأول كلمة بمفردها خارج حلقة التكرار.. هكذا سيكون شكل حلقة التكرار العامة التي تعالج الكتابة العروضية لكل كلمة وتعالج حالات التقاء ساكنين لكل زوجين من الكلمات:

```
DoAroodWrite(words[1], 1)
For i = 2 to L
  DoAroodWrite(words[i], 0)
  if words[i][1] = ex.Aleph
    معالجة التقاء ساكنين //
  end
Next
```

تذكر أن المعامل الثاني للدالة DoAroodWrite يستقبل رقم الكلمة في الجملة.. الحالة المؤثرة هي أن تكون الكلمة هي رقم ١ في الجملة، ولهذا أرسلنا ١ مع أول كلمة

في المصفوفة words، لكن غير هذا (داخل حلقة التكرار) لن يفيدنا رقم الكلمة في شيء، لهذا يمكن أن نرسل أي رقم غير ١، سواء كان قيمة العداد i أو حتى صفرا.

الآن نحن جاهزون لمعالجة حالات التقاء ساكنين، وكما قلنا، هي تحدث فقط إذا كانت الكلمة الحالية words[i] تبدأ بألف.. وقد استخدمنا التعبير words[i][1] في جملة الشرط لأن أول حرف في الكلمة الحالية هو أول خانة في المصفوفة التي تمثل النص.

المطلوب الآن داخل جملة الشرط، أن نفحص نهاية الكلمة السابقة، ونرى كيف سنعالج حالات التقاء ساكنين.. لكن علينا أولا أن نجعل الكود أوضح بوضع بعض القيم في متغيرات لها أسماء واضحة، وهذا أيضا سيوفر بعض الوقت بمنع تكرار أداء بعض العمليات البرمجية:

```
lastWordId = i - 1
lastWord = words[lastWordId]
lastWordLen = lastWord.Length
endOfLastWord = lastWord[lastWordLen]
Harakah = ""
```

المتغيرات الأربعة الأولى واضحة، أما المتغير Harakah فسنضع فيه حركة (علامة تشكيل: فتحة أو كسرة أو ضمة)، كما سنرى أثناء كتابة الكود.

والآن، سنفحص آخر حرف في الكلمة السابقة، حيث يعتبر ساكنا في الحالات الموضحة في جملة الاختيار التالية:

```
Switch endOfLastWord
  Case ex.Waw
    // الكلمة السابقة تنتهي بواو
  Case ex.Yaa
    // الكلمة السابقة تنتهي بياء
  Case ex.Aleph
    // الكلمة السابقة تنتهي بألف
  Case ex.AlephLeen
    // الكلمة السابقة تنتهي بألف لينة
  Case ex.Sokon
    // الكلمة السابقة تنتهي بسكون
End
```

لكن مهلا.. هذه هي أول مرة نرى فيها مقطع الكود switch.. فما هو؟

جملة الاختيار من البدائل Switch:

المقطع Switch هو حالة خاصة من الجملة If ElseIf، وهو يجعل كتابة الكود أسهل وأسرع حينما تتعامل مع متغير واحد (مثل endOfLastWord)، وتريد فحص الاحتمالات المختلفة للقيمة التي يحملها، حيث توضع كل قيمة في حالة Case يليها الكود الذي تريد تنفيذه عند تحقق هذه الحالة.. ويمكنك أن تستخدم Else لتنفيذ كود معين في حالة عدم تحقق كل الحالات السابقة:

```
n = 10
Switch n
  Case 1
     ? "n = 1"
  Case 2
     ? "n = 1"
   Else
     ? "n > 2"
End
```

وتتيح لك رينج صيغة أخرى، تستخدم فيها On (بدلا من Case) و Other (بدلا من Else) و Off (بدلا من End)، وهذا باعتبار أن الكلمة Switch تعني مفتاح التشغيل، والكلمة On تعني تشغيل المفتاح عند قيمة معينة، والكلمة Off تعني إغلاق المفتاح (نهاية المقطع):

```
n = 10
Switch n
  On 1
     ? "n = 1"
  On 2
     ? "n = 1"
  Other
     ? "n > 2"
Off
```

ولا يوجد ما يمنع من استخدام مزيج من كل الكلمات السابقة، كأن تستخدم On مع قيمة و Case مع قيمة أخرى، ثم Else للقيم الأخرى ثم تنهي المقطع بـ Off.

وكما في كل المقاطع التي تعرفنا عليها سابقا، يمكنك تغليف المقطع Switch

بالقوسين المتعرجين بدلا من استخدام End أو Off:

```
n = 10
Switch n {
  On 1
    ? "n = 1"
  Case 2
    ? "n = 1"
  Other
    ? "n > 2"
}
```

ولك مطلق الحرية في صياغة الكود بالشكل الذي ترتاح له أكثر، لكني سألتزم بالصيغة الأولى Case.. Else.. End في كتابة الكود في هذا الكتاب.

والآن دعنا نكتب الكود الخاص بكل حالة من حالات التقاء الساكنين:

الكلمة السابقة تنتهي بواو (Case ex.Waw):

تذكر أننا وضعنا سكونا على الواو اللينة، وأننا هنا نتعامل مع كلمة تنتهي بواو فقط (ليس عليها سكون).. إذا فهذه واو مد، ويجب حذفها عند التقاء ساكنين، فمثلا (قتلوا الحلم) ستتحول إلى (قتلُ لْحلم)، إلا في حالة واحدة، إذا كانت الواو مشددة مثل (النّموّ السكاني) التي ستتحول إلى (أنْنموْو سْسكاني).. انتبه إلى أننا قد فكننا التشديد بالفعل، وبالتالي سنعرف أن الواو مشددة إن كان عدد حروف الكلمة أكبر من ٣ وكانت تنتهي بواو مشددة مفكوكة "وْو":

```
If not (lastWordLen > 3 and lastWord.EndsWith("وْو"))
  Harakah = ex.Dammah
End
```

لاحظ أننا لم نفعل أي شيء هنا للتعامل مع واو المد، ما عدا وضع ضمة في المتغير Harakah.. هذا مؤشر Flag على ضرورة حذف آخر حرف في الكلمة ووضع ضمة بدلا منه.. السبب في هذا أن هناك تشابها بين حالات الواو والياء والألف، لهذا من الأفضل تنفيذها جميعا بنفس الكود كما سنرى بعد قليل.

الكلمة السابقة تنتهي بياء (Case ex.Yaa):

هذه الحالة مماثلة لحالة الواو، إلا أننا هنا نتعامل مع الياء، وبالطبع سنضع كسرة في المتغير Harakah:

```
If not (lastWordLen > 3 and lastWord.EndsWith("يْ"))
   Harakah = ex.Kasrah
End
```

الكلمة السابقة تنتهي بألف مد (Case ex.Aleph):

لا يمكن أن يكون ألف المد مشددا، لهذا في كل الأحوال يجب أن نحذف الألف (علامة هذا أن نضع فتحة في المتغير Harakah) إلا استثناء واحدا: عند نداء لفظ الجلالة بالأداة "يا": (يا الله)، فلا تحذف أل بل تنطق همزة قطع، وهي حالة فريدة من نوعها في اللغة العربية، فنقول (يا ألـلاه).. هذا هو سبب استخدامنا للأمر Loop حتى نتجاهل باقي اللفة الحالة في حلقة الدوران For ونقفز مباشرة إلى اللفة التالية:

```
If (lastWord.EndsWith("يا") or lastWord.EndsWith("يَا")) And
        words[i].StartsWith("الْلاه")
  words[i].SetCharAt(1, "أ")
  Loop
Else
  Harakah = ex.Fat7ah
End
```

الكلمة السابقة تنتهي بألف لينة (Case ex.AlephLeen):

لا توجد استثناءات للألف اللينة، لذا سنحولها إلى فتحة في كل الأحوال:

```
Harakah = ex.Fat7ah
```

وعلى المستخدم أن يفرق بين الياء "ي" والألف اللينة "ى".. لقد تجاوزنا عن الفارق في بعض الحالات، لكن هناك فروقا لا نستطيع التغاضي عنها مثل هذه الحالة، لهذا من الأفضل أن يعوّد المستخدم نفسه على استخدام الياء (التي تحتها نقطتان).. فلو كتب مثلا: (يجرى الماء) فسنحولها إلى (يجرَ لْماء) بوضع فتحة على الراء وليس كسرة، لكن في كلتا الحالتين هذا لن يغير وزن البيت.

الكلمة السابقة تنتهي بسكون (Case ex.Sokon):

في الحالة العامة، الكلمة التي تنتهي بسكون وتليها كلمة تبدأ بحرف ساكن مثل (قالتْ الأعراب)، يتم تحريك السكون إلى كسرة (قالتِ لْأعراب).. ويستثنى من هذا حالتان:

- حرف الجر مِن، حيث يتم تحريك سكونه بالفتح مثل (ومِنْ الناس) التي تتحول إلى (ومِنَ نْناس).
- الواو اللينة (التي عليها سكون)، حيث يتم تحريكها بالضم، مثل (الذين آتَوْا الزكاة) تتحول إلى (ألْلذين آتَوُ زْزكات).

إذن، فكل ما سنفعله هو تحويل السكون (آخر حرف في الكلمة السابقة) إلى فتحة أو ضمة أو كسرة تبعا لنوع الحالة التي نتعامل معها:

```
If lastWord = "مِنْ" or lastWord = "ومِنْ" or lastWord = "فمِنْ"
   Harakah = ex.Fat7ah
ElseIf lastWord[lastWordLen - 1] = ex.Waw
  Harakah = ex.Dammah
Else
  Harakah = ex.Kasrah
End
```

الآن، لم يتبق إلا أن نستخدم المتغير Harakah، فلو كان فارغا فلن نفعل شيئا (الكلمة السابقة لا تنتهي بحرف ساكن)، أما أن كان يحتوي على علامة تشكيل، فيجب أن نضعها بدلا من آخر حرف في الكلمة السابقة، لكن بشرط ألا يكون الحرف قبل الأخير عليه تشكيل بالفعل مثل "قالُو" فلو وضعنا ضمة بدلا من الواو، فستصير هناك ضمتان على اللام!

```
If Harakah != ""
   If IsTashkeel(lastWord[lastWordLen - 1])
      words[lastWordId].Delete(lastWordLen, 1)
   Else
      words[lastWordId].SetCharAt(lastWordLen, Harakah)
   End
End
```

أخيرا، سنحذف الألف من بداية الكلمة الحالية.. وهناك شيء آخر سنفعله لجعل الكتابة العروضية أوضح في القراءة، فوجود حرف ساكن في بداية الكلمة يبدو مربكا،

لهذا سننقله إلى نهاية الكلمة السابقة.. فمثلا (في الفصل) سنحولها إلى (فلْ فصل).. وسنراعي احتياطا الحالة التي قد يكتب فيها المستخدم ألفا مستقلة ككلمة، وهذا خطأ بلا معنى، لكنه قد يحدث ويسبب خطأ في الكود:

```
If words[i].Length = 1
   words[i].Delete(1, 1) // حذف الألف
Else  //نقل الحرف الساكن الذي يلي الألف إلى الكلمة السابقة
   words[lastWordId].Append(words[i].Mid(2, 2))
   words[i].Delete(1, 3) // حذف الألف والحرف الساكن الذي يليه
End
```

وهكذا نكون قد أنهينا الكتابة العروضية بحمد الله.

اختبار الوحدات Unit Testing:

ستجد ١٤٠ اختبارا تغطي الحالات المختلفة للكتابة العروضية في الملف AroodWrite.Tests.ring في المجلد Tests.. هذه الاختبارات مهمة للغاية وساعدتني في حل مشاكل كثيرة، فبعد كل حالة من حالات الكتابة العروضية كنت أكتب اختبارات تغطي كل احتمالاتها، وبعد أن أضيف حالات أخرى للكتابة العروضية أعيد تشغيل كل الاختبارات، لأتأكد من أن الحالات الجديدة لم تفسد شيئا في الحالات السابقة.. يسمى هذا باختبار الوحدات Unit Testing، وهو يعني اختبار كل وحدات (أجزاء) البرنامج للتأكد من أنها تعمل بشكل صحيح، وهي مفيدة في ضمان ألا يتسبب تعديل جزء من الكود أو إضافة كود جديد في حدوث أي أخطاء في وحدات البرنامج القديمة.. بالنسبة للكتابة العروضية، هذه عملية معقدة ومتشابكة وترتيبها حساس جدا، لهذا حدثت معي أخطاء كثيرة وأنا أضيف بعض القواعد الجديدة، ما استلزم مني إعادة ترتيب القواعد أو تعديلها، وبدون اختبار الوحدات كان من الصعب أن أجعل هذا الكود يعمل بشكل صحيح.

لهذا لو قررت أنت إجراء تعديلات في أي ملف في البرنامج، فقم بها خطوة خطوة، وبعد كل خطوة نفذ الاختبارات الخاصة بالملف الذي تعدله، فهذا سيكشف لك أي أثار

سلبية للتغير الذي أجريته في الخطوة الأخيرة، ومن ثم يمكنك التراجع عنها أو تصحيحها.

الاختبارات جزء حيوي في البرمجة الحديثة، يقي البرنامج من كثير من الأخطاء، ويوفر عليك الكثير من الوقت والجهد والمال عند تعديل البرنامج وتطويره مستقبلا.

التعبيرات اللفظية Verbal Expressions:

وصلنا الآن إلى مرحلة فهم التعبيرات النمطية التي استخدمناها في الدالة DoAroodWrite لتحويل الكتابة الإملائية إلى كتابة عروضية.

لو نظرت لبداية الملف AroodWrite.ring فسترى تعريف التعبيرات النمطية، مثل:

```
Haa_Dmh = ex.Haa + ex.Dammah
Haa_Dmh_EdgEx = new Expression(Haa_Dmh + ex.Edge, 1)
```

الكود السابق يستخدم الكائن ex واسمه اختصار للكلمة الإنجليزية expression.. هذا الكائن معرف في الملف aExpressions.ring، كنسخة من فئة باني التعبيرات:

```
ex = New ExpressionBuilder
```

وهي تمنحنا بعض الخصائص والوسائل التي تساعدنا في بناء التعبيرات النمطية باستخدام تعبيرات لفظية.

دعنا ننظر أولا لأحد التعبيرات النمطية، لنفهم لماذا نحتاج إلى استخدام التعبيرات اللفظية.. هذا مثلا تعبير البحث عن واو الجماعة:

```
@B4WawGama3ah(-1)ْ)~ا(ْ)~و)^
```

واضح طبعا أن هذه الصيغة صعبة الفهم، إضافة إلى أن وجود حروف عربية وسط النص الإنجليزي يجعل من المستحيل عليك قراءة النص بالترتيب الصحيح، ولكي تدرك هذا، سأقسم لك النص السابق إلى أجزاء بالترتيب الذي كتبته به، وقارنه بالترتيب الظاهر لك عند عرضه مجمعا:

```
@B4WawGama3ah( 1)
و
~(°)
ا
~(°)
^
```

رغم أن الأمور قد صارت أوضح بهذا التقسيم، لكن

التعبير اللفظي ما زال يحتوي على رموز مبهمة، ويبدو أقرب للشفرة.. هذا صحيح، فكما قلنا هو أشبه ما يكون بلغة برمجة تحتوي على رموز توجه بعض الأوامر لمحرك التعبيرات النمطية للبحث عن صيغ معينة في النص.

لاحظ أن هناك صيغة قياسية عالمية للتعبيرات النمطية تلتزم بها معظم لغات البرمجة، لكني لم ألتزم بها هنا، لسببين:

١- أن هذا محرك تعبيرات نمطية مخصص لوظيفة محددة، وقد استبعدت الكثير من الصيغ الشهيرة لأني لن أستخدمها، وأضفت بعض الصيغ الجديدة التي تؤدي وظائف تجعل الأمور أبسط بالنسبة للمهمة المطلوبة (الكتابة العروضية).

٢- أننا في كل الأحوال سنستخدم التعبيرات اللفظية، وبالتالي لا يهمنا شكل التعبير النمطي فلن نستخدمه في الكود بأنفسنا مباشرة.. الحقيقة أنني أتمنى أن تحتوي كل لغات البرمجة على تعبيرات لفظية لتسهيل التعامل مع التعبيرات النمطية.

إذن فقد عدنا مجددا إلى السؤال: ما هي التعبيرات اللفظية Verbal Expressions؟

ستعرف الإجابة بنفسك، لو رأيت كيف بنينا التعبير النمطي السابق في البرنامج:

```
maybeSokon = ex.Maybe(ex.Sokon)
WawGama3ah = ex.LookBack(1, "B4WawGama3ah") +
        ex.Waw + maybeSokon + ex.Aleph + maybeSokon + ex.Edge
WawGama3ahEx = New Expression(WawGama3ah, 1)
```

في الكود السابق، استخدمنا باني التعبيرات Expression Builder، لكتابة تعبيرات لفظية يسهل قراءتها وفهم وظيفتها بمجرد النظر، وبمجرد جمعها معا ينتج نص التعبير النمطي المطلوب.. وفي الكود السابق استخدمنا:

- **الوسيلة ex.Maybe**، وهي تستقبل نصا، وتخبر محرك التعبير النمطي أنه قد يجده في النص في هذا الموضع أو لا يجده.. في المثال السابق نحن نقول إننا قد نجد سكونا على الواو أو سكونا على الألف.. هذه هي حالة واو الجماعة، حيث يمكن أن تكون الواو ساكنة مثل (سعَوْا)، ويمكن أن توضع سكون على الألف للدلالة على أنها ألف زائدة لا تنطق.. سنسمح للمستخدم بعدم وضع أي من هاتين السكونين أو وضعهما لو شاء.. ويمكنك أن تستنتج أنني أستخدم

الرمز ~ في التعبير النمطي لأخبره أن النص قد يوجد أو لا، وأضع النص بعد هذا الرمز بين قوسين مثل (a)~.

- **الوسيلة ex.LookBack** بمعنى "انظر للوراء" وهي تستقبل معاملين، الأول هو عدد الحروف التي سننظر إليها قبل الموضع الحالي في النص، والثاني اسم دالة الفحص التي نريد من محرك التعبيرات النمطية إرسال هذه الحروف إليها، للتأكد أنها تعيد True، وإلا فلن يصلح الموضع الحالي للعثور على الصيغة المطلوبة، ويجب البحث في موضع آخر من النص.

وفي مثالنا هذا، نريد أن ننظر إلى حرف واحد قبل واو الجماعة، ونرسله إلى دالة اسمها B4WawGama3ah، وهي دالة سنكتبها نحن للتأكد من أن هذه ليست واو أصلية يتبعها ألف المثنى مثل "نَمَوَا" أو ألف التنوين مثل "رخْوًا".. وسنرى كود هذه الدالة بعد قليل.

- وهناك **وسيلة مشابهة اسمها LookForward**، مهمتها النظر للأمام لعدد من الحروف التالية للموضع الحالي، وترسلها إلى دالة فحص.

وأنا أستخدم الرمز @ مع دوال النظر للأمام أو الخلف، وأُتبعه باسم الدالة، ثم قوسين بينهما رقم واحد فقط (من صفر إلى ٩) لأني لا أحتاج لأكثر من هذا في البرنامج، ويكون هذا الرقم موجبا في حالة النظر للأمام، وسالبا في حالة النظر للخلف.. وفي مثالنا هذا، ينتج هذا النص عن استدعاء الوسيلة LookBack:

@B4WawGama3ah(-1)

وتحتوي التعبيرات النمطية القياسية على صيغ للنظر للأمام والخلف، لكنها لا تستدعي دالة، وإنما تطابق صيغة تعبير نمطي أخرى على النص الذي تفحصه.. بالنسبة لي فضلت استدعاء دالة وكتابة بعض كود رينج، لأقلل تعقيد كود محرك التعبيرات النمطية وأختصر كتابة التعبيرات اللفظية التي تولد التعبير النمطي، بالإضافة إلى منحنا إمكانيات أكبر بمزج كود رينج بالتعبيرات النمطية.. لاحظ أننا نستطيع الاستغناء عن التعبيرات النمطية تماما وكتابة كود الكتابة العروضية كاملا بلغة رينج مباشرة، لكن نظرا لتداخل العلاقات بين قواعد الكتابة

العروضية وأهمية إجرائها بترتيب معين، كان هذا سيتطلب كتابة كود طويل جدا يتكون من كثير من حلقات التكرار، وسيكون شرحه وفهمه صعبا وتعديل أي جزء فيه محفوفا بالمخاطر.. لذا فالوضع المثالي هو استخدام تعبيرات نمطية بسيطة، تستدعي عددا قليلا من الدوال المكتوبة بلغة رينج عند الحاجة إليها.

- **الخاصية ex.Edge**، وهي تضيف الرمز ^ للتعبير الرقمي، ولو أضفته في أول الصيغة فسيخبر محرك التعبيرات النمطية أن الحرف التالي يجب أن يكون أول حرف في النص، أو بتعبير آخر: يتأكد أن النص يبدأ بالصيغة الخاصة بنا.. أما إن أضفت الرمز ^ في نهاية الصيغة، فسيخبر محرك التعبيرات النمطية أن النص يجب أن ينتهي بالصيغة الخاصة بنا.. هذا هو السبب في تسمية هذا الرمز بالحافة Edge، فهو يمثل أحد حافتي النص (البداية أو النهاية).. انتبه جيدا أن الرمز ^ لا يطابق حرفا معينا، أي أنه في حد ذاته ليس أول حرف في النص ولا آخر حرف في النص، ولكنّه يمثل شرطا يجب التزامه أثناء البحث.
- أضفت بعض الخصائص لتمثيل بعض الحروف العربية مثل Waw و Aleph حتى لا نكتب الحروف العربية وسط النصوص الإنجليزية.. كان من الممكن أن نستخدم معظم هذه الحروف من الملف aLetters.ring، مثل Letters.Waw، لكن نظرا لأن التعبيرات اللفظية طويلة، فضلت اختصار الاسم إلى ex إضافة إلى أنه أكثر صلة بالتعبيرات النمطية.. وقد أضفت بعض الاختصارات الأخرى مثل ex.Al وهي تمثل "ال" التعريف لأننا سنستخدمها كثيرا في بناء التعبيرات.

وهناك الكثير من الخصائص والوسائل الأخرى معرفة في الفئة ExpressionBuilder، سنتعرف عليها ونحن نبني التعبيرات النمطية.

والآن، فلنبدأ بناء التعبيرات النمطية باستخدام التعبيرات اللفظية:

١- وضع فتحة على واو العطف:

هذه القاعدة نفذناها بكود رينج مباشرة بدون استخدام تعبير نمطي، لأن هذا أبسط في كتابته وأسرع في تنفيذه، لكن لا يوجد ما يمنعك من تعريف تعبير نمطي لهذه الحالة كما يلي:

WEx = new Expression(ex.Edge + e.Waw + ex.Edge, 1)

لاحظ أن المعامل الثاني لمنشئ الفئة Expression يستقبل قيمة منطقية (True/False)، وأنت تعرف أن True في الحقيقة = ١ و False = ٠.. لو أرسلت True (أو ١) فأنت بهذا تطلب من محرك التعبيرات النمطية إيقاف البحث عن الصيغة بعد أول مرة يجدها في النص.. هذا سيوفر بعض الوقت عندما نكون متأكدين أن هذه الصيغة لا يمكن أن تأتي أكثر من مرة في نفس الكلمة.

وهذا هو تعبير الاستبدال الخاص بهذا التعبير النمطي:

WRep = e.Waw + ex.Fat7ah

وبهذا يمكنك تنفيذ هذه القاعدة في بداية الدالة DoAroodWrite كالتالي:

WEx.Replace(word, WRep)

لكني لا أنصحك بهذا، فتكرار فحص هذه القاعدة في كل الكلمات سيضيع وقتا بلا قيمة، في حين أن هذه حالة نادرة جدا تتعلق بكلمة تتكون من حرف الواو فقط، ولا تحتاج إلا لجملة شرط واحدة مباشرة لمعالجتها كما رأينا سابقا.

٢- إشباع الهاء المضمومة:

هذا التعبير سهل جدا، ويمكنك فهمه مباشرة:

Haa_Dmh = ex.Haa + ex.Dammah
Haa_Dmh_EdgEx = new Expression(Haa_Dmh + ex.Edge, 1)

ولعلك لاحظت تعريفي للمتغير Haa_Dmh، لأني سأكرر استخدامه في الكود لاحقا، ووضعه في متغير سيكون أفضل من إعادة تشبيك نفس النص عدة مرات، ناهيك أن هذا سيجعل الكود أكثر اختصارا ووضوحا.. وسأستخدم هذه الطريقة في

تكوين أجزاء التعبيرات النمطية التي سأعيد استخدامها.. والآن تعال نستخدم المتغير Haa_Dmh في تكوين نص الاستبدال:

```
Haa_WawRep = Haa_Dmh + ex.Waw
```

٣- إشباع الهاء المكسورة:

مماثل للتعبير السابق، مع تغيير الضمة إلى كسرة:

```
Haa_Ksrh = ex.Haa + ex.Kasrah
Haa_Ksrh_EdgEx = new Expression(Haa_Ksrh + ex.Edge, 1)
Haa_YaaRep = Haa_Ksrh + ex.Yaa
```

٤- تحويل التاء المربوطة الساكنة إلى هاء:

تذكر أن ex.Taa5 يمثل التاء المربوطة "ة":

```
Taa5_SokonEx = new Expression(ex.Taa5 + ex.Sokon, 1)
Haa_SokonRep = ex.Haa + ex.Sokon
```

٥- تحويل التاء المربوطة إلى تاء مفتوحة:

بعد تنفيذ التعبير النمطي السابق، ستتبقى فقط التاء المربوطة التي ليس عليها سكون، لهذا سنحولها إلى "ت":

```
Taa5Ex = new Expression(ex.Taa5, 1)
```

انتبه إلى أننا لم نشترط وجود التاء المربوطة في نهاية الكلمة، فقد يليها فتحة أو ضمة أو كسرة أو تنوين.

ولا نحتاج لتكوين صيغة استبدال هنا، فهي ببساطة ex.Taa لهذا أرسلناها مباشرة إلى المعامل الثاني للدالة Replace في كود الدالة DoAroodWrite.

٦- حذف ألف واو الجماعة:

لقد شرحنا بالفعل هذا التعبير النمطي:

```
maybeSokon = ex.Maybe(ex.Sokon)
WawGama3ah = ex.LookBack(1, "B4WawGama3ah") +
    ex.Waw + maybeSokon + ex.Aleph + maybeSokon + ex.Edge
WawGama3ahEx = New Expression(WawGama3ah, 1)
```

ولـم يتبـق إلا تعريـف الدالـة B4WawGama3ah.. لاحـظ أن B4 هـي اختصـار الكلمـة الإنجليزيـة "قبـل" Before.. هـذه الدالـة إذن سـتتأكد أن الحـرف السـابق لـواو الجماعة يحقق بعض الشروط.. هذا هو كودها:

```
Func B4WawGama3ah(str)
  If str = "" return false End

  hrf = str[1]
  if hrf = ex.Kasrah or hrf = ex.Sokon
    return false
  else
    return true
  end
EndFunc
```

أول شرط في الدالة، هو التأكد إن كان النص المرسل إليها فارغا، وفي هذه الحالة ستعيد False، لأن واو الجماعة لا يمكن أن تأتي في بداية الكلمة.

ستسألني: ألم نطلب البحث عن واو الجماعة في نهاية الكلمة؟

بلى.. ولكن المستخدم قد يكتب مثلا "وا إسلاماه".. "وا" هنا ليست واو الجماعة!

ستسألني أيضا: وكيف تستقبل دالة النظر للخلف نصا فارغا، وقد طلبنا أن تنظر إلى حرف واحد سابق؟

في الحقيقة نحن طلبنا أن ننظر إلى عدد من الحروف السابقة بحد أقصى حرف واحد.. أي أن صفر هو عدد مقبول أيضا.. هذا يجعل دوال النظر أكثر مرونة، فلو كانت تنظر لثلاثة حروف، فيمكنها أيضا أن تغطي احتمال وجود حرفين فقط، أو حرف واحد فقط، أو عدم وجود أي حروف.. لهذا علينا أن نفحص النص المرسل إليها ونتأكد من طوله، ونأخذ القرار المناسب لكل احتمال.

في حالتنا هذه، عدم وجود حرف سابق يعني أن هذه ليست واو الجماعة.. نفس الأمر ينطبق على الحالتين التاليتين:

- لو كان الحرف السابق سكونا، فهذه واو أصلية تتبعها ألف التنوين مثل "حلْوا".
- لو كان الحرف السابق فتحة، فهذه واو أصلية تتبعها ألف المثنى مثل "نمَوَا".

وغير هذا، ستعيد الدالة True لأن هذه هي واو الجماعة فعلا.

والآن، فلنكتب تعبير استبدال واو الجماعة:

WawGama3ahRep = ex.Waw + ex.Var(2)

الجديد هنا هو ex.Var(2).. الدالة Var هي اختصار للكلمة "متغير" Variable وهي تعيد النص الموجود في مصفوفة المتغيرات في الخانة التي نرسل رقمها كمعامل، وفي مثالنا هذا أرسلنا الرقم ٢، لقراءة الخانة الثانية في المصفوفة.

فما هي إذن مصفوفة المتغيرات؟

لو تأملت التعبيرات النمطية التي أنشأناها حتى الآن، فستكتشف أن أجزاءها تنتمي إلى أحد نوعين متميزين:

- نصوص ثابتة يجب أن نجدها كما هي في النص.
- ونصوص مجهولة لا نعرف ما هي مسبقا، لكنها تحقق شروطا معينة.. ويحفظ محرك التعبيرات النمطية هذه النصوص المجهولة (والتي تصير معلومة له بمجرد العثور على الصيغة التي يبحث عنها) في مصفوفة المتغيرات بنفس ترتيب العثور عليها.

ولو نظرت للتعبير النمطي WawGama3ahEx، فستجد أنه يحتوي على المتغيرات التالية:

- ex.LookBack: عند النظر للأمام أو الخلف، يتم حفظ النص الذي ننظر له في مصفوفة المتغيرات، وفي حالتنا هذه سيكون رقم ١ في المصفوفة ويمكن التعامل معه من خلال ex.Var(1).
- ex.Maybe: عند مطابقة نص محتمل (قد يوجد أو لا)، سيتم حفظ النتيجة في مصفوفة المتغيرات، فإن كان النص موجودا فستحتوي الخانة على هذا النص، وإن لم يكن موجودا فستكون فارغة.. في مثالنا هذا هناك احتمال لوجود سكون على الواو (المتغير رقم ٢)، وسكون على الألف (المتغير رقم ٣).

وللعلم، يشير نص الاستبدال إلى المتغيرات ببساطة بوضع # قبل رقم المتغير.. أي أن ex.Var(2) تضيف الصيغة "#2" إلى نص الاستبدال، وهي كما ترى أكثر

اختصارا، لكن استخدام الدالة Var يجعل الأمور أوضح لنا عند قراءة الكود، دون إرهاق أذهاننا لتذكر معنى الرمز #.

والآن، لو أعدت النظر إلى تعبير الاستبدال، فستجد أننا نريد وضع قيمة المتغير رقم ٢ بعد الواو.. وهذا معناه لو كانت هناك سكون على الواو، فسنحافظ عليها.. أما الألف فستحذف وبالتالي ستحذف معها سكونها إن وجدت، لهذا لا يعنينا المتغير رقم ٣.. كما لا يعنينا المتغير رقم ١ هنا، لأنه ينظر للحروف السابقة للواو ولا نحتاجها في عملية الاستبدال.. انتبه جيدا إلى أن دالة النظر للأمام والخلف لا تعتبر جزءا من النص الذي نعثر عليه، لهذا لا تتأثر الحروف التي ننظر لها بعملية الاستبدال لأنها خارج اهتمامنا، إلا إذا وقع جزء من هذه الحروف ضمن النص المطلوب بالفعل، وهذا يمكن أن يحدث عند النظر للأمام، كما سنرى لاحقا.. لكن كقاعدة عامة: طول النص الذي نعثر عليه ونجري عليه عملية الاستبدال، تحدده كل أجزاء التعبير النمطي ما عدا دوال النظر للأمام والخلف.

٧- إضافة فتحة بعد الواو التي تليها ألف في نهاية الكلمة:

```
W_AEx = New Expression(ex.Waw + ex.Aleph + ex.Edge, 1)
W_Fat7a_ARep = ex.Waw + ex.Fat7ah + ex.Aleph
```

٨- حذف الألف التي عليها سكون:

```
A_SokonEx = new Expression(ex.Aleph + ex.Sokon + ex.Edge, 1)
```

وتعبير الاستبدال هنا هو ex.Aleph.

٩- معالجة حرف شمسي مشدد بعد أل:

سنعرّف بعض المتغيرات لتسهيل كتابة التعبيرات التالية:

```
Al_MaybeSokon = ex.AL + maybeSokon
shmsy = ex.Validate(1, "Shams")
shmsy_Shdh = shmsy + ex.Shaddah
Al_shmsy_Shdh = Al_MaybeSokon + shmsy_Shdh
```

الجديد هنا هو الدالة ex.Validate، وهي تستقبل عدد الحروف التي نريد فحصها، واسم الدالة التي سيتم استدعاؤها لفحص هذه الحروف.. لاحظ أن دالة الفحص Validate تختلف عن دالة النظر للأمام أو الخلف في شيئين:

- أنها تفحص الحروف بدءا من الموضع الحالي في التعبير النمطي.. ففي مثالنا هذا، نريد تكوين تعبير نمطي يبحث عن "ال" ربما تليها السكون والحرف التالي سيتم إرساله للدالة Shams للتأكد من أنه حرف شمسي، والحرف التالي شدة.
- أن الحروف التي نفحصها بالدالة Validate تدخل ضمن النص الذي نبحث عنه، على عكس الدالتين LookForward و LookBack، فهما تنظران للأمام أو الخلف لبعض الحروف دون اعتبارها جزءا من النص.

وللعلم، ستضيف الدالة ex.Validate في الكود السابق النص التالي للتعبير النمطي، لتطلب إرسال حرف واحد للدالة التي اسمها Shams:

```
$Shams(1)
```

فإن أعادت هذه الدالة True، فسيتم وضع هذا الحرف في مصفوفة المتغيرات، ومواصلة فحص باقي التعبير النمطي، وإن أعادت False فسيعني هذا فشل المطابقة في هذا الموضع، وينتقل محرك التعبيرات النمطية للبحث عن الصيغة في موضع آخر.

هذا هو تعريف التعبير النمطي، مع ملاحظة أنه سيبدأ المطابقة من بداية الكلمة:

```
Edg_Al_shmsy_ShdhEx = new Expression(ex.Edge +
    Al_shmsy_Shdh, 1)
```

فللنظر إذن إلى الدالة Shams وكودها في منتهى البساطة:

```
Func Shams(str)
  return find(ex.Shamsiah, str.Text)
EndFunc
```

فكل ما فعلناه هو البحث عن الحرف في مصفوفة الحروف الشمسية ex.shamsiah، وجعل الدالة Shams تعيد نتيجة هذا البحث.. أنت تعلم أن الدالة Find تبحث في المصفوفة عن النص المطلوب، فإن وجدته أعادت رقم الخانة (تذكر أن أول خانة رقمها ١)، وإن لم تجده أعادت صفرا، وأنت تعلم أن رينج

تعتبر الشرط خاطئا لو كانت قيمته صفرا (تناظر False)، بينما لو كانت للشرط أي قيمة غير الصفر فستعتبره رينج صحيحا.. لكن احذر في هذه الحالة أن تستخدم Shams(x) = True في كتابة الشرط، لأن True في رينج متغير قيمته ١، والدالة Shams يمكن أن تعيد أرقاما أكبر من ١!.. لقد راعيت هذا بالتأكيد وأنا أكتب كود محرك التعبيرات النمطية، لكن لو أردت أن تكتب الدالة السابقة بطريقة لا تدع أي مجال للخطأ، فعليك التأكد من أن ناتج الدالة Find أكبر من صفر، وهذا سيضمن أن الدالة Shams ستعيد قيمة منطقية (صفر أو ١):

```
return find(ex.shamsiah, str.Text) > 0
```

هذه عملية مقارنة إضافية ستستغرق جزءا من المليار من الثانية، لكن أحيانا قد يكون هذا وقتا طويلا بالنسبة لبعض البرامج التي تنفذ عمليات معقدة كالرسوم وتستدعي نفس الدالة ملايين المرات!.. لكن ليس هذا هو الحال هنا لحسن الحظ.

بقي أن نكتب تعبير الاستبدال الخاص بهذا التعبير النمطي.. في الحقيقة لن نستطيع أن نكتب هذا التعبير هنا مباشرة، لأنه يعتمد على موضع الكلمة في الجملة، فكما شرحنا من قبل، لو كانت الكلمة في بداية الجملة، فسنحول الألف إلى ألف مهموزة "أ"، وهذا هو سبب استخدامنا لجملة شرط (من خلال الدالة Select) في كود الدالة DoAroodWrite:

```
_A = Select(wordNum = 1, AHmz_FkkShdhRep, A_FkkShdhRep)
```

إذن، فما يعنينا هنا هو تعريف التعبيرين A_FkkShdhRep و AHmz_FkkShdhRep، ونظرا لوجود جزء مشترك بينهما، فسنضعه في متغير اسمه FkkShdhRep لنعيد استخدامه في أكثر من موضع:

```
FkkShdhRep = ex.Expand(2)
A_FkkShdhRep = ex.Aleph + FkkShdhRep
AHmz_FkkShdhRep = ex.AlephHmz + FkkShdhRep
```

الجديد هنا في نص الاستبدال هو استخدامنا للدالة ex.Expand(2).. الكلمة Expand تعني تمديد وأنا أستخدمها هنا للإشارة إلى فك الشدة.. سنحتاج إلى فك التشديد في العديد من التعبيرات النمطية القادمة، ودائما سيكون الحرف المشدد

مجهولا وسيوضع في مصفوفة المتغيرات عند العثور على نتيجة، ولكن موضع هذا المتغير سيختلف من تعبير نمطي لآخر، لهذا بدلا من إعادة كتابة التعبير التالي عدة مرات (حيث n هي رقم المتغير):

ex.Var(n) + ex.Sokon + ex.Var(n)

فسنستدعي الدالة ex.Expand ونرسل إليها رقم المتغير.

لو نظرت الآن لتعبير الاستبدال FkkShdhRep، فسيتضح لك أنه يفك تشديد الحرف الذي وضعناه في الخانة الثانية في مصفوفة المتغيرات.. الخانة الأولى في هذه المصفوفة ستحتوي على سكون أو نص فارغ تبعا لنتيجة التعبير maybeSokon، والخانة الثانية ستحتوي على الحرف الذي أكدت الدالة Shams أنه حرف شمسي، وأنت تعرف أننا ننطق الحرف الشمسي بعد "ال" مشددا سواء كانت عليه شدة أم لا.

١٠- معالجة حرف شمسي غير مشدد بعد "ال":

هذا التعبير مشابه للتعبير السابق، لكن الحرف الشمسي ليس عليه شدة، وكما شرحنا من قبل، يجب أن نستثني التاء من الحروف الشمسية حتى لا تلتبس مع صيغة "افتعل" مثل "التمس":

```
Edg_Al_MaybeSokon = ex.Edge + Al_MaybeSokon
shmsyNoTaa = ex.Validate(1, "ShamsNoTaa")
Edg_AL_ShmsyNotTaaEx = new Expression(
            Edg_Al_MaybeSokon + shmsyNoTaa, 1)
```

وهذا هو كود الدالة ShamsNoTaa، وهي مشابهة للدالة Shams لكننا نتأكد أن الحرف ليس تاء قبل أن نتأكد من وجوده في مصفوفة الحروف الشمسية:

```
Func ShamsNoTaa(str)
   hrf = str.Text
   return hrf != "ت" And find(ex.shamsiah, hrf)
EndFunc
```

والمريح هنا أن صيغة الاستبدال ستكون A_ التي استخدمناها مع التعبير النمطي السابق.

١١- معالجة حرف شمسي بعد "ال" ليست في البداية:

هذا التعبير النمطي مشابه للتعبير رقم ٩، لكنه غير ملزم بالبدء من بداية الكلمة، فقد عالج التعبير رقم ٩ تلك الحالة:

```
Al_shmsy_ShdhEx = new Expression(Al_shmsy_Shdh, 1)
```

وسيكون تعبير الاستبدال هو FkkShdhRep لأن كلا من الألف واللام ستحذفان هنا (تذكر أن هناك بادئة تسبقهما مثل "والشمس" التي تنطق "وشْشمس).

١٢- معالجة حرف شمسي بعد "ال" عليها سكون:

في هذا التعبير نحن متأكدون من وجود السكون، لكن غير متأكدين من وجود الشدة، ولن نشترط البدء من بداية الكلمة، لكن سنتأكد أن الحرف التالي للّام حرف شمسي ما عدا التاء:

```
Al_SokonRep = ex.AL + ex.Sokon
maybeShdh = ex.Maybe(ex.Shaddah)
ShmsyMaybeShdh = shmsyNoTaa + maybeShdh
AL_Sokon_ShmsyEx = new Expression(Al_SokonRep +
    ShmsyMaybeShdh, 1)
```

وسيكون تعبير الاستبدال هنا هو فك الحرف الشمسي.. هذا الحرف ستعثر عليه دالة الفحص وستضيفه في الخانة رقم ١ في مصفوفة المتغيرات، لهذا لن نستطيع استخدام تعبير الاستبدال FkkShdhRep هنا لأنه يتعامل مع المتغير رقم ٢:

```
FkkShmsyRep = ex.Expand(1)
```

١٣- إضافة سكون بعد "ال" في بداية الكلمة:

هذا التعبير مباشر، لأنه سيعتمد على متغير عرّفناه في القاعدة رقم ١٠:

```
Edg_Al_MaybeSokonEx = new Expression(
    Edg_Al_MaybeSokon, 1)
```

لكن تعبير الاستبدال الخاص به يعتمد على موضع الكلمة في الجملة، وقد سبق تعريفه في الوسيلة DoAroodWrite كالتالي:

```
_AlRep = Select(wordNum = 1, AHmz_L_SokonRep, Al_SokonRep)
```

وقد سبق تعريف المتغير Al_SokonRep في القاعدة رقم ١٢، ولم يتبق إلا تعريف المتغير AHmz_L_SokonRep:

```
L_Sokon = ex.Lam + ex.Sokon
AHmz_L_SokonRep = ex.AlephHmz + L_Sokon
```

١٤- إضافة سكون بعد أي حرف ألف في بداية الكلمة:

قبل أن نضع السكون، سنتأكد أن الحرف التالي للألف لا توجد عليه سكون أو أي تشكيل آخر.. لفعل هذا سننظر للأمام لحرفين بعد الألف ونرسلهما إلى الدالة CanAddSokon لفحصهما:

```
ifCanAddSokon = ex.LookForward(2, "CanAddSokon")
```

الآن، سنكتب التعبير النمطي الذي يستخدم دالة النظر للأمام، مع ملاحظة أنه يجب أن يبدأ من حافة الكلمة:

```
A_Hrf = ex.Aleph + ex.Any
A_Hrf_NoTsh = A_Hrf + ifCanAddSokon
Edg_A_HrfEx = new Expression(ex.Edge + A_Hrf_NoTsh, 1)
```

التعبير النمطي السابق يبحث عن الألف وحرف مجهول بعده (سيوضع في أول خانة في مصفوفة المتغيرات)، ثم يتأكد أن الحرف التالي له ليس علامة تشكيل.

والآن سنكتب تعبير الاستبدال، وكالعادة عند التعامل مع ألف في بداية الكلمة، يعتمد تعبير الاستبدال على موضع الكلمة في الجملة، لذا كتبناه في الدالة AroodWrite:

```
_AlephRep = Select(wordNum = 1, E_HrfSakenRep, A_HrfSakenRep)
```

لكن علينا هنا تعريف المتغيرين A_HrfSakenRep و E_HrfSakenRep، وهما يعتمدان على جزء مشترك سنضعه في المتغير HrfSakenRep:

```
HrfSakenRep = ex.var(1) + ex.Sokon
A_HrfSakenRep = ex.Aleph + HrfSakenRep
E_HrfSakenRep = ex.AlephMaksor + HrfSakenRep
```

يتكون تعبير الاستبدال HrfSakenRep من الحرف المجهول التالي للألف (رقم ١ في مصفوفة المتغيرات)، مع إضافة سكون بعده.. وكل ما يفعله التعبيران الآخران هو إضافة "ا" أو "إ" في البداية.

والآن لم يبق أمامنا إلا كتابة كود الدالة CanAddSokon، وهي تتأكد أن هناك حرف تالٍ للحرف المجهول التالي للألف، وأن هذا الحرف ليس تشكيلا.. لكن، نظرا لأننا سنحتاج لنفس هذه الدالة في مواضع أخرى لتفحص حرفين تاليين وليس حرفا واحدا، فسندمج الحالتين معا، ونتأكد من أن أول حرف ليس عليه تشكيل، وأن ثاني حرف ليس عليه سكون أو شدة (كما قلت سنحتاج لهذا في حالات لاحقة).. وكما ذكرنا سابقا، لو وجد محرك التعبيرات النمطية عددا من الحرف أقل من المطلوب فسيرسل النص الذي يجده حتى لو كان طوله صفرا (نصا فارغا).. هذا هو الكود:

```
Func CanAddSokon(str)
   L = str.Length
   if L = 0 or IsTashkeel(str[1])
      return false
   elseIf L = 1
      return true
   elseIf str[2] = ex.Sokon or str[2] = ex.Shaddah
      return False
   else
      return true
   end
EndFunc
```

كود هذه الدالة بسيط للغاية، لكنه يحتاج لبعض التركيز لأن منطق الشروط يعتمد على ترتيبها:

- في الشرط الأول لو كان طول النص صفرا (لا توجد حروف بعد الألف) فستعيد الدالة False.. انتبه إلى أن المعامل Or لن يفحص الشرط التالي له إلا إذا كان الشرط السابق له خاطئا، أي أن طول النص أكبر من صفر، وهناك على الأقل حرف واحد بعد الألف، وفي هذه الحالة لو كان تشكيلا فسنعيد False أيضا.
- في الشرط الثاني، لو كان طول النص = ١، فسنعيد True.. انتبه جيدا إلى أن الشرط السابق أكد لنا أن طول النص أكبر من صفر (يمكن أن يكون ١ أو ٢ أو ...)، وأكد لنا أيضا أن الحرف الأول ليس تشكيلا، لهذا لم نكرر التأكد

من هذا هنا.. لدينا حرف واحد فقط بعد الألف، ونحن واثقون أنه ليس تشكيلا، إذن فسنضع عليه السكون.. حتى الآن نحن لا نتعامل مع حالات عملية، لكن فقط نتجنب أخطاء الكتابة من المستخدم، كأن يكتب "اذ" بدلا من "إذ").

- إذا وصلنا للشرط الثالث، فهذا يعني وجود حرفين بعد الألف، أولهما ليس تشكيلا، لهذا لو كان الحرف الثاني سكونا أو شدة فسنعيد False.
- ما عدا كل ما سبق (المقطع Else) سنعيد True.

١٥- تحويل الألف إلى ألف مكسور في بداية الكلمة:

هذه القاعدة مكملة للقاعدة السابقة، فهي تتعامل مع الحالات التي لم نضع فيها سكون على الحرف التالي للألف، لهذا سنبحث فقط عن الألف في بداية الكلمة، لأننا ما زلنا نحتاج لتحويلها إلى ألف مكسورة الهمزة في أول الجملة:

Edg_AEx = new Expression(ex.Edge + ex.Aleph, 1)

وقد كتبنا تعبير الاستبدال في الدالة AroodWrite، وهو مباشر ولا يحتاج لتعريف أي متغيرات جديدة هنا:

_A = Select(wordNum = 1, ex.AlephMaksor, ex.Aleph)

تعبيرات حالات "لل":

فيما يلي سنكتب التعبيرات النمطية التي تعالج دخول اللام على "ال".. يمكن أن تكون اللام مفتوحة (تستخدم للتوكيد) أو مكسورة (تستخدم كحرف جر)، مع احتمال وجود فتحة أو كسرة بعد اللام الأولى، ولهذا سنستخدم الدالة ex.Maybe لمراعاة هذا.. سابقا كنا نرسل لهذه الدالة نصا واحدا، لكننا هنا سنرسل إليها مصفوفة، لنخبر محرك التعبيرات النمطية بأنه ربما يجد أحد عناصرها في هذا الموضع:

LL = ex.Lam + ex.Maybe([ex.Fat7ah, ex.Kasrah]) + ex.Lam

لاحظ أن ex.Maybe([ex.Fat7ah, ex.Kasrah]) هو في الحقيقة مجرد اختصار لـ:

ex.Maybe(ex.Fat7ah) + ex.Maybe(ex.Kasrah)

وهذه ليست أفضل صيغة للتعبير عن موضع واحد ربما توجد فيه فتحة أو كسرة أو لا

توجدان، لكن المحرك النمطي الخاص بنا لا يحتوي على صيغة كهذه، ولم أرَ ضرورة لتعقيد الأمر، ما دامت الصيغة السابقة تؤدي الغرض، لكن مع عيبين صغيرين:

- أنها تطابق موضعين، وهذا معناه أنها تضيف متغيرين مختلفين لمصفوفة المتغيرات (في الخانتين ١ و ٢ هنا)، ويجب مراعاة هذا في صيغ الاستبدال.
- أنها تسمح بوجود الفتحة والكسرة معا (فتحة تليها كسرة)، لكني لا أراه عيبا خطيرا، لأنه سيكون مجرد خطإ مطبعي من المستخدم، وستكون ميزة في البرنامج أنه يستطيع التغاضي عن هذا الخطأ، مع ملاحظة أن هذا لن ينطبق على حالة وجود كسرة تليها فتحة!

والآن، دعنا نستخدم المتغير LL في صيغة التعبيرات النمطية الخاصة بحالات "لل".

١٦- معالجة حرف شمسي بعد "للْ":

هذا التعبير النمطي مباشر جدا، فهو يتكون من أجزاء سبق لنا تعريفها مثل LL و shmsy و maybeShdh:

```
LL_Sokon_ShmsyEx = new Expression(LL + ex.Sokon +
      shmsy + maybeShdh, 1)
```

وسيكون تعبير الاستبدال بسيطا أيضا، فأنت تعرف أن اللام الشمسية لن تنطق، لهذا ستظل معنا اللام الأولى فقط، وسنفك تشديد الحرف الشمسي وهو محفوظ في الخانة الثالثة في مصفوفة المتغيرات، فقد حجزت الفتحة والكسرة أول خانتين:

```
L_FkkShmsyRep = ex.Lam + ex.Expand(3)
```

لاحظ أنني لم أضع التشكيل على اللام الأولى، تلافيا لحالة وجود الفتحة والكسرة معا، لكن لو شئت المحافظة على التشكيل، فعدل الكود السابق ليصير:

```
L_FkkShmsyRep = ex.Lam + ex.Var(1) + ex.Var(2) + ex.Expand(3)
```

ولكن انتبه إلى أن هذا سيؤدي إلى فشل بعض الاختبارات في الملف AroodWrite.Tests.ring، لأني كتبت الاختبارات باعتبار حذف تشكيل اللام الأولى، لهذا لو قررت الإبقاء عليه، فعليك إضافة التشكيل إلى نتيجة هذه الاختبارات حتى لا تفشل.

١٧- معالجة حرف شمسي مشدد بعد "ل":

في هذا التعبير ربما توجد السكون أو لا (ستوضع في الخانة الثالثة في مصفوفة المتغيرات)، لكن العلامة المميزة هي وجود شدة على حرف شمسي بعد "ل"، لهذا سنستخدم المتغير shmsy_Shdh الذي عرّفناه سابقا:

```
LL_MayBeSokon = LL + ex.Maybe(ex.Sokon)
LL_shmsy_ShdhEx = new Expression(LL_MayBeSokon +
    shmsy_Shdh, 1)
```

ولكتابة تعبير الاستبدال، لاحظ أن الحرف الشمسي في التعبير النمطي السابق سيوضع في الخانة الرابعة من مصفوفة المتغيرات:

```
L_FkkShdhRep = ex.Lam + ex.Expand(4)
```

١٨- معالجة حرف شمسي بعد "ل":

سنتعامل هنا مع "ل" التي قد تسبقها بعد البوادئ مثل "أ"، "ف"، "و".. لهذا سنرسل هذه البوادئ إلى الدالة ex.Maybe:

```
bwadee= ex.Maybe([ex.AlephHmz, ex.Faa, ex.Waw])
```

التعبير السابق سيبحث في ٣ مواضع متتالية عن احتمال وجود ٣ حروف مختلفة، وهذا معناه أنه سيضيف ثلاث خانات لمصفوفة المتغيرات.. في الحقيقة، هناك ميزة كبيرة في استخدام هذه الطريقة هنا، فهي تسمح أيضا بمطابقة كلمات تحتوي على حرفين قبل اللام (مثل "أفللشمس تنظر؟"، "أوللناس عهد؟")، أو حتى وجود الحروف الثلاثة معا!.. هذا هو التعبير النمطي:

```
Edg_Atf_LL_shmsy = new Expression(ex.Edge + bwadee +
    LL_MayBeSokon + ex.LookForward(2, "IsShamsy") +
    ex.Any + maybeShdh, 1)
```

هذا التعبير يبدأ من حافة الكلمة، التي ربما تبدأ ببعض البوادئ قبل "ل"، التي ربما تليها سكون، ثم أي حرف، وربما عليه شدة.

في التعبير السابق لسنا متأكدين من وجود السكون ولا الشدة، ولا أن الحرف المجهول حرف شمسي، لذا أرسلنا الحرف المجهول والحرف التالي له إلى الدالة IsShamsy، وهي دالة تنظر للأمام أي أنها ليست جزءا من التعبير النمطي،

ولكنها تتأكد من أن جزءا من الصيغة يحقق شرطا معينا، لهذا سيوجد ex.Any في التعبير السابق في الموضع التالي مباشرة لـ LL_MayBeSokon كأن ex.LookForward غير موجود في الصيغة.. هذا هو كود الدالة IsShamsy:

```
Func IsShamsy(str)
   L = str.Length
   if L < 2 or IsTashkeel(str[1])
      return false
   elseIf str[2] = ex.Shaddah
      return find(ex.shamsiah, str[1])
   elseIf IsTashkeel(str[2])
      return False
   else
      return find(ex.shamsiah, str[1])
   end
EndFunc
```

الدالة السابقة ستعيد False إذا لم توجد أي حروف بعد "ل" أو كان بعدها حرف واحد فقط، فهذه بالتأكيد ليس حالة اللام الشمسية.

كما أنها ستعيد False أيضا لو كان أول حرف علامة تشكيل.. تذكر أن الحرف الوحيد المسموح به على اللام الثانية في "ل" هو السكون، وأن النص الذي نفحصه يقع بعد السكون المحتملة.. لو وجدنا فتحة هنا، فهذا يعني أن النص يحتوي على "ل" + فتحة، أو يحتوي على "ل" + سكون + فتحة، وكلاهما غير مقبول.

إذا عبرنا الشرط السابق (الحرف الأول ليس علامة تشكيل)، وكان الحرف الثاني شدة، فعلينا أن نتأكد أن الحرف الأول هو حرف شمسي، لهذا جعلنا الدالة تعيد نتيجة البحث عن الحرف الأول في مصفوفة الحروف الشمسية، فإن لم يكن موجودا فستعيد صفرا وهذا يعني False، وإن كان موجودا فستعيد رقما أكبر من صفر وهذا يعني True.

فإن فشل الشرط السابق، فسنفحص إن كان الحرف الثاني تشكيلا، ونظرا لأننا استبعدنا الشدة في الشرط السابق، فمجرد وصولنا هنا يعني أننا نفحص وجود أي تشكيل ما عدا الشدة، فإن وجدنا هذا التشكيل فسنعيد False، فنحن هنا لا نتعامل

مع اللام الشمسية، وإنما مع لام أصلية في الكلمة، مثل "للطْفك" فاللام هنا أصلية من كلمة "لُطف"، وهذا يختلف عن "للطّفل فهنا دخلت اللام الشمسية على "طفل".

إذا فشلت كل الشروط السابقة ووصلنا إلى المقطع Else، فنحن الآن متأكدون أننا نتعامل مع حرفين عاديين بعد "لـ" (ليسا علامتي تشكيل)، لهذا سنعيد نتيجة البحث عن الحرف الثاني في مصفوفة الحروف الشمسية، أي أن الدالة ستعيد True لو كان الحرف الثاني شمسيا.

ولكتابة تعبير الاستبدال الخاص بهذا التعبير النمطي، يجب علينا إضافة البوادئ الموجودة في الخانات الثلاث الأولى في مصفوفة المتغيرات، مع ملاحظة أن الحرف المشدد التالي ل "لـ" سيكون موجودا في الخانة الثامنة في مصفوفة المتغيرات: فالخانات الثلاث الأولى محجوزة للبوادئ، ثم خانتان للفتحة والكسرة، ثم خانة للسكون، ثم خانة لدالة النظر للأمام، ثم الحرف الشمسي في الخانة الثامنة:

```
Atf_LRep = ex.var(1) + ex.var(2) + ex.var(3) + ex.Lam
Atf_L_FkkShdhRep = Atf_LRep + ex.Expand(8)
```

١٩- إضافة سكون بعد "ل":

هذا التعبير سيبحث عن "لـ" التي قد تسبقها بوادئ، ولا يوجد بعدها أي تشكيل، والحرف التالي لها ليس عليه شدة أو سكون.. الشرطان الأخيران يمكن التأكد منهما باستخدام دالة النظر للأمام ifCanAddSokon التي كتبناها سابقا، وهنا يتضح لماذا أضفت لها شرط عدم وجود سكون أو شدة على الحرف التالي ل "ل":

```
Edg_Atf_LLEx = new Expression(ex.Edge + bwadee +
    LL + ifCanAddSokon, 1)
```

ولا جديد في تعبير الاستبدال يستحق الشرح:

```
Atf_LL_SokonRep = Atf_LRep + L_Sokon
```

٢٠- إضافة سكون على الواو أو الياء إذا سبقهما فتحة:

سنستخدم هنا دالة جديدة هي ex.OneOf، وسنرسل إليها مصفوفة، لنخبرها أننا نتوقع وجود أحد عناصر هذه المصفوفة في هذا الموضع.. في حالتنا هذه سنتعامل

مع فتحة تليها إما الياء وأما الواو:

```
Fat7ah_WawYaa = ex.Fat7ah + ex.OneOf([ex.Waw, ex.Yaa])
```

وللعلم، فإن الصيغة المناظرة لهذه الدالة في التعبير النمطي تشبه صيغة المصفوفة الخاصة برينج، وفي حالتنا هذه سيكون التعبير الناتج عن الكود السابق هو:

[يَ,وَ]

وسيضاف النص الذي تعثر عليه هذه الصيغة إلى مصفوفة المتغيرات.

لكن ما زالت هناك شروط يجب التحقق منها لهذا سنرسل الحرف التالي للواو إلى الدالة AfterWaw للتأكد من أنه ليس ألف مد أو تشكيلا:

```
Fat7ah_WawYaaEx = New Expression(Fat7ah_WawYaa +
        ex.LookForward(1, "AfterWaw"), 1)
```

هذا هو كود هذه الدالة:

```
Func AfterWaw(str)
   if str = "" return true end
   hrf = str[1]
   If hrf = ex.Aleph or IsTashkeel(hrf)
      return false
   else
      return true
   end
EndFunc
```

وأخيرا، هذا هو تعبير الاستبدال، حيث المتغير رقم ١ يحمل الواو أو الياء تبعا لما عثرت عليه الصيغة OneOf:

```
Fat7ah_WawYaaRep = ex.Fat7ah + ex.var(1) + ex.Sokon
```

تدريب:

الفرق بين ex.OneOf([ex.Waw, ex.Yaa]) و ex.Maybe([ex.Waw, ex.Yaa]) هو أن التعبير OneOf يجب أن يعثر على أحد الحرفين في الموضع الحالي (وإلا فشلت عملية البحث)، ولهذا يحجز خانة واحدة فقط في مصفوفة المتغيرات يضع فيها الحرف الذي وجده، بينما قد يجد التعبير Maybe أحد الحرفين أو كليهما (بنفس الترتيب) أو لا يجد أيا منهما، وهو يحجز خانتين في مصفوفة المتغيرات.. الحقيقة أن

محرك التعبيرات النمطية الخاص بنا ينقصه التعبير MaybeOneOf الذي يستقبل مصفوفة نصوص ويفحص احتمال وجود أحدها في الموضع الحالي مع تقبل عدم وجود أي منها، مع حجز خانة واحدة في مصفوفة المتغيرات يوضع فيها الحرف الذي عثرنا عليه أو نص فارغ إن لم نعثر على أي حرف.. لو شئت أن تضيف هذه الصيغة لمحرك التعبيرات النمطية، فيمكنك تمثيلها بوضع العلامة ~ قبل مصفوفة النصوص كالتالي:

~[str1, str2, str3, …. , strN]

وسأترك لك كتابة الكود الذي يحلل هذه الصيغة في محرك التعبيرات النمطية كتدريب، ومن ثم يمكنك استخدام هذه الصيغة الجديدة لإعادة كتابة التعبير النمطي الذي يبحث عن "لل" مع احتمال وجود فتحة أو كسرة بعد اللام الأولى، مع مراعاة التغييرات التي ستحدث في صيغ الاستبدال بسبب اختلاف عدد خانات مصفوفة المتغيرات.

٢١- معالجة الأسماء التي تبدأ بألف مد يليها حرف ساكن ومعرفة بأل:

لجعل هذا التعبير يعالج الكلمات التي تبدأ بـ "ال" و "لل" مثل "الاسْتثناء" و"للاسْتثناء"، سنجعل وجود الألف في "ال" محتملا باستخدام الدالة Maybe.. فوجود الألف سيجعلنا نحذفها، لكن عدم وجودها لن يؤثر في شيء، لأن العلامة المميزة هنا هي وجود لام ساكنة تليها ألف مد، ونحن نضمن وجود السكون على اللام دائما لأننا عالجناها من قبل في حالات "ال" و "لل":

```
maybeA_L_Sokon = ex.Maybe(ex.Aleph) + L_Sokon
L_Sokon_A_Hrf = maybeA_L_Sokon + ex.Aleph + ex.Any
maybeA_L_Sokon_A_SakenEx = new Expression(
      L_Sokon_A_Hrf + ex.Sokon, 1)
```

وهذا هو تعبير الاستبدال:

```
L_Kasrah_HrfSakenRep = ex.Lam + ex.Kasrah +
      ex.Var(2) + ex.Sokon
```

٢٢- معالجة الأسماء التي تبدأ بألف ومعرفة بأل:

استكمالا للحالة السابقة، لو لم يضع المستخدم السكون على الحرف التالي للألف، فسنضيفها نحن، لكن يجب التأكد أولا من عدم وجود تشكيل على الحرف التالي للألف، لهذا سننظر لحرفين للأمام ونرسلهما إلى الدالة CanAddSokon التي استخدمناها أكثر من مرة سابقا:

```
maybeA_L_Sokon_A_HrfEx = new Expression(
      L_Sokon_A_Hrf + ifCanAddSokon, 1)
```

وسيكون تعبير الاستبدال هو L_Kasrah_HrfSakenRep الذي استخدمناه في الحالة السابقة.

٢٣- فك الشدة:

لا جديد في هذا التعبير النمطي، ولا صيغة استبداله:

```
Hrf_ShdhEx = New Expression(ex.Any + ex.Shaddah, 0)
Hrf_Sokon_HrfRep = ex.Expand(1)
```

٢٤- حذف أي ألف وسط الكلمة بعدها حرف ساكن:

في هذا التعبير يجب ألا توجد الألف في بداية الكلمة، وألا يوجد الحرف الساكن التالي لها في نهاية الكلمة.. لفرض هذين الشرطين، سنستخدم الخاصية ex.NotEdge، وواضح من اسمها أنها عكس الخاصية ex.Edge، وهي تضيف للتعبير النمطي الرمز &، الذي يخبر محرك التعبير النمطي بضرورة وجود حرف واحد على الأقل في بداية الكلمة (أو نهايتها).. ومرة أخرى، هذا الرمز لن يكون جزءا من النص الذي نبحث عنه، بل مجرد شرط إضافي، ويمكنك اعتباره إحدى صيغ النظر للأمام أو الخلف.. في الحقيقة يمكننا كتابة دالة نظر للأمام أو الخلف للقيام بوظيفتي الخاصيتين ex.Edge و ex.NotEdge، لكن واضح طبعا أن استخدامها أسهل وأكثر اختصارا.. هذا هو التعبير النمطي:

```
NotEdg_A_Saken_NotEdgEx = new Expression(ex.NotEdge +
      A_Hrf + ex.Sokon + ex.NotEdge, 0)
```

لقد أرسلنا صفرا (False) إلى المعامل الثاني للتعبير النمطي، وهذا يعني أن عليه البحث عن كل مواضع هذه الصيغة في النص، ولا يتوقف بعد أول نتيجة يعثر عليها.. بتعبير آخر: ربما توجد ألف يليها حرف ساكن أكثر من مرة في النص.
وسيكون تعبير الاستبدال الخاص بهذا التعبير النمطي، هو نفسه HrfSakenRep الذي سبق تعريفه.

٢٥- تنوين فتح تليه ألف مد:

هذا التعبير النمطي في غاية الوضوح:

tnwn_AlephEx = new Expression(ex.TanweenFat7 + ex.Aleph, 1)

وهذا هو تعبير الاستبدال الخاص به، والذي سنستخدمه أيضا مع باقي حالات تنوين الفتح:

Ft7h_Noon_SokonRep = ex.Fat7ah + ex.Noon + ex.Sokon

٢٦- ألف مد يليها تنوين فتح:

Aleph_tnwnEx = new Expression(ex.Aleph + ex.TanweenFat7, 1)

٢٧- تنوين فتح تليه ألف لينة:

tnwn_LeenEx = new Expression(ex.TanweenFat7 + ex.AlephLeen, 1)

٢٨- ألف لينة يليها تنوين فتح:

Leen_tnwnEx = new Expression(ex.AlephLeen + ex.TanweenFat7, 1)

٢٩- تنوين فتح:

tnwnFt7Ex = new Expression(ex.TanweenFat7, 1)

٣٠- تنوين ضم:

tnwnDmEx = new Expression(ex.TanweenDamm, 1)
Dmh_Noon_SokonRep = ex.Dammah + ex.Noon + ex.Sokon

٣١- تنوين كسر:

tnwnKsrEx = new Expression(ex.TanweenKasr, 1)
Ksrh_Noon_SokonRep = ex.Kasrah + ex.Noon + ex.Sokon

وبهذا نكون قد انتهينا من تعريف كل التعبيرات النمطية بحمد الله، وفهمنا كل الكود الموجود في الملف AroodWrite.ring.

تدريب:

قد يخطي المستخدم ويكتب تشكيلات زائدة (ككتابة تنوين الضم مرتين متتاليتين مثلا) وهذا سيؤدي إلى نتائج خاطئة في البرنامج.. سيكون مفيدا لو كتبت بعض التعبيرات النمطية لحذف أي تشكيل خاطئ، مثل:

- شدتين متتاليتين: احذف الشدة الزائدة.
- شدة تليها سكون: لو كان هذا هو الحرف الأخير فاتركها، وإلا فاحذف السكون.
- أي تشكيل غير الشدة يليه تشكيل آخر: احذف التشكيل الزائد.
- علامة تنوين في وسط الكلمة (مع استثناء التنوين الذي يوضع قبل ألف المد أو الألف اللينة): احذفها أو حول تنوين الضم إلى ضمة وتنوين الفتح إلى فتحة وتنوين الكسر إلى كسرة او لم يسبقها تشكيل آخر.
- ألف مد عليها أي تشكيل غير تنوين الفتح أو السكون: احذف التشكيل.
- ياء في آخر الكلمة عليها ضمة: إما أن تضيف شدة قبل الضمة، وإما أن تحذف الضمة.

وهكذا، يمكنك معالجة أي حالات تشكيل أخرى غير مقبولة.

وإذا لم تكن متقبلا لفكرة تصحيح التشكيل، فمكن الممكن أن تعرض رسالة للمستخدم تسأله عن القرار الصحيح الذي يجب اتخاذه بخصوص التشكيل الزائد.

وبالنسبة لنا هنا، سنواصل كتابة البرنامج مفترضين أن المستخدم يكتب الأبيات بشكل صحيح.

– ٦ –

الرموز الصوتية

نجحنا حتى الآن في الحصول على الكتابة العروضية للبيت الشعري.. الخطوة التالية سهلة جدا، وهي تحويل هذه الكتابة العروضية إلى رموز تعبر عن أصوات الحروف.

الرموز الصوتية:

لدينا في الشعر صوتان فقط:

- الصوت الساكن: ونرمز له بدائرة أو صفر 0 .. وكما ذكرنا سابقا، يأتي الصوت الساكن من حروف المد واللين، ومن الحروف التي عليها سكون.. تذكر أننا تخلصنا في مرحلة الكتابة العروضية من الشدة بفك الحرف المشدد إلى حرفين أولهما ساكن، وتخلصنا أيضا من التنوين بتحويله إلى نون ساكنة، لهذا لن نقلق بشأنهما.
- الصوت المتحرك: ويأتي من باقي الحروف، ونرمز له بشرطة قائمة | .. ولا تهمنا الفتحة والضمة والكسرة إلا إذا وُضِعَت على الياء والواو للدلالة على أنهما حرفان متحركان.. أي أننا سنفترض أن كل حرف عادي (غير ألف المد وغير الياء والواو في أي موضع غير بداية الكلمة) هو حرف متحرك إذا لم يكن عليه سكون.

والآن، دعنا نطبق هذا الكلام.

فئة الرموز Romooz Class:

أنشئ ملفا جديدا في محرر رينج اسمه Romooz.ring، وأضف إليه فئة اسمها Romooz.. وكما هو معتاد، يجب تحميل فئة النصوص العربية في بداية الملف.. هذا هو الهيكل المفرغ لهذه الفئة:

```
load "aString.ring"

class Romooz
    Harakaat = []
    Horoof = []

    Func init(words, eshbaa3)
        // …..
    EndFunc

    Func Horoof(lengthes)
        return split(Horoof, lengthes)
    EndFunc

    Func Harakaat(lengthes)
        return split(Harakaat, lengthes)
    EndFunc

    Func GetOrgHarakaat( )
        return split(Harakaat, originalLengths)
    EndFunc

    Func ToString( )
       result = ""
       L = len(Harakaat)
       for i = 1 to L
          h = Harakaat[i]
          if h != " "
             result += h
          end
       next
       return result
    EndFunc
```

```
Private
  originalLengths = []

  Func Split(list, lengthes)
     // .....
EndFunc

EndClass
```

تحتوي الفئة Romoz على العناصر التالية:

- مصفوفة الحركات Harakaat: سنضع فيها الرموز الصوتية، كل رمز في خانة.
- مصفوفة الحروف Horoof: سنضع فيها الحروف المناظرة لكل رمز صوتي في مصفوفة الحركات.. ويمكن أن تحتوي كل خانة على حرف واحد أو اثنين (حرف عليه علامة تشكيل).. تذكر أننا في الكتابة العروضية لم نفك همزة المد "آ" إلى حرفين: "أ" تليها "ا"، لأن كتابة "أا" تبدو مربكة من وجهة نظري.. وفي كلتا الحالين، سينتج عن همزة المد الرمزان الصوتيان 0/ وسيضافان في خانتين متتاليتين في مصفوفة الحركات، فكيف سنملأ الخانتين المناظرتين لهما في مصفوفة الحروف؟.. اخترت هنا أن أضع في الخانة الأولى "آ"، وأترك الخانة الثانية فارغة.
- دالة الإنشاء init، التي تستدعيها رينج عند إنشاء نسخة جديدة من الفئة باستخدام الكلمة new.. ولهذه الدالة معاملان، الأول يستقبل مصفوفة الكلمات words التي تحتوي على الكتابة العروضية لكلمات البيت، والثاني متغير منطقي اسمه "إشباع" eshbaa3، وسأشرح وظيفته بعد قليل.
- الدالة Horoof والدالة Harakaat، وسنستخدمهما في تقطيع البيت عروضيا كما سأشرح بعد قليل.
- الدالة GetOrgHarakaat: تعيد مصفوفة الرموز التي تناظر الكلمات الأصلية (بدون تقطيع عروضي).. لاحظ أننا سنخزن الأطوال الأصلية لرموز كل كلمة في مصفوفة خاصة اسمها originalLengths.
- الدالة ToString: وهي تدمج خانات المصفوفة Harakaat في نص واحد مع

التخلص من أي مسافات، وتعيد إليك هذا النص.. كود هذه الدالة واضح ولا يحتاج لشرح إضافي.

- دالة التقسيم Split، وهي موجودة في الجزء الخاص Private لذا لا يمكننا استدعاؤها إلا من داخل الفئة.. هذه الدالة تقسم المصفوفة المرسلة إليها إلى أجزاء، وتعتمد عليها الدوال Horoof و Harakaat و GetOrgHarakaat في القيام بوظائفها.

الدالة Init:

سنبدأ كالعادة بوضع عدد الكلمات في متغير، وسنعرّف أيضا متغيرا اسمه hL لنضع فيه طول مصفوفة الحركات، وسيكون صفرا في البداية:

```
L = len(Words)
hL = 0
```

ثم سنكتب حلقة تكرار للمرور على كل كلمة للحصول على الرموز الصوتية الخاصة بها وإضافتها للمصفوفة Harakat:

```
For i = 1 To L
   word = Words[i]
   wordLen = Len(word)
   hrf = ""
   nextHrf = word[1]
   // كود ١
Next #i
```

في الكود السابق عرّفنا بعض المتغيرات الأساسية:

- word: يحمل الكلمة الحالية.
- wordLen: يحمل طول الكلمة الحالية.
- hrf: يحمل حرفا من الكلمة الحالية، وسيكون فارغا في البداية دلالة على أننا نتعامل مع الحرف رقم صفر في الكلمة (وهو غير موجود بالطبع).
- والمتغير nextHrf سيحمل الحرف التالي له، وسيبدأ بالحرف رقم ١ في الكلمة.. الحكمة في هذا أننا سنكتب حلقة تكرار داخلية (عدادها j) للمرور على كل حرف

في الكلمة الحالية، وسنحتاج في الكود للنظر للحرف السابق word[j-1] والحرف الحالي word[j] والحرف التالي word[j+1].. لكني فضلت استخدام متغيرات بأسماء واضحة (prevHrf و hrf و nextHrf) بدلا من التعامل مع هذه الخانات مباشرة، ليكون الكود أسهل في الفهم، ولتجنب تكرار عمليات طرح وجمع لا ضرورة لها.

وكل ما سنفعله في بداية كل لفة في حلقة التكرار، هو أن نزحزح قيم المتغيرات للخلف، فالحرف الحالي سيصير الحرف السابق في اللفة الجديدة، والحرف التالي سيصير الحرف الحالي، والحرف التالي سيأخذ قيمة جديدة من المصفوفة إن كانت هناك خانة تالية، وإلا فسنضع فيه نصا فارغا.. أضف الكود الحالي في موضع التعليق "كود ١":

```
For j = 1 To wordLen
   prevHrf = hrf
   hrf = nextHrf
   if j = wordLen
      nextHrf = ""
   else
      nextHrf = word[j + 1]
   end
   // كود ٢
Next #j
```

وبعد الانتهاء من المرور على كل حروف الكلمة والخروج من حلقة التكرار الداخلية، سنضيف مسافة لكل من مصفوفة الحروف ومصفوفة الرموز لنفصل بين الكلمات.. سيجعلنا هذا نعرض النتائج للمستخدم بشكل أفضل كما سنرى لاحقا:

```
Horoof + " "
Harakaat + " "
```

لاحظ أن هذا سيضيف مسافة زائدة بعد الكلمة الأخيرة في البيت، لهذا علينا بعد انتهاء حلقة التكرار الخارجية (التي عدادها i) أن نحذف آخر خانة من مصفوفة الحروف ومصفوفة الرموز.. سأكتب الكود الذي يفعل هذا لاحقا.

والآن سنحسب عدد الحركات التي أضفناها إلى مصفوفة الحركات (بطرح طولها السابق المحفوظ في المتغير hL من طولها الفعلي حاليا وطرح ١ أيضا بسبب المسافة الزائدة التي أضفناها فلا نريد حسابها ضمن الطول)، ونضيف هذا العدد إلى المصفوفة originalLengths، ثم نحدّث قيمة المتغير hL ليحتوي على الطول الجديد لمصفوفة الحركات، لكن هنا يجب أن تدخل المسافة في العد لهذا لم نطرح ١:

```
hL2 = len(Harakaat)
originalLengths + (hL2 - hL - 1)
hL = hL2
```

الآن نحن جاهزون للحصول على الرموز الصوتية، حيث سنكتب كل الكود التالي في موضع التعليق "كود ٢" داخل حلقة التكرار الداخلية (التي عدادها j):

```
If hrf = Letters.HamzetMadd
   Horoof + hrf + ""
   Harakaat + "/" + "0"

ElseIf hrf = Letters.Fat7ah or hrf = Letters.Kasrah or
        hrf = Letters.Dammah or hrf = Letters.Sokon
   n = Len(Horoof)
   Horoof[n] += hrf

ElseIf hrf = Letters.Aleph or NextHrf = Letters.Sokon
   Horoof + hrf
   Harakaat + "0"

ElseIf hrf = Letters.AlephLeen or hrf = Letters.Yaa or
      hrf = Letters.Waw
   // كود ٣
Else
   Horoof + hrf
   Harakaat + "|"
End
```

الكود السابق بسيط للغاية، فهو يتكون من جملة شرط مركبة If..ElseIf..Else لفحص الشروط التالية:

- لو كان الحرف الحالي همزة مد "آ" فسنضيف خانتين لمصفوفة الحروف، الخانة

الأولى سنضع فيها همزة المد (وهي موجودة في المتغير hrf) والثانية سنتركها فارغة.. وتسمح رينج أن نضيف الخانتين للمصفوفة في سطر واحد:

```
Horoof + hrf + ""
```

وسنضيف أيضا خانتين لمصفوفة الحركات، فيهما الرمزان | و 0 على التوالي:

```
Harakaat + "|" + "0"
```

- أما لو كان الحرف الحالي علامة تشكيل، فيجب وضعها مع الحرف السابق في نفس الخانة في مصفوفة الحروف، ولن نضيف أي شيء لمصفوفة الحركات.. انتبه أن الحرف السابق موجود في آخر خانة في مصفوفة الحروف (الخانة التي رقمها = طول المصفوفة):

```
ElseIf hrf = Letters.Fat7ah or hrf = Letters.Kasrah or
    hrf = Letters.Dammah or hrf = Letters.Sokon
  n = Len(Horoof)
  Horoof[n] += hrf
```

- أما لو كان الحرف الحالي ألفا "ا"، أو كان الحرف التالي علامة سكون، فسنضيف الحرف الحالي لمصفوفة الحروف (ولا تقلق بخصوص السكون، فستضاف في اللفة التالية بسبب جملة الشرط السابقة)، ونضيف الرمز 0 لمصفوفة الحركات:

```
ElseIf hrf = Letters.Aleph or NextHrf = Letters.Sokon
  Horoof + hrf
  Harakaat + "0"
```

- أما لو كان الحرف الحالي ألفا لينة أو ياء أو واوا، فسنحتاج لكتابة بعض الكود لمعرفة هل هذا الحرف من حروف اللين أو المد، أم أنه حرف متحرك، لهذا سنترك هذه الحالة لنشرحها لاحقا بالتفصيل:

```
ElseIf hrf = Letters.AlephLeen or hrf = Letters.Yaa or
    hrf = Letters.Waw
  كود ٣ //
```

- أخيرا، نصل إلى المقطع Else الذي يعالج الحالات التي لا تنطبق عليها الشروط السابقة.. وما دمنا وصلنا هنا، فهذا معناه أن الحرف متحرك، لذا سنضيفه لمصفوفة الحروف ونضيف الرمز | لمصفوفة الحركات:

```
Else
   Horoof + hrf
   Harakaat + "|"
End
```

لم يبق إذن إلا معالجة حالة الألف اللينة والياء والواو.. أضف هذا الكود في الموضع "كود ٣"، وهو يبدأ بإضافة الحرف لمصفوفة الحروف، والعمل كله سيتركز حول معرفة إن كان الحرف ساكنا أم متحركا:

```
Horoof + hrf
If j = 1 or Harakaat[Len(Harakaat)] = "0" or
   nextHrf = Letters.Fat7ah or nextHrf = Letters.Dammah or
   nextHrf = Letters.Kasrah or nextHrf = Letters.Aleph or
   nextHrf = Letters.Waw or nextHrf = Letters.Yaa or
   nextHrf = Letters.AlephLeen
      Harakaat + "|"
ElseIf j < WordLen - 1 and Word[j + 2] = Letters.Sokon
   If i = L And j + 2 = Wordlen
      Harakaat + "0"
   Else
      Harakaat + "|"
   End
Else
   Harakaat + "0"
End
```

دعنا نفهم هذا الكود:

- أول شرط في الكود السابق مكون من عدد كبير من الشروط، وكلها تؤدي إلى اعتبار الواو أو الياء أو الألف اللينة حروفا متحركة (تذكر أن بعض المستخدمين قد يكتب الألف اللينة بدلا من الياء في نهاية الكلمة).. وهذه الشروط هي:
  ١- إذا كان الحرف في بداية الكلمة.
  ٢- إذا كان الحرف السابق ساكنا.
  ٣- إذا كان الحرف التالي فتحة أو ضمة أو كسرة أو ألف مد أو ألفا لينة أو واوا أو ياء.

ملحوظة: لن نعالج هنا حالة إذا كان الحرف التالي للواو أو الياء سكونا، فقد

عالجناها في الشرط الذي يفحص وجود سكون تالية لأي حرف بشكل عام.

- الشرط التالي يعالج حالة وجود حرف عليه سكون بعد الواو.. هنا يجب أن نتأكد أن المصفوفة تحتوي على خانتين على الأقل بعد الحرف الحالي باستخدام الشرط j < WordLen – 1 وأن الحرف الموجود في الخانة التي تلي الخانة الحالية بموضعين هو سكون Word[j+2] = Letters.Sokon.. هنا أمامنا احتمالان:

١- هذه هي آخر كلمة في البيت (الشرط i = L) والسكون هي آخر حرف في الكلمة (الشرط j+2 = Wordlen)، وكما قلنا، يمكن أن تكون القافية ساكنة وفي هذه الحالة لا مانع من التقاء ساكنين، لهذا سنعتبر الياء أو الواو حرفي مد ساكنين.

٢- غير هذا، سنعتبر الواو والياء حرفين متحركين لمنع التقاء ساكنين في منتصف الكلمة أو منتصف البيت.

- أخيرا نصل للمقطع Else، وبفشل الشرطين السابقين في المقطعين If و ElseIf، لم يبق أمامنا إلا اعتبار الواو والياء والألف اللينة حروف مد ساكنة عادية.

بهذا نكون قد أنهينا الكود الخاص بحلقتي التكرار كاملا، وبقيت أمامنا خطوة واحدة فقط، هي استخدام المعامل الثاني للدالة Init واسمه "إشباع" eshbaa3.

إذا أرسلنا إلى هذا المعامل القيمة True وكانت آخر كلمة في البيت تنتهي بحرف متحرك، فسنشبع حركة هذا الحرف بمدها في النطق لتناسب القافية.. هذا معناه أن الكلمة الأخيرة لو كانت تنتهي بالفتحة فسنضيف ألفا، ولو كانت تنتهي بالضمة فسنضيف واوا، ولو كانت تنتهي بالكسرة فسنضيف ياء، ولو كانت تنتهي بحرف عادي (غير الألف والواو والياء) وليس عليه تشكيل، فسنضيف خانة فارغة لأننا لا نملك معلومات كافية عن نطق الكلمة.. وفي كل هذه الحالات، سنضيف رمزا ساكنا 0 في الخانة المناظرة في مصفوفة الحركات.

لفعل هذا، أضف هذا الكود بعد نهاية كود حلقتي التكرار (بعد Next #i)، وهو واضح ولا يحتاج لتفسير، وتذكر ما قلته سابقا عن ضرورة حذف المسافة الزائدة من آخر

خانـة فـي كـل مـن مصـفوفتي الحـروف والرمـوز.. يمكـن حـذف خانـة مـن المصـفوفة باستخدام الدالة Del، وهي تستقبل المصفوفة، ورقم الخانة المراد حذفها:

```
Del(Horoof, Len(Horoof))
L = Len(Harakaat)
del(Harakaat, L)
L-- \\ إنقاص عدد الخانات بواحد بعد حذف الخانة
```

```
If eshbaa3 and Harakaat[L] = "|"
   Switch hrf
      case Letters.Fat7ah
         Horoof + "ا"
      case Letters.Kasrah
         Horoof + "ي"
      case Letters.Dammah
         Horoof + "و"
      else
         Horoof + ""
   Off
   Harakaat + "0"
End
```

التقطيع العروضي والدالة Split:

بعد أن نكتشف وزن القصيدة، ونعرف البحر الذي تنتمي لـه، سنحتاج إلـى تقطيعها عروضيا على تفعيلات هذا البحر.. مثال:

الشطرة	هلا سألتِ القوم يا ابنةَ مالكٍ
الكتابة العروضية	هلْلا سألتلْ قوم يبْ نتَ مالكنْ
الرموز الصوتية	٠|٠| ٠|٠|| |٠| ٠| || |٠| ٠|٠|

سيخبرنا البرنامج لاحقا أن هذه الشطرة تنتمي إلى البحر الكامل، وأنها على وزن:

مستفعلنْ مستفعلنْ متفاعلنْ

وهنـا يجـب علينـا تقطيـع الكتابـة العروضـية لتناسـب هـذا الـوزن.. أي يجـب أن نعـرض

للمستخدم ثلاث كلمات تتكون كل منها من ٧ حروف (بدون حساب التشكيل) لتناسب التفعيلات (مستفعلنْ مستفعلنْ متفاعلنْ)، وبالمثل علينا تقسيم الرموز الصوتية.. لفعل هذا أضفنا الدالة Horoof لفئة الرموز، وكل ما سنفعله هو أن نرسل إليها مصفوفة تحتوي على أطوال التفعيلات:

```
x = r.Horoof([7, 7, 7])
```

وبالمثل، يمكننا استخدام الدالة Harakaat لتقطيع الرموز الصوتية:

```
y = r.Harakaat([7, 7, 7])
```

وستبدو النتائج التي نحصل عليها كالتالي:

هلْلا سألْ – تلْ قوم يبْ – نتَ مالكنْ

٠||٠| || – ٠| |٠| ٠| – ٠|٠|

لعلك لاحظت فائدة تركنا للمسافات في مصفوفتي الحروف والرموز، فهي تجعل التقطيع العروضي أوضح وأسهل في القراءة، عما لو كتبت:

هلْلاسألْ – تلْقوميبْ – نتَمالكنْ

أنت تعلم أن الدالتين Horoof و Harakaat تستدعيان الدالة Split لتنفيذ عملية التقطيع.. دعنا إذن نكتب كود هذه الدالة.. هذا هو هيكلها العام:

```
Func Split(list, parts)
  L = len(list)
  partsCount = Len(parts)
  result = [ ]
  i = 1

  for j = 1 to partsCount
    // كود التقطيع
  next

  return result
EndFunc
```

يستقبل المعامل List المصفوفة التي نريد تقسيمها وسنضع طولها في متغير اسمه L، ويستقبل المعامل parts مصفوفة تحتوي على أطوال الأجزاء المراد تقطيعها، وسنضع عدد هذه الأجزاء في متغير اسمه partsCount، أما نتيجة التقطيع فسنضعها في

مصفوفة اسمها result.

وقد عرّفنا متغيرا اسمه i لنستخدمه كعداد للمرور عبر خانات المصفوفة List، ووضعنا فيه ١ ليبدأ من أول خانة.. لاحظ أننا لن نكتب حلقة تكرار لهذه العملية، لكننا سنزيد قيمة المتغير i يدويا، لأننا لا نعرف بالضبط عدد الخانات التي ستعطينا الجزء المطلوب.. تذكر مثلا أن مصفوفة الحروف قد تحتوي على بعض الخانات الفارغة أو خانات فيها مسافة، وهذين النوعين من الخانات لن يدخلا في العد وحساب طول الجزء، لأن ما يعنينا هنا هو الحروف التي تؤثر في موسيقى الشعر وليس الخانات الفارغة والمسافات.

وقد استخدمنا حلقة تكرار For Next (عدادها j) للمرور عبر مصفوفة الأجزاء parts، والقيام بعملية التقسيم.

دعنا إذن نكتب "كود التقطيع"، ونضعه داخل حلقة التكرار السابقة.. كل المطلوب هو الحصول على جزء من المصفوفة List طوله يساوي طول الجزء الموجود في الخانة الحالية في مصفوفة الأجزاء parts[j].. وسنضع هذا الطول في متغير اسمه partLen ليكون الكود أوضح:

```
partLen = parts[j]
```

والجزء الذي سنقطعه سنضعه في متغير نصي اسمه part، سيكون فارغا في البداية:

```
part = ""
```

وسنعرف متغيرا اسمه count يبدأ ب ١ ونزيد قيمته كلما أخذنا حرفا من المصفوفة List وأضفناه للمتغير part.. وستستمر هذه العملية إلى أن تصير قيمة المتغير count أكبر من طول الجزء المطلوب:

```
count = 1
While count <= partLen
   // كود اقتطاع جزء
end #while

if part != ""
   result + part
end
```

انتبه لجملة الشرط الموجودة بعد الحلقة while.. هذه الجملة تضيف الجزء الذي حصلنا عليه إلى مصفوفة الناتج بشرط ألا يكون هذا الجزء فارغا.

دعنا إذن نكتب كود حلقة التكرار الداخلية:

في البداية سنتأكد أن العداد i الذي نستخدمه للمرور عبر خانات المصفوفة List لم يتجاوز طولها، وإلا كان علينا أن نضيف الجزء الذي حصلنا عليه حتى الآن إلى مصفوفة الناتج، ونخرج من حلقتي التكرار For و While معا.. هذا ممكن في رينج باستخدام الأمر exit متبوعا بعدد حلقات التكرار المراد الخروج منها، وفي حالتنا هذه سنستخدم الأمر exit 2:

```
if i > L
  if item != ""
    result + part
  end
  exit 2
end
```

مرة أخرى تذكر أن الأوامر Return و Loop و Exit تغنينا عن كتابة المقطع Else لجملة الشرط، لأن رينج لن تصل إلى الكود التالي لجملة الشرط إلا إذا فشل الشرط.. لهذا إذا فشل الشرط السابق، فمهمتنا أن نقرأ الحرف الموجود في الخانة رقم i في المصفوفة List، فإن لم يكن مسافة فسنضيفه للمتغير part ونزيد المتغير count بمقدار ١.. أما إذا كان مسافة فلن نعدها (لن نزيد قيمة المتغير count)، لكن رغم هذا سنضيف هذه المسافة للمتغير part بشرط ألا تكون في بداية الكلمة.. نستطيع أن نعرف هذا من ملاحظة أن قيمة المتغير count تكون ١ في بداية الكلمة:

```
if list[i] = " "
   if count > 1
      part += list[i]
   end
else
   part += list[i]
   count++
end
```

ولا تنس أن تزيد قيمة العداد i بواحد للتحرك إلى خانة تالية في المصفوفة List

استعدادا لتنفيذ اللفة التالية من الحلقة While:

i++

ولكن لعلك تتساءل:

لماذا لم نتعامل مع الخانة الفارغة مثل تعاملنا مع المسافة.. من الواضح في الكود السابق أن قيمة العداد count ستزيد بواحد أيضا في حالة التعامل مع خانة فارغة؟.. فلماذا نعامل الخانة الفارغة كأنها حرف؟

السبب في هذا أننا أضفنا خانة فارغة لتقابل حركة مشبعة مجهولة في موضع القافية لم نستطع أن نحسم إن كانت ألفا أو واوا أو ياء، ولهذا يجب أن نعامل الخانة الفارغة كأنها حرف.. كما أننا أضفنا خانة فارغة بعد خانة تحمل همزة المد "آ" وهي كما تعلم تمثل الحرفين "أا"، لكننا أضفنا ١ فقط مقابل همزة المد، لهذا يجب أن نضيف ١ آخر مقابل الخانة الفارغة لنضبط الحسبة.

وبهذا يكون كود هذه الفئة قد تم بحمد الله، ويمكنك الاطلاع عليه كاملا في الملف Romooz.ring واختباره باستخدام الملف Romooz.Tests.ring.

-٧-

التفعيلات والبحور

بحور الشعر العربي:

كتب الشعراء العرب معظم قصائدهم على ١٦ وزنا موسيقيا مختلفا، سماها الخليل بن أحمد (واضع علم العروض) بالبحور الشعرية، وقد اكتشف منها ١٥ بحرا (أشهرها البحر الطويل والبحر الكامل والبحر الوافر)، ثم تداركه تلميذه الأخفش ببحر آخر (لهذا سمي بالبحر المتدارك)، كما أن هناك أوزانا أخرى لم يكتب عليها العرب، أو نادرا ما كتبوا عليها، سماها الخليل بالبحور المهملة (مثل البحر المتوفر، والبحر المتئد، والبحر المستطيل)، وهناك بحور أخرى جلبت من الأمم المجاورة كالفرس والأتراك أو استحدثها بعض الشعراء (مثل بحر الدوبيت، وبحر السلسلة، وبحر التطيلي، وبحر شوقي).

وليسهل على الشعراء التفريق بين البحور، قسم الخليل أوزانها إلى وحدات موسيقية تسمى التفعيلات (مثل: مُسْتَفْعلنْ، فاعِلاتُنْ، مفاعيلُنْ)، حتى يستخدمها الشعراء بدلا من الرموز الصوتية.. فمثلا، تفعيلة البحر الكامل هي مُتَفاعِلنْ (ويمكن أن تتحول في أي موضع إلى مُسْتَفْعلنْ).. يمكن الآن أن نقول "مُتَفاعِلنْ" بدلا من |٠||٠|||٠، و"مُسْتَفْعلنْ" بدلا من |٠|٠||٠ عند وزن الأبيات.. لهذا يمكننا أن نتغنى ببيت عنترة:

هلّا سألتِ القومَ يا ابنةَ مالكٍ = إن كنتِ جاهلةً بما لم تعلمي

على الوزن: مُسْتَفْعلنْ مُسْتَفْعلنْ مُتَفاعِلنْ = مُسْتَفْعلنْ مُتَفاعِلنْ مُسْتَفْعلنْ

ومن ثم يمكننا تقطيع البيت عروضيا إلى:

هلْلا سألْ – تِلْ قو مَ يَبْ – نةَ مالكنْ = إن كنتِ جا – هلتَنْ بما – لم تعلمي

ولهذا نقول إن مُعلّقة عنترة بن شداد تنتمي إلى البحر الكامل، لأن وزنها منتظم على التفعيلة مُتَفاعِلنْ وصيغتها البديلة مُسْتَفْعلنْ.

والجدول التالي يلخص أسماء البحور وتفعيلاتها، مرتبة بنفس الترتيب الذي سنستخدمه في البرنامج، حيث إن أول ١٦ بحرا هي البحور العربية الشهيرة، تليها البحور المهملة والمستحدثة.

م	اسم البحر	تفعيلاته	عدد التفعيلات الدورية
١	الرَّجَز	مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ	١
٢	الكامِل	مُتَفاعِلنْ مُتَفاعِلنْ مُتَفاعِلنْ	١
٣	الهَزَج	مَفاعيلُنْ مَفاعيلُنْ مَفاعيلُنْ	١
٤	الوافر	مُفاعِلَتُنْ مُفاعِلَتُنْ مُفاعِلَتُنْ	١
٥	الرَّمَل	فاعِلاتُنْ فاعِلاتُنْ فاعِلاتُنْ	١
٦	المُتَقارِب	فَعولُنْ فَعولُنْ فَعولُنْ فَعولُنْ	١
٧	المُتَدارَك	فاعِلُنْ فاعِلُنْ فاعِلُنْ فاعِلُنْ	١
٨	المُنْسَرِح	مُسْتَفْعِلُنْ مَفْعُولَاتُ مُسْتَفْعِلُنْ	٢
٩	البسيط	مُسْتَفْعِلُنْ فاعِلُنْ مُسْتَفْعِلُنْ فَعِلُنْ	٢
١٠	المُضارِع	مَفاعيلُنْ فاعلاتُنْ مفاعيلُنْ	٢
١١	الخفيف	فاعلاتُنْ مُسْتَفْعِلُنْ فاعلاتُنْ	٢
١٢	المديد	فاعلاتُنْ فاعِلُنْ فاعلاتُنْ	٢
١٣	الطَّوِيل	فَعولُنْ مَفاعيلُنْ فَعولُنْ مُفاعِلُنْ	٣
١٤	المُقْتَضَب	مَفْعُولاتُ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ	٣
١٥	السَّرِيع	مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مَفْعُولاتُ	٣
١٦	المُجْتَثّ	مُسْتَفْعِلُنْ فاعلاتُنْ فاعلاتُنْ	٣
١٧	شَوْقِي	مَفْعُولُ مَفْعُولُ مَفْعُولُ	١
١٨	الدُّوبِيت	مَفْعُولاتُ مَفْعُولاتُ مَفْعُولاتُ	١
١٩	المُتَوَفِّر	فَاعِلاتُكَ فَاعِلاتُكَ فَاعِلاتُكَ	١

م	اسم البحر	تفعيلاته	عدد التفعيلات الدورية
٢٠	السَّلْسِلَة	مَفْعُولاتُ مُسْتَفْعِلُنْ مَفْعُولاتُ	٢
٢١	المُسْتَطِيل	مَفَاعِيلُنْ فَعُولُنْ مَفَاعِيلُنْ	٢
٢٢	المُمْتَدّ	فاعِلُنْ فاعلاتُنْ فاعِلُنْ فاعلاتُنْ	٢
٢٣	البَارُودِي	فاعِلُنْ فَعولُنْ فاعِلُنْ	٢
٢٤	التُّطَيِّلِيّ	مُسْتَفْعِلُنْ مَفْعُولاتُ مَفْعُولاتُ	٣
٢٥	مُخَلَّعُ البَسِيط	مُسْتَفْعِلُنْ فاعِلُنْ فَعولُنْ	٣
٢٦	المُنْسَرِد	مَفَاعِيلُنْ مَفَاعِيلُنْ فاعِلاتُنْ	٣
٢٧	المُطَّرِد	فاعِلاتُنْ مَفَاعِيلُنْ مَفَاعِيلُنْ	٣
٢٨	المُتَّئِد	فاعِلاتُنْ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ	٣
٢٩	الخَبَب	فَعِلُكَ فَعِلُكَ فَعِلُكَ فَعِلُكَ	١

البحور البسيطة والمركبة:

من الجدول السابق، تلاحظ وجود بحور بسيطة تتكون من تفعيلة واحدة تتكرر ثلاث مرات أو أربع مثل البحر الكامل (متفاعلن متفاعلن متفاعلن).

لكنْ هناك أيضا بحور مركّبة تتكون من أكثر من تفعيلة، ومنها بحور ثنائية وبحور ثلاثية.. في الواقع كلا النوعين يتكون من تفعيلتين مختلفتين، لكن في البحور الثلاثية تأتي على النمط ١-٢-٢ مثل البحر المتئد (فاعِلاتُنْ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ) أو النمط ١-١-٢ مثل البحر المنسرد (مَفَاعِيلُنْ مَفَاعِيلُنْ فاعِلاتُنْ).. لاحظ أن البحر المسمى بـ "مُخَلَّعُ البَسِيط" هو مجرد صيغة من البحر البسيط، ورغم أنه يبدو مكونا من ٣ تفعيلات مختلفة (مُسْتَفْعِلُنْ فاعِلُنْ فَعولُنْ) فهو في الحقيقة مكون من تفعيلتين فقط، لأن فعولن هي في الواقع "مُتَفْعِلْ" وهي تحوير للتفعيلة مُسْتَفْعِلُنْ بدخول زحاف وعلة عليها.. لكن رغم هذا سنعتبره بحرا ثلاثيا، ليختلف عن "البسيط" الذي هو بحر ثنائي.

وقد أضفت العمود الرابع للجدول السابق المعنون بـ "عدد التفعيلات الدورية"، لأحدد فيه

نوع البحر (أحادي ثنائي ثلاثي).

ولكن، فيم يفيدنا كون البحر أحاديا أو ثنائيا أو ثلاثيا؟

هذا الأمر مهم في شعر التفعيلة.. أنت تعرف أن الشعر العمودي يتكون من أبيات متساوية الطول (أي أن كل بيت يحتوي على نفس العدد من التفعيلات)، وأطول شطرة في الشعر العمودي تتكون من ٤ تفعيلات، أي أن البيت الذي يحتوي على شطرتين يتكون من ٨ تفعيلات، لكن شعر التفعيلة يتكون من سطور مختلفة الطول، فيمكن أن يتكون سطر من تفعيلة واحدة، وآخر من ٣ تفعيلات، وثالث من ١٠ تفعيلات أو أكثر بدون حد أقصى!

وبالنسبة للبحور البسيطة، لا توجد مشكلة في شعر التفعيلة، فلدينا تفعيلة واحدة تتكرر باستمرار في نفس السطر، ولعل هذا هو السبب في أن معظم شعر التفعيلة مكتوب على البحور البسيطة.

لكن بالنسبة للبحور المركبة، يجب أن نتوقع التفعيلة التي ستأتي في كل موضع.. خذ البحر الطويل مثلا (فعولن مَفاعيلن فعولن مَفاعِلُن) مع ملاحظة أن مفاعلن هي صيغة من مفاعيلن.. لو افترضنا كتابة قصيدة تفعيلة على هذا البحر، فسنعامله كبحر ثنائي، تتكرر فيه الوحدتان فعولن مفاعيلن، كما يلي:

فعولن مفاعيلن

فعولن

فعولن مفاعيلن فعولن

فعولن مفاعيلن فعولن مفاعيلن فعولن مفاعلن

فعولن مفاعيلن فعولن مفاعيلن فعولن

فعولن مفاعلن

نفس الأمر في البحور الثلاثية، لكن التكرار الدوري في نفس السطر سيكون بعد كل ثلاث تفعيلات.

الزحافات والعلل:

لكل بحر عدة صيغ موسيقية، تأتي بسبب ثلاثة اختلافات، ودعنا نأخذ البحر الكامل كمثال للتوضيح:

١- عدد التفعيلات في كل شطرة:

يمكن أن يكون البحر تاما (تتكون شطرته من ٣ تفعيلات: مُتَفاعِلنْ مُتَفاعِلنْ مُتَفاعِلنْ)، أو مجزوءا (تتكون شطرته من تفعيلتين: مُتَفاعِلنْ مُتَفاعِلنْ)، أو منهوكا (تتكون شطرته من تفعيلة واحدة: مُتَفاعِلنْ).

٢- دخول بعض الزحافات على تفعيلات البحر:

والزِّحَاف هو تغير يحدث في موضع من التفعيلة سواء بتسكين حرف متحرك (مثل تسكين الحرف الثاني من مُتَفاعِلنْ الذي يحولها إلى مُتْفاعِلنْ والتي تكافئ التفعيلة مسْتفْعلن) أو بحذف حرف من التفعيلة (مثل حذف الحرف الثاني من مُتَفاعِلنْ الذي يحولها إلى مُفاعِلنْ والتي هي التفعيلة مُتفْعلنْ).. ولكل بحر أنواع مقبولة موسيقيا من الزحافات (مثل تحول متفاعلن إلى مستفعلن في البحر الكامل)، وأنواع مستقبحة ينصح بالابتعاد عنها (مثل تحول متفاعلن إلى متفعلن في البحر الكامل)، وأنواع مرفوضة تعتبر كسرا في الوزن، فلا يمكن مثلا تحويل متفاعلن إلى مُسْتَعِلُن في البحر الكامل، لكن من المقبول أن تتحول مستفعلن إلى مستعلن في بحر الرجز.. ولكن أخوض في هذا الكتاب في شرح زحافات كل بحر حتى لا نتوه في تفاصيل عروضية كثيرة، لكننا سنراعي هذه الزحافات في البرنامج وسأوضح لك كيف سنفعل هذا بعد قليل.

٣- دخول عِلّة على التفعيلة الأخيرة في الشطرة:

والعِلّة هي تغير يلحق بنهاية التفعيلة بزيادة حرف ساكن (كأن تتحول متفاعلنْ إلى متفاعلانْ) أو حرفين (كأن تتحول متفاعلنْ إلى "متفاعلنْ فع" والتي تكافئ "متفاعلاتنْ")، كما يمكن أن يتم حذف حرف أو أكثر من التفعيلة مع تسكين الحرف الأخير لو كان متحركا (كحذف حرف من متفاعلنْ لتتحول إلى متفاعلْ

التي تكافئ التفعيلة فَعِلاتُنْ، وحذف ثلاثة حروف من متفاعلن لتتحول إلى متفا التي تكافئ التفعيلة فَعِلُنْ).

وسنكتب برنامج الشاعر بصيغة عامة ليراعي كل هذه الاختلافات، فنظرا لأنه مكتوب ليقبل الشعر التفعيلة فلن تحدث مشاكل مع أطوال البيت العمودي المختلفة، كما أن البرنامج سيقبل الزحافات والعلل الممكنة لكل بحر في أي طول له.

هنا، يجب أن نتعرف أولا على كل التفعيلات التي سنتعامل معها في البرنامج، قبل أن نبدأ في تطبيق تلك القواعد عليها.

التفعيلات:

الجدول التالي يلخص التفعيلات التي سنتعامل معها في البرنامج، مع ملاحظة أن التفعيلات من ١ إلى ٢٤ هي التفعيلات الأساسية وزحافاتها، وهي تفعيلات يمكن أن تأتي في أي موضع من الشطرة، بينما التفعيلات الباقية (من ٢٥ إلى النهاية) هي تفعيلات العلة، وتأتي فقط كآخر تفعيلة في الشطرة:

م	التفعيلة	رموزها
١	فَعُولُ	|٠||
٢	فَاعِلُ	||٠|
٣	مَفْعُولُ	|٠|٠|
٤	مُفَاعِلُ	||٠||
٥	فَعِلاتُ	|٠|||
٦	مَفَاعِيلُ	|٠|٠||
٧	مُتَفَاعِلُ	||٠|||
٨	فَاعِلاتُ	|٠||٠|
٩	مَفْعُولاتُ	|٠|٠|٠|
١٠	فَاعِلاتُكَ	||٠||٠|

م	التفعيلة	رموزها
١١	مُسْتَفْعِلُنْ	٠||٠|٠|
١٢	مُتَفَاعِلُنْ	٠||٠|||
١٣	مَفَاعِيلُنْ	٠|٠|٠||
١٤	مُفَاعِلَتُنْ	٠||٠|||
١٥	فَاعِلاتُنْ	٠|٠||٠|
١٦	فَعِلاتُنْ	٠|٠|||
١٧	مُتَفْعِلُنْ	٠||٠||
١٨	مُفْتَعِلُنْ	٠|||٠|
١٩	فَعِلُكَ	||||
٢٠	فَعُولُنْ	٠|٠||

م	التفعيلة	رموزها
٢١	فَاعِلُنْ	٠||٠|
٢٢	مُتَعِلُنْ	٠||||
٢٣	فَعْلُنْ	٠|٠|
٢٤	فَعِلُنْ	٠|||
٢٥	مُسْتَفْعِلانْ	٠٠||٠|٠|
٢٦	مُتَفَاعِلانْ	٠٠||٠|||
٢٧	مَفَاعِيلانْ	٠٠|٠|٠||
٢٨	مُفَاعِلَتَانْ	٠٠|||٠||
٢٩	فَاعِلاتَانْ	٠٠|٠||٠|
٣٠	فَعِلاتَانْ	٠٠|٠|||
٣١	مُتَفْعِلانْ	٠٠||٠||
٣٢	مُفْتَعِلانْ	٠٠|||٠|
٣٣	مَفْعُولانْ	٠٠|٠|٠|
٣٤	فَعُولانْ	٠٠|٠||

م	التفعيلة	رموزها
٣٥	فَاعِلانْ	٠٠||٠|
٣٦	مُتَعِلانْ	٠٠||||
٣٧	فَعْلانْ	٠٠|٠|
٣٨	فَعِلانْ	٠٠|||
٣٩	فَعُولْ	٠٠||
٤٠	فَعْلْ	٠٠|
٤١	مُتَفَاعِلاتُنْ	٠|٠||٠|||
٤٢	فَاعِلاتَاتُنْ	٠|٠|٠||٠|
٤٣	مَفْعُولاتُنْ	٠|٠|٠|٠|
٤٤	فَاعِلاتُكُمْ	٠||٠||٠|
٤٥	فَعَلْ	٠||
٤٦	فَعْ	٠|
٤٧	مَفْعُولُنْ	٠|٠|٠|
٤٨	مَفْعُولاتانْ	٠٠|٠|٠||٠|

معلومات البحور:

نريد الآن صياغة المعلومات العروضية الخاصة بالبحور بكود رينج.. أسهل طريقة هي وضع البيانات الخاصة بكل بحر في مصفوفة، ووضع كل مصفوفات البحور كعناصر في مصفوفة أكبر اسمها BohorInfo.. سأريك هنا جزءًا من هذه المصفوفة، ويمكنك أن تراها كاملة في الملف BohorInfo.ring في المجلد AroodInfo في مجلد برنامج الشاعر:

```
BohorInfo = [
   #1
         [
           :OrderID = 1,
           :Name = "الرَّجَز",
           :Tafs = [11],
           :PreferedTafCount = 3
         ],
   #2
         [
           :OrderID = 2,
           :Name = "الكامِل",
           :Tafs = [12],
           :PreferedTafCount = 3
         ],
   #3
         //…….
]
```

لاحظ أهمية التنسيق في إظهار البيانات بصورة واضحة ومنظمة، حيث:

- استخدمنا مفاتيح لتسمية العناصر الموجودة في مصفوفة كل بحر، ليكون معناها واضحا، وليكون الكود الذي سيستخدمها لاحقا أسهل في كتابته وقراءته.. هذه من المميزات الجميلة في رينج، التي تجعل استخدام المصفوفات بديلا سهلا لتعريف الفئات التي لا تحتوي إلا على بعض الخصائص.
- وضعنا كل عنصر في المصفوفة في سطر مستقل.. هذا أيضا يسهل قراءة البيانات، لكنّ له هدفا أهم هنا، فكتابة اسم البحر باللغة العربية في سطر واحد مع كلمات إنجليزية أخرى سيربك محرر رينج ويجعله يعرض البيانات بترتيب مخلوط عجيب.. لهذا كقاعدة عامة، إذا أردت أن تكتب حروفا عربية في رينج، فيجب ألا تضعها في بداية السطر بل ضع قبلها بعض الحروف اللاتينية (مثل Name = : في حالتنا هذه)، ويمكنك وضع حروف لاتينية مثل الفاصلة بعد الحروف العربية، لكن لا تضع بعد الفاصلة حروفا عربية ولا أرقاما وإلا فسيختل ترتيب العرض!

- سيكون رقم كل بحر، هو ترتيب إضافته إلي المصفوفة BohorInfo، ولتسهيل معرفة رقم كل بحر بمجرد النظر، أضفته كتعليق في سطر مستقل قبل كل بحر (لأن وضعه في بداية أي سطر سيفسد طريقة عرضه بسبب الحروف العربية).. تذكر أن # هي علامة التعليق في رينج مثلها مثل //، وقد فضّلت هنا الرمز # لأن من الشائع استخدامه مع الأرقام.

وتحتوي مصفوفة كل بحر على البيانات التالية:

- OrderID: رقم يمثل ترتيب البحر عند عرض نتائج التحليل العروضي، لأنك ستكتشف بعد قليل أن بعض الشطرات يمكن تقطيعها على أكثر من بحر، وأحيانا يمكن اعتبار القصيدة كلها من بحرين مختلفين (يحدث هذا مع الرجز والكامل، ومع الهزج والوافر).. لاحظ أن OrderID لا يساوي رقم البحر، فالبحر المنسرح مثلا هو رقم ٨ في المصفوفة، لكن OrderID الخاص به ١٤.. في الحقيقة كان من الأفضل لو رتبنا البحور في المصفوفة بأولوية العرض بحيث لا نحتاج للعنصر OrderID، لكني صممت هذه البيانات في الإصدار الأول من الشاعر عام ١٩٩٨ ولم أكن أهتم حينها بترتيب عرض البحور، وحينما أردت فعل هذا بعد عدة سنوات، أضفت الحقل OrderID لكل بحر، لأن تغيير أرقام البحور صار أمرا معقدا بعد أن استخدمتها في تعريف التفعيلات والزحافات والعلل!.. الدرس المستفاد من هذا هو أن استخدام الثوابت Constants أفضل من استخدام الأرقام مباشرة.. فلو قمنا مثلا بتعريف ثابت اسمه $Ragaz ووضعنا فيه الرقم ١، واستخدمنا الثابت $Ragaz في كل علاقة تخص بحر الرجز بدلا من الرقم ١، فسيسهل علينا لاحقا تغيير رقم بحر الرجز (بتغيير ترتيبه في المصفوفة BohorInfo) وكل المطلوب حينها ليعمل البرنامج بشكل صحيح هو تغيير قيمة الثابت $Ragaz ليحمل الرقم الجديد بدون العبث بباقي الكود. سأترك لك تجربة هذا بنفسك لو أردت.. لكن على كل حال، استخدام خاصية مستقلة للترتيب هي خبرة جيدة أحببت أن ألفت نظرك لها، لأنك حتى لو كتبت البحور

بالترتيب الذي تفضله، فقد تقرر بعد فترة تغيير ترتيب عرضها، والحل المثالي لهذا سيكون باستخدام الحقل OrderID.. وسيفيدك هذا مثلا لو أردت إضافة بحور جديدة مستقبلا كالبحر اللاحق مثلا (مستفعلاتن مستفعلاتن مستفعلاتن) وغيره من البحور المستحدثة التي لم أضِفها للبرنامج.

- Name: اسم البحر.
- Tafs: مصفوفة تحتوي على أرقام تفعيلات البحر.. هذه المصفوفة ستحتوي على تفعيلة واحدة في البحور الأحادية، وتفعيلتين في البحور الثنائية، وثلاث تفعيلات في البحور الثلاثية.. فمثلا، بحر الرجز بحر أحادي يتكون من تكرار التفعيلة مستفعلن (وهي التفعيلة رقم ١١ في الجدول السابق)، لهذا وضعت في المصفوفة Tafs الرقم ١١.. الحقيقة أنني لن أستخدم المصفوفة Tafs في هذا الكتاب، لكنك تستطيع استخدامها لو أردت لتطبيق الزحافات والعلل على أسماء التفعيلات الأصلية للبحر، لتعرض للمستخدم مثلا مُتْفاعلنْ بدلا من مُسْتفعلنْ في البحر الكامل عندما يدخل الزحاف على التفعيلة مُتَفاعلن فيجعل الحرف الثاني فيها ساكنا.. لكننا هنا سنتجاهل تفاصيل الزحافات والعلل للتبسيط.
- PreferedTafCount: عدد التفعيلات المفضل للبحر، وهو عدد تفعيلات الشطرة على الصيغة التامة من البحر، فمثلا شطرة الرجز تتكون من ٣ تفعيلات (مستفعلن مستفعلن مستفعلن).. لن نستخدم هذه المعلومة في هذا الكتاب أيضا، لكنك تستطيع الاستفادة بها لعرض التفعيلة "متفاعلاتن" بدلا من "متفاعلن فع".

معلومات التفعيلات وخوارزمية التقطيع العروضي:

نحتاج الآن إلى كتابة معلومات التفعيلات TafsInfo بكود رينج، وهي مصفوفة تحتوي على بيانات كل تفعيلة، بنفس الطريقة التي اتبعناها مع معلومات البحور.

قد تظن أننا سنكتب اسم التفعيلة ورموزها فقط، لكن الأمر أعمق من هذا، لأن بيانات كل تفعيلة هي التي ستساعدنا في تحليل الشطرة وتقطيعها على بحر أو أكثر.

دعنا أولا نفهم الخوارزمية التي سنتبعها لتقطيع الشطرة:

مبدئيا ستكون لدينا الرموز الصوتية التي تمثل الشطرة التي نريد تحليلها عروضيا..
ولنأخذ مثلا هذه الرموز: ٠||٠|٠|٠||٠|

سنحاول تقطيع هذه الرموز بكل التباديل الممكنة، مع مراعاة هذه القواعد:

- أقصر تفعيلة في أي موضع (ما عدا الموضع الأخير) طولها ٤ رموز.
- أطول تفعيلة في أي موضع (ما عدا الموضع الأخير) طولها ٧ رموز.
- يمكن أن تدخل العلّة على التفعيلة الأخيرة.. لهذا يمكن أن ينحصر طول التفعيلة الأخيرة بين ٢ و ٩.
- لا يمكن أن تبدأ تفعيلة بالرمز ٠ لكن يمكن أن تنتهي بعض التفعيلات بالرمز |.

باستخدام هذه القواعد، يمكننا تقطيع الرموز السابقة إلى الاحتمالات التالية:

التقطيع	التفعيلات المناظرة
٠||٠| – ٠|٠|| – ٠|٠|	فَعْلُنْ فعولنْ فاعلنْ
٠|| – ٠|٠|٠|| – ٠|٠|	فَعْلُنْ مفاعليلنْ فَعَلْ
٠||٠| – ٠|٠| – |٠|٠|	مفْعولُ فَعْلُنْ فاعلنْ
٠|| – ٠|٠|٠| – |٠|٠|	مفْعولُ مفْعولنْ فَعَلْ
٠|| – ٠|٠| – ٠||٠|٠|	مسْتفْعلنْ فَعْلُنْ فَعَلْ
٠| – |٠|٠| – ٠||٠|٠|	مسْتفْعلنْ مفْعولُ فَعْ
٠||٠|٠| – |٠|٠||٠|	مسْتفْعلنْ مسْتفْعلنْ

أعرف أنك مندهش الآن.. فتحليل هذه الرموز القصيرة (١٤ رمزا) أعطانا ٧ تقطيعات مختلفة، فلماذا حدث هذا؟

السبب هو أن جميع هذه التقطيعات تعطي نفس الصوت الموسيقي، ولو قرأتها بصوت مسموع فستكتشف أنها جميعا تنويعات للنغمة (مسْتفْعلنْ مسْتفْعلنْ)، وهي النغمة التي ننتظر أن يخبرنا بها البرنامج ليتسق مع قواعد عروض الخليل بن أحمد.. فكيف نجعله يفعل هذا؟

الحل هو استبعاد التقطيعات التي لا تعطي بحورا.. الطبيعي أن نكتشف البحر من

خلال التفعيلات، لكننا سنكتشف التفعيلات أيضا من خلال البحر!.. فكيف نفعل هذا؟ بتحليل جدول البحور، يمكننا أن نعرف البحور التي تظهر فيها كل تفعيلة وفي أي موضع.. دعنا مثلا نأخذ التفعيلة مستفعلن، وسنلاحظ ما يلي:

- في البحور البسيطة (أحادية التفعيلة) يمكن أن تأتي مستفعلن في أي موضع في بحر الرجز (تفعيلته الأساسية) وكذلك في البحر الكامل (كزحاف للتفعيلة متفاعلن).
- في البحور المركبة ثنائية التفعيلة يمكن أن تظهر مستفعلن في:

أ. المواضع الزوجية (التفعيلة الثانية أو الرابعة أو السادسة... إلخ) في كل من البحر الخفيف (فاعلان مستفعلن فاعلاتن) وبحر السلسلة (مفعولاتُ مستفعلن مفعولاتُ).

ب. المواضع الفردية (التفعيلة الأولى أو الثالثة أو الخامسة... إلخ) في كل من البحر المنسرح (مستفعلن مفعولاتُ مستفعلن) والبحر البسيط (مستفعلن فاعلن مستفعلن فاعلن).

- في البحور المركبة ثلاثية التفعيلة يمكن أن تظهر مستفعلن في:

أ. المواضع الثلاثية (التفعيلة الثالثة أو السادسة أو التاسعة... إلخ) في كل من البحر المقتضب (مفعولاتُ مستفعلن مستفعلن) والبحر المُتَّئِد (فاعلاتن فاعلاتن مستفعلن).

ب. المواضع ما بعد الثلاثية (التفعيلة الأولى أو الرابعة أو السابعة... إلخ) في كل من البحر السَّريع (مستفعلن مستفعلن مفعولاتُ) والبحر المُجْتَثّ (مستفعلن فاعلاتن فاعلاتن) وبحر التُّطَيِّلِيّ (مستفعلن مفعولاتُ مفعولاتُ) وبحر مخلع البسيط (مستفعلن فاعلن فعولن).

ت. المواضع ما قبل الثلاثية (التفعيلة الثانية أو الخامسة أو الثامنة... إلخ) في كل من البحر المقتضب (مفعولاتُ مستفعلن مستفعلن) والبحر السَّريع (مستفعلن مستفعلن مفعولاتُ)

وبالمثل، يمكننا تحليل البحور الممكنة لكل التفعيلات، ومن ثم يمكننا معرفة التقطيعات

المختلفة بالخوارزمية البسيطة التالية:

١- سنحسب البحور الممكنة لكل تفعيلة على حسب موضعها في الشطرة.

٢- سنحسب البحور المشتركة بين كل التفعيلات.. فإن كان التقاطع صفرا (لا توجد بحور مشتركة)، فهذا التقطيع مرفوض، وإن وجدنا بحرا مشتركا (أو أكثر) فسنقبل هذا التقطيع.

ولو طبقنا هذه الخوارزمية على الاحتمالات المذكورة في الجدول السابق، فستستبعد ٥ تقطيعات وتترك لنا تقطيعين مقبولين فقط، هما:

التقطيع	التفعيلات المناظرة	البحور
٠|٠| - ٠|٠|٠|| - ٠||	فَعْلُنْ مفاعليلنْ فَعَلْ	البحر الطويل
٠||٠|٠| - ٠||٠|٠|	مسْتفْعلنْ مسْتفْعلنْ	بحر الرجز، البحر الكامل

لاحظ أن سبب ظهور البحر الطويل، هو دخول زحاف اسمه الخَرْم على التفعيلة الأولى (ولا يدخل هذا الزحاف إلا على التفعيلة الأولى ويكون بحذف أول حرف منها) وهذا أدى إلى تحويل فعولن إلى عولن التي تكافئ فَعْلنْ، كما أن التفعيلة الأخيرة فَعَلْ ناتجة عن علّة حذفت حرفين من فعولن فحولتها إلى فعو (= فعل).. لهذا سيعتبر البرنامج أن عولن مفاعلين فعو هي صيغة مجزوءة من صيغ البحر الطويل (فعولن مفاعيلن فعولن مفاعيلن)!.. وقبل أن يعترض اللغويون منكم، أذكرهم أن ما دفعني لهذا التعميم هو السماح بالتعامل مع شعر التفعيلة، الذي تطول فيه السطور وتقصر كما يريد الشاعر فيأتي أحيانا بصيغ غير متوقعة في الشعر العمودي.

وليس عليك أن تقلق من هذا الاحتمال الزائد، فعند تحليل عدة شطرات من القصيدة من المستبعد أن يظهر فيها جميعا، لهذا سيكون احتمالا مهملا كما سنرى لاحقا.

بقيت ملحوظة أخيرة، تخص العلل:

كما قلنا، تدخل العلة على التفعيلة الأخيرة في الشطرة بحذف أو إضافة حرف أو أكثر، وقد أضفنا تفعيلات العلة في نهاية جدول التفعيلات، لكن لاحظ أن بعض التفعيلات

العادية قد تصير تفعيلات علة في بعض البحور.. فمثلا، لو حذفت آخر حرف من التفعيلة "مُتَفَاعِلُنْ" وسكنت ما قبله فستصير "مُتَفَاعِلْ" وهي نفسها التفعيلة "فَعِلاتُنْ".. هذا معناه أن التفعيلة "فَعِلاتُنْ" تأتي كزحاف للتفعيلة "قَاعِلاتُنْ" وتأتي كعلة للتفعيلة "مُتَفَاعِلُنْ"، لهذا يجب أن نقبل التفعيلة "فعِلاتن" في نهاية الشطرة في البحر الكامل (مُتَفَاعِلُنْ مُتَفَاعِلُنْ فَعِلاتُنْ).

لفعل هذا، سنضيف حقلا في مصفوفة كل تفعيلة، يخبرنا بالتفعيلات الأصلية التي لو دخلت عليها علة تحولها إلى هذه التفعيلة، ومن ثمّ سنستخدم التفعيلة الأصلية لمعرفة البحور التي تظهر فيها ونأخذها في اعتبارنا مع التفعيلة الحالية.. كل تفعيلات العلة ستحتوي على هذا الحقل (ما عدا التفعيلة "فَعْ" لأنها تضاف كزائدة لكل التفعيلات)، وبعض التفعيلات الأصلية ستحتوي على هذا الحقل أيضا.

لقد فهمنا الخوارزمية، والآن نحن جاهزون لكتابة المصفوفة TafsInfo وفيها كل هذه البيانات.. ستجدها كاملة في الملف TafsInfo.ring في المجلد AroodInfo في مجلد المشروع، وهذا جزء صغير منها للتوضيح:

```
TafsInfo = [
  #1
      [
            "||0|",
            :Name = "فَعُولُ",
            :BahrAllN = [6],
            :BahrN20 = [21, 23],
            :BahrN21 = [13],
            :BahrN30 = [25]
      ],
      // ..............
  #16
      [
            "|||0|0",
            :Name = "فَعِلاتُنْ",
            :BahrAllN = [5, 19],
            :BahrN20 = [10, 22],
```

```
                :BahrN21 = [11, 12],
                :BahrN30 = [16, 26],
                :BahrN31 = [27, 28],
                :BahrN32 = [16, 28],
                :EllaOf = [5, 7, 12, 24]
        ],
    // ……..
    #47
        [
                "|0|0|0",
                :Name = "مَفْعُولُنْ",
                :kharmOf = 13,
                :EllaOf = [3, 9, 11, 12, 15, 21]
        ],
    #48
        [
                "|0|0|0|00",
                :Name = "مَفْعُولاتانْ",
                :EllaOf = [9]
        ]
]
```

لقد استخدمنا نفس التنسيق لكتابة المصفوفة بشكل منظم وواضح، حيث تحتوي مصفوفة كل تفعيلة على البيانات التالية:

- رموز التفعيلة: لم أستخدم مفتاحا لهذا المعلومة، لأختصر صيغة البحث في المصفوفة.. هذا يريك أنك تستطيع استخدام المفاتيح مع بعض القيم، وعدم استخدامها مع أخرى.. لغة رينج مرنة جدا في هذا الأمر.
- Name: يحتوي هذا الحقل على اسم التفعيلة.
- BahrAllN: يحتوي هذا الحقل على مصفوفة، فيها أرقام البحور البسيطة التي يمكن أن تظهر فيها التفعيلة في كل موضع.. أستخدم هنا الحرف N كاختصار للكلمة الإنجليزية Number.. من الشائع في الرياضيات والبرمجة أن نرمز للمواضع بالحرف N.
- BahrN20: يحتوي هذا الحقل على مصفوفة، فيها أرقام البحور المركبة التي يمكن

أن تظهر فيها التفعيلة في المواضع الزوجية، وهي المواضع التي يكون باقي قسمتها على ٢ صفرا.. هذا يشرح لك معنى المقطع N20، فالحرف N كما قلنا هو موضع التفعيلة في الشطرة، والرقم ٢ هو الرقم الذي سنقسم عليه، والرقم صفر هو باقي القسمة.. حينما نكتب كود رينج، سيتحول هذا إلى التعبير:

If N % 2 = 0

- BahrN21: مصفوفة نحمل البحور التي تأتي فيها التفعيلة في المواضع الفردية.
- BahrN30 و BahrN31 و BahrN32:

هذه المصفوفات تحمل البحور الثلاثية التي يمكن أن تظهر فيها التفعيلة في المواضع المختلفة (حيث باقي القسمة على ٣ = صفر أو ١ أو ٢ بالترتيب).

- EllaOf: يحتوي هذا الحقل على مصفوفة فيها التفعيلات التي لو دخلت عليها علة تحولها إلى التفعيلة الحالية.

لاحظ أنك غير مجبر على كتابة كل هذه الحقول في كل التفعيلات، فبعض التفعيلات لا تظهر إلا في البحور البسيطة، وبعض التفعيلات لا تظهر في البحور الثلاثية، وبعض التفعيلات لا يمكن أن تكون علة.. ومن التسهيلات التي تتيحها لك رينج، السماح بإزالة الحقول الفارغة، فلو بحثت في المصفوفة عن قيمة في حقل غير موجود فستعيد رينج القيمة ٠ ولن تعطيك خطأ، وهذا سمح باختصار تعريف التفعيلات، بدلا من إضافة كل الحقول في كل التفعيلات وترك بعضها فارغا.

تضارب المتغيرات العامة:

نريد الآن استخدام معلومات التفعيلات ومعلومات البحور في تقطيع شطرة.. سنضيف للبرنامج ملفا اسمه Taqtee3.ring، وسنحمّل في بدايته ملفي معلومات البحور والتفعيلات:

```
load "AroodInfo\TafsInfo.ring"
load "AroodInfo\BohorInfo.ring"
```

وسنضيف إليه دالة تستقبل رموز الشطرة، وتعيد إلينا مصفوفة تحتوي على معلومات التقطيع العروضي الخاصة بها:

```
Func GetTaqtee3(strHarakaat)
   t = new Taqtee3Class(strHarakaat)
   return t.Taqtee3
EndFunc
```

واضح أن كل ما فعلناه في هذه الدالة هو تعريف نسخة من فئة التقطيع Taqtee3Class مع إرسال رموز الشطرة لمنشئ الفئة، ثم جعلنا الدالة تعيد قيمة الخاصية Taqtee3 المعرّفة في الفئة Taqtee3Class.

إذن، فكل العمل سيتم داخل الفئة Taqtee3Class.. لكن قبل أن نكتب كود هذه الفئة، علينا أن نفهم أولا، لماذا لم نكتب الكود كله مباشرة داخل الدالة GetTaqtee3 دون الحاجة لإنشاء فئة لن نستخدمها في البرنامج إلا مرة واحدة؟

في الحقيقة كلا الأمرين ممكن، لكن الكود الذي سنكتبه سيحتاج لاستدعاء عدة دوال ولو كتبناها في الملف مباشرة (في الجزء العام Global قبل تعريف أي فئة)، فستصير عامة لكل المشروع ويمكن استدعاؤها من أي ملف آخر يقوم بتحميل الملف Taqtee3.ring.. لهذا لو أردت إخفاء هذه الدوال، فيجب وضعها في المقطع Private داخل فئة.

ربما لا يكون هذا سببا كافيا لإنشاء الفئة، فلا شيء يجبرنا على استدعاء هذه الدوال حتى لو كانت عامة.. فما هو إذن السبب الأهم الذي دفعنا لإنشاء الفئة؟

كما سترى بعد قليل، تعتمد طريقة عمل هذه الدوال على وجود ٤ متغيرات عامة (معرّفة خارج الدوال حتى يمكن استخدامها في كود أي دالة منها).. لو وضعنا هذه المتغيرات في الجزء العام من الملف، فستكون هي أيضا عامة ومرئية للملفات الأخرى في المشروع.. وهنا يمكن أن تحدث مشكلة كبيرة، فالمتغيرات العامة في أي ملف تصير مرئية في الملفات التي تقوم بتحميل هذا الملف باستخدام الأمر Load، لكن الأغرب من هذا أنها تصير مرئية أيضا في الملفات التي قام هذا الملف بتحميلها!

دعني أوضح الأمر.. خذ الملف aString.ring كمثال.. عندما تضيف الجملة:

```
Load "aString.ring"
```

في أي ملف (مثل الملف Taqtee3.ring)، فسيحدث ما يلي:

- كل المتغيرات العامة في الملف aString.ring ستصير مرئية من داخل الملف Taqtee3.ring وهذا أمر طبيعي ومتوقع.
- كل المتغيرات العامة في الملف Taqtee3.ring ستصير مرئية من داخل الملف aString.ring، وهو أمر غير متوقع!

يمكنك أن تلخص الأمر ببساطة، في أن رينج تمزج المتغيرات العامة للملفين في نطاق عام واحد، وهذا أمر له مزاياه، فلو عدت بذاكرتك إلى فصل الكتابة العروضية، فستذكر أننا كنا نرسل اسم دالة النظر للأمام أو الخلف إلى محرك التعبيرات النمطية، وكان محرك التعبيرات النمطية ينجح في استدعاء هذه الدالة رغم أنها معرّفة في ملف آخر لا يعلم عنه شيئا!.. السبب أن تحميل ملف التعبيرات النمطية داخل ملف الكتابة العروضية، جعل كل دوال الكتابة العروضية مرئية من داخل ملف التعبيرات النمطية!

ولكن رغم هذه الفائدة الهامة التي تقلل تعقيد كتابة الكود أحيانا، يظل لهذا الأمر عيب خطير، هو إمكانية حدوث تضارب بين أسماء المتغيرات العامة في الملفين تؤدي لأخطاء توقف تنفيذ البرنامج.. لهذا عليك الابتعاد عن تعريف متغيرات عامة بأسماء قصيرة ومتداولة مثل x و n و name، لأن تكرار تعريفها في أكثر من ملف سيسبب تضاربات وأخطاء.. وعليك أيضا ألا تعرّف متغيرا من الفئة في الجزء العام من الملف، ثم تعيد استخدام نفس اسم المتغير داخل الفئة نفسها، لأن هذا سيدمر الفئة أثناء تنفيذ كودها.. فمثلا لو لديك فئة اسمها myClass تعريفها كالتالي:

```
Class MyClass
  Value

  Func Init(n)
    value = n
  EndFunc

  Func GetNext( )
    s = new MyClass(value + 1)
    return s
  EndFunc
EndClass
```

ثم قررت استخدام هذه الفئة في ملف اسمه TestMyClass.ring كالتالي:

```
Load "MyClass.ring"
s = new MyClass(1)
x = s. GetNext( )
? x.Value
```

فسيحدث هذا الخطأ في البرنامج عند تنفيذ الكود:

```
Line 8 Error (R31) : Trying to destroy the object using the self
reference In method GetNext() in file F:\ring\AlSha3er\Notes\MyClass.ring
called from line 3 in file F:/ring/AlSha3er/Notes/TestMyClass.ring
```

والسبب أن رينج لن تعتبر المتغير s الذي تستخدمه في الدالة GetNext متغيرا جديدا، وإنما ستعتبره المتغير s المعرّف في المنطقة العامة في ملف آخر لم تكن تدري عنه شيئا وأنت تكتب الفئة، وعند وضع قيمة جديدة في هذا المتغير، ستكتشف رينج أنه يحمل نسخة الكائن الذي نتعامل معه حاليا من الفئة myClass، وتغيير قيمته سيدمر هذا الكائن ويحذفه من الذاكرة، وبالتالي يصير الكود الذي ننفذه بلا معنى، ولهذا تضطر رينج لمنع هذا بعرض رسالة خطأ وإيقاف البرنامج!

وتقدم لك رينج حلا سهلا لمنع هذا الخطأ، باستخدام الأمر Load package بدلا من الأمر Load.. عدل أول سطر في الكود السابق ليصير:

```
Load package "myClass.ring"
```

نفذ الكود.. لن يحدث خطأ هذه المرة، فالأمر Load package يمنع الملف MyClass.ring من رؤية المتغيرات العامة في الملف TestMyClass.ring.

حل آخر، هو ألا تضيف أي متغيرات عامة في ملفاتك بدون داع.. فما دام المتغير ليس عاما فعلا، وستستخدمه على مستوى الملف فقط، فالأفضل أن تعزله داخل دالة، أو داخل فئة لو كنت تحتاجه على مستوى عدة دوال.. هذا الكود لن يسبب أي خطأ:

```
Load "myClass.ring"
Test( )

Func Test( )
  s = new MyClass(1)
  x = s. GetNext( )
  ? x.Value
EndFunc
```

السبب في هذا أن المتغير s هنا هو متغير موضعي معرف داخل الدالة Test، وبالتالي تستحيل رؤيته من أي مكان خارج هذه الدالة.

لهذا، عليك دائما الابتعاد عن تعريف المتغيرات في المنطقة العامة في ملفات رينج إلا إذا كنت تريد فعلا رؤيتها في ملفات أخرى، وبدلا من هذا ضعها داخل دالة، أو عرّفها كخصائص على مستوى فئة لتكون متاحة لأكثر من دالة.. وإن أردت استخدام ملف كود لا يلتزم بهذه القواعد وتسبب أخطاء عند تنفيذه، فاستخدم الأمر Load package لتدراك الموقف وحل المشكلة ☺

فئة التقطيع العروضي Taqtee3Class:

تحتوي هذه الفئة على عنصرين عامين فقط، هما:

- الخاصية Taqtee3، وهي مصفوفة تحتوي على نتائج تقطيع الشطرة، وقد وضعنا فيها مصفوفة فارغة [ ] كقيمة ابتدائية.
- دالة إنشاء الفئة Init، ولها معامل واحد يستقبل النص الذي يحتوي على رموز الشطرة، وفي كود هذه الدالة سننسخ النص من هذا المعامل إلى خاصية اسمها harakaat، ونضع طوله في خاصية اسمها hL (وكلا الخاصيتين سنعرّفها في الجزء الخاص private من الفئة)، ثم سنستدعي دالة اسمها GetTafs وسنرسل إلى معاملها القيمة ١، لتبدأ البحث عن التفعيلات من أول رمز في الشطرة:

```
Class Taqtee3Class
   Taqtee3 = [ ]

   Func init(strHarakaat)
      harakaat = strHarakaat
      hL = len(harakaat)
      GetTafs(1)
   EndFunc

   Private

EndClass
```

فلننظر الآن للجزء الخاص من الفئة:

في هذا الجزء سنعرّف أربع خصائص، ذكرنا اثنتين منها وهما harakaat و hL، ونزيد عليهما مصفوفة سنضع فيها أرقام التفعيلات التي نعثر عليها وسنسميها Tafs، ومصفوفة أخرى سنضع فيها نتيجة التقاطع بين البحور المحتملة للتفعيلات وسنسميها propBohor، حيث prob هو اختصار Probable بمعنى "محتمل"، و Bohor هي الكتابة الإنجليزية للكلمة بحور:

```
private
   harakaat
   hL
   Tafs = [ ]
   propBohor = [ ]
```

بعد هذا سنكتب كود الدالة GetTafs، وهي تحتاج إلى تعريف ثلاث دوال أخرى لأداء عملها، سنتعرف عليها لاحقا.

وللدالة GetTafs معامل واحد يستقبل موضع الرمز الذي سنبدأ منه البحث عن تفعيلة، وليس لهذا الدالة قيمة عائدة، فهي تكتب النتائج مباشرة في المصفوفة Taqtee3، وهذا هو سبب تعريفنا لها كخاصية عامة على مستوى الفئة.

وقبل أن نكتب كود هذه الدالة، علينا أن نتعرف أولا على مفهوم هام جدا في البرمجة، وهو الدوال الارتدادية.

الدوال الارتدادية Recursive Functions:

الدالة الارتدادية (التكرارية)، هي دالة يتم تعريفها بطريقة عادية جدا، لكن المختلف فيها هو أنها تستدعي نفسها، وهي بهذا تعمل كأنها حلقة تكرار Loop، فعند تنفيذ كود الدالة ستمر بسطر من الكود يجعلها تستدعي نفسها، وفي هذا الاستدعاء الجديد ستكرر تنفيذ السطر الذي يجعلها تستدعي نفسها، وهكذا دواليك.. لهذا لو لم تضع شرطا يوقف هذه الحلقة بعد تحقيق الهدف الذي تسعى إليه، فسيحدث خطأ في البرنامج يسمى تجاوز سعة الرصة Stack Overflow.. والرصة Stack هي مساحة

من الذاكرة تخصصها لغة البرمجة لحفظ قيم المتغيرات أثناء تشغيل البرنامج، وسميت بهذا الاسم لأنها تعمل مثل رصة الأطباق، فآخر قيمة تضعها فوق الرصة هي أول قيمة تسحبها منها.

دعنا نأخذ مثالا عمليا شهيرا.. كلنا درس دالة المضروب Factorial في الرياضيات، حيث إن مضروب العدد ن هو حاصل ضرب الأعداد من ن إلى ١، فمثلا:

مضروب ٥ = ٥ × ٤ × ٣ × ٢ × ١ = ١٢٠.

وبملاحظة بسيطة ستكتشف أن:

مضروب ٥ = ٥ × مضروب ٤

وبصيغة عامة: مضروب ن = ن × مضروب ن−١

وهو ما يمكن كتابته في البرمجة باستخدام دالة ارتدادية كالتالي:

```
? Factorial(5)

Func Factorial(n)
    return n * Factorial(n - 1)
EndFunc
```

لو جربت المثال السابق فستحصل على الخطأ:

Error (R4) : Stack Overflow! In function factorial( )

فكما ذكرنا، يجب وضع شرط يوقف تكرار استدعاء الدالة لنفسها.. الشرط المناسب في مثالنا هذا، هو وصولنا إلى صفر، فمضروب صفر يساوي ١ (لأن مضروب ١ = ١ × مضروب صفر، فلا بد أن تكون قيمة مضروب صفر = ١)، وأي أرقام سالبة غير مقبولة في دالة المضروب، لهذا سنطلق خطأ في البرنامج.. هذا هو الكود المعدل:

```
Func Factorial(n)
  If n < 0
    Raise("n can't be negative")
  ElseIf n = 0
    Return 1
  Else
    Return n * Factorial(n - 1)
  End
EndFunc
```

والآن لو جربت هذه الدالة فلن تسبب خطأ تجاوز مساحة الرصة، لأن جمل الشرط تمنع تنفيذ دالة المضروب على أعداد سالبة، وتوقف الارتداد (تكرار الاستدعاء) عند الوصول إلى الرقم ١، وهذا يجعل الدالة تستدعي نفسها فقط للأعداد الأكبر من ١.. هذا ما سيحدث مثلا في المثال الخاص بمضروب الرقم ٥:

١. سنستدعي الدالة بالمعامل ٥.
٢. ستستدعي الدالة نفسها بالمعامل ٤.
٣. ستستدعي الدالة نفسها بالمعامل ٣.
٤. ستستدعي الدالة نفسها بالمعامل ٢.
٥. ستستدعي الدالة نفسها بالمعامل ١.
٦. ستستدعي الدالة نفسها بالمعامل صفر.
٧. ستجعل جملة الشرط الدالة تعيد ١.
٨. بعد انتهاء الدالة في الخطوة السابقة، سنرتد للدالة في الخطوة رقم ٦، التي ستضرب القيمة التي حصلت عليها (وهي ١) في قيمة المعامل n (وهو ١ في تلك الدالة)، وتعيد الناتج.
٩. بعد انتهاء الدالة في الخطوة السابقة، سنرتد للدالة في الخطوة رقم ٥، التي ستضرب القيمة التي حصلت عليها (وهي ١) في قيمة المعامل n (وهو ٢ في تلك الدالة)، وتعيد الناتج.
١٠. بعد انتهاء الدالة في الخطوة السابقة، سنرتد للدالة في الخطوة رقم ٤، التي ستضرب القيمة التي حصلت عليها (وهي ٢) في قيمة المعامل n (وهو ٣ في تلك الدالة)، وتعيد الناتج.
١١. بعد انتهاء الدالة في الخطوة السابقة، سنرتد للدالة في الخطوة رقم ٣، التي ستضرب القيمة التي حصلت عليها (وهي ٦) في قيمة المعامل n (وهو ٤ في تلك الدالة)، وتعيد الناتج.
١٢. بعد انتهاء الدالة في الخطوة السابقة، سنرتد للدالة في الخطوة رقم ٢، التي

ستضرب القيمة التي حصلت عليها (وهي ٢٤) في قيمة المعامل n (وهو ٥ في تلك الدالة)، وتعيد الناتج.

١٣. بعد انتهاء الدالة في الخطوة السابقة، نعود إلى السطر الذي استدعينا فيه الدالة لأول مرة في الخطوة رقم ١، حيث ستطبع رينج الناتج النهائي على الشاشة، وهو ١٢٠.

هذه فكرة عامة عن الدوال الارتدادية، حاول أن تستوعبها جيدا، قبل الانتقال للمقطع التالي، لأننا سنستخدم الدوال الارتدادية للحصول على كل الاحتمالات الممكنة لتقطيع رموز الشطرة إلى تفعيلات.

الدالة GetTafs:

وظيفة هذه الدالة هي الحصول على تفعيلة صحيحة تبدأ من الموضع المرسل إليها كمعامل.. دعنا نأخذ الرموز ٠|٠||٠|٠|٠|٠ كمثال.. في البداية سنرسل الرقم ١ إلى معامل الدالة GetTafs لتبدأ تحليل الرموز من موضع البداية.

كما ذكرنا، فإن طول التفعيلة العادية (ليست تفعيلة علة) ينحصر بين ٤-٧ حروف، لهذا فالاحتمالات الممكنة للتفعيلات في مثالنا هذا ستكون:

- ٠|٠| وهي رموز التفعيلة فَعْلُنْ.
- ٠|٠|٠| وهي رموز التفعيلة مَفْعولُ.
- ٠|٠|| ولا توجد تفعيلة لها هذه الرموز.. في الحقيقة لا نحتاج للبحث في مصفوفة التفعيلات أصلا، لأن الرمز التالي لهذه الرموز هو ٠ وكما قلنا لا توجد تفعيلة تبدأ بحرف ساكن، أي أن هذا المسار مقطوع وهذا الاحتمال مرفوض.
- ٠|٠||٠| وهي رموز التفعيلة مُسْتَفْعِلُنْ.

تبدو الأمور سهلة حتى الآن، فلدينا ثلاث تفعيلات صحيحة يمكن أن تبدأ من أول موضع.. لكن هذا يعني أن لدينا ثلاثة مسارات مختلفة للتقطيع، أو فلنقل: لدينا شجرة لها ثلاثة فروع، وعلينا أن نفحص كل فرع على حدة.. هنا تأتي أهمية الدوال

الارتدادية، فكل ما سنفعله ببساطة بعد الحصول على كل تفعيلة محتملة، هو جعل الدالة GetTafs تستدعي نفسها لتبحث عن التفعيلة التالية بدءًا من الموضع التالي لنهاية التفعيلة الحالية:

- ففي حالة التفعيلة فَعْلُنْ سنستدعي GetTafs(5).
- وفي حالة التفعيلة مَفْعولُ سنستدعي GetTafs(6).
- وفي حالة التفعيلة مُسْتَفْعِلُنْ سنستدعي GetTafs(8).

وفي كل مسار من تلك المسارات، ستبحث الدالة GetTafs عن الشجرة الفرعية لكل تفعيلة.. ويعتبر الفرع مبتورا إذا وصلنا إلى رموز لا تمثل تفعيلة أو يليها الرمز ٠ .. والجدول التالي يلخص لك كل شجرة الاحتمالات لهذا المثال:

<table>
<tr><td rowspan="5">|٠|٠ فَعْلُنْ</td><td>||٠| فعولُ</td><td colspan="2">x</td></tr>
<tr><td rowspan="2">||٠|٠ فعولُنْ</td><td>|٠|| فاعلُ</td><td>x</td></tr>
<tr><td colspan="2">|٠||٠ فاعلُنْ</td></tr>
<tr><td>||٠|٠| مفاعيلُ</td><td colspan="2">x</td></tr>
<tr><td>||٠|٠|٠ مفاعيلُنْ</td><td colspan="2">||٠ فَعَلْ</td></tr>
<tr><td rowspan="5">|٠|٠| مَفْعولُ</td><td rowspan="2">|٠|٠ فَعْلُنْ</td><td>|٠|| فاعلُ</td><td>x</td></tr>
<tr><td colspan="2">|٠||٠ فاعلُنْ</td></tr>
<tr><td>|٠|٠| مفعولُ</td><td colspan="2">x</td></tr>
<tr><td>|٠|٠|٠ مفعولُنْ</td><td colspan="2">||٠ فَعَلْ</td></tr>
<tr><td>|٠|٠|٠| مفعولاتُ</td><td colspan="2">|٠ فعْ</td></tr>
<tr><td>||٠|٠|</td><td colspan="3">x</td></tr>
<tr><td rowspan="4">|٠|٠||٠ مُسْتَفْعِلُنْ</td><td>|٠|٠ فَعْلُنْ</td><td colspan="2">||٠ فَعَلْ</td></tr>
<tr><td>|٠|٠| مَفْعولُ</td><td colspan="2">|٠ فعْ</td></tr>
<tr><td>||٠|٠|</td><td colspan="2">x</td></tr>
<tr><td colspan="3">|٠|٠||٠ مُسْتَفْعِلُنْ</td></tr>
</table>

كما تلاحظ، هناك فروع مبتورة (تنتهي بالعلامة x)، وهناك مسارات تبدأ من أول رمز وتنتهي عند آخر رمز وتعطينا تقطيعا كاملا يتكون من ثلاث أو أربع تفعيلات، وهي:

١. فعلن فعولن فاعلن
٢. فعْلنْ مفاعيلنْ فعلْ
٣. مفْعولُ فعْلن فاعلنْ
٤. مفْعولُ مفْعولنْ فعلْ
٥. مفْعولُ مفْعولاتُ فعْ
٦. مُسْتَفْعلُنْ فعْلن فعلْ
٧. مُسْتَفْعلُنْ مفْعولُ فعْ
٨. مُسْتَفْعلُنْ مُسْتَفْعلُنْ

ولكن، كيف ستحتفظ الدالة GetTafs بهذه الاحتمالات؟

سنستخدم هنا طريقة بسيطة للغاية:

حينما تعثر الدالة GetTafs على تفعيلة، فسنضيفها كخانة جديدة في نهاية المصفوفة Tafs، لكن عند العودة للوراء لتجربة احتمال آخر فسنحذف التفعيلة التي أضفناها ليمكننا تجربة فرع جديد.. فعلى سبيل المثال، بعد فشل المسار "فعلن فعولن فاعلُ" سنحذف فاعلُ من المصفوفة Tafs، لنجرب احتمالا آخر وهو فاعلنْ التي سنضيفها إلى المصفوفة Tafs فتصير "فعلن فعولن فاعلنْ" وهو مسار كامل وصحيح.. ثم سنحذف "فاعلنْ" مرة أخرى، وهنا تنتهي احتمالات هذا الفرع، فنخرج من الدالة GetTafs الحالية لنرتدّ إلى الدالة GetTafs التي استدعتها، والتي ستحذف فعولن من المصفوفة Tafs وتضيف بدلا منها مفاعيلُ لتبدأ في فحص فرع جديد.. وهكذا.

هنا يبرز سؤال هام: كيف سنحتفظ بالنتائج ما دامت المصفوفة Tafs تتغير باستمرار؟

لا تقلق.. فكلما حصلنا على مسار صحيح كامل، سنضيف نسخة من المصفوفة Tafs إلى مصفوفة التقطيع Taqtee3، التي ستحتوي على تفاصيل كاملة عن كل مسار ممكن للتفعيلات والبحر المناظر له.

دعنا إذن نكتب الكود الذي ينفذ هذه الخوارزمية:

```
Func GetTafs(startAt)
   minTafLen = 4
   maxTafLen = hL - startAt + 1
   if maxTafLen > 9
      maxTafLen = 7
   elseif maxTafLen < 4
      minTafLen = 2
   end
   // باقي كود الدالة
EndFunc
```

تستقبل الدالة المعامل startAt الذي يمثل الموضع الذي ستبحث عنده عن التفعيلات المحتملة، وعلينا معرفة أصغر وأكبر طول ممكن للتفعيلة.. في الوضع العادي سيكون الطول محصورا بين ٤ و ٧ رموز، لكن التفعيلة الأخيرة يمكن أن تصاب بالعلة، لهذا يمكن أن ينحصر طولها بين ٢ و ٩.. لهذا سنجري بعض الحسابات لحسم هذا الأمر:

- في البداية سنجل أقصر طول للتفعيلة minTafLen يساوي ٤.
- وسنجعل أطول طول للتفعيلة maxTafLen مساويا لعدد الرموز المتبقية في النص منذ الموضع startAt إلى النهاية، وهو يساوي طول الرموز الذي حفظناه من قبل في المتغير hL مطروحا منه startAt ومضافا إليه ١.
- لو كان maxTafLen أكبر من ٩، فهذا يعني بالتأكيد أن هذه ليست التفعيلة الأخيرة، لأن الرموز التي نتعامل معها تصلح لتقسيمها إلى تفعيلتين، لهذا سنضع في maxTafLen القيمة ٧ لأننا سنتعامل مع تفعيلة عادية بدون علة.
- أما إذا كان maxTafLen أصغر من ٤، فنحن بالتأكيد نتعامل مع تفعيلة علة، وفي هذه الحالة سنعدل قيمة minTafLen لنسمح بمعالجة هذه الحالة، لهذا سنضع فيه نفس قيمة maxTafLen.. هذا معناه أن لدينا احتمالا واحد فقط (أصغر طول = أطول طول).. السبب في هذا أن الاحتمالات الأقل من ٤ هي:
  ١. حرف واحد: ولا توجد تفعيلة بهذا الطول لهذا فهو غير مقبول.
  ٢. حرفان، وهي التفعيلة فَعْ.

٣. ثلاثة حروف وهي التفعيلة فَعَلْ، فلا يمكن تقسيم ثلاثة رموز إلى تفعيلتين، فلو أخذت التفعيلة فعْ فقط فسيتبقى بعدها رمز واحد (لا يصلح كتفعيلة).

فالخلاصة: لو كان المتبقي من الرموز أقل من ٤، فعلينا فحص هذه الرموز كتفعيلة واحدة.

بعد معرفة أصغر وأطول طول ممكنين للتفعيلات المحتملة في هذا الموضع، سنكتب حلقة تكرار لأخذ كل الأطوال المحصورة بين هاتين القيمتين، وستؤدي الدالة GetTafs وظيفتها كاملة داخل حلقة التكرار، حيث سنقتطع جزءا من الرموز يبدأ من الموضع startAt وطوله TafLen، ونتأكد أن الرمز التالي له ليس ٠ (مسار مرفوض)، قبل أن نبحث عن الرموز التي اقتطعناها في مصفوفة التفعيلات، فإن كان ناتج البحث صفرا (لا توجد تفعيلة مناظرة لهذه الرموز)، فسننهي اللفة الحالية من حلقة التكرار لنفحص الاحتمال التالي:

```
For TafLen = minTafLen to maxTafLen
   endAt = startAt + TafLen - 1
   if endAt < hl and harakaat[endAt + 1] = "0" Loop End
   rmz = subStr(harakaat, startAt, TafLen)
   tafID = find(TafsInfo, rmz, 1)
   if tafID = 0 Loop End
   // ……
Next
```

لاحظ أننا أرسلنا معاملا ثالثا للدالة find.. هذا المعامل يستقبل رقم العمود الذي تريد البحث فيه، وهذا مفيد حينما تبحث في مصفوفة متعددة الأعمدة (متعددة الأبعاد).. في حالتنا هذه نريد البحث في أول عمود في المصفوفة، لأننا وضعنا فيه رموز التفعيلة.

دعنا نواصل كتابة كود حلقة التكرار (في موضع التعليق):

سنحتاج إلى متغير منطقي اسمه lastTaf نضع فيه True لو كنا نتعامل مع التفعيلة الأخيرة.. يمكننا معرفة هذا لو كانت نهاية التفعيلة endAt تساوي طول الرموز hl:

```
lastTaf = (endAt = hl)
```

أذكرك مجددا أن الكود السابق هو اختصار للكود التالي:

```
If endAt = hl
   lastTaf = True
Else
   lastTaf = False
End
```

لا تجعل الكود الأول يربكك، فأنت تعلم أن رينج تستخدم الرمز = لإجراء عملية إسناد القيم Assignment وعملية المقارنة Comparison، ومن الممكن أن تستخدمه بالمعنيين في سطر واحد، لأن عملية الإسناد تأتي مرة واحدة فقط في بداية الأمر، وأي = بعد ذلك ستفهمها رينج على أنها عملية مقارنة.

سنحتاج أيضا إلى متغير اسمه tafPos نضع فيه موضع التفعيلة الجديدة في المصفوفة Tafs، وهو يساوي عدد خانات المصفوفة Tafs مضافا إليه واحد (موضع الخانة الجديدة التي سنضيفها بعد قليل):

```
tafPos = len(Tafs) + 1
```

نريد الآن فحص رقم التفعيلة TafID الذي حصلنا عليه من عملية البحث، فلو كنا نتعامل مع تفعيلة ليست في الموضع الأخير فلن تصلح تفعيلات العلة (تذكر أن أرقامها في الجدول أكبر من ٢٤) ولهذا لو كان TafID أكبر من ٢٤ فسننهي اللفة الحالية من حلقة التكرار ونقفز للفة التالية لدراسة احتمال آخر.. لكن هناك استثناء واحد فقط، هو التفعيلة "مفعولن" (رقم ٤٧)، فرغم أنها تفعيلة علة تأتي في الموضع الأخير، فمكن الممكن أن تكون أيضا تفعيلة زحاف في حالة دخول زحاف اسمه الخَرْم على التفعيلة "مفاعيلن" فيحذف أول حرف منها لتصير "فاعيلن" التي تكافئ "مفعولن".. هكذا يمكننا صياغة هذه الشروط:

```
if not lastTaf
   if tafID = 47
      if tafPos > 1 Loop end
   elseif tafID > 24
      Loop
   end
end
```

الآن، سنستدعي الدالة HasBahr لنتأكد من وجود بحر مشترك بين التفعيلة التي

عثرنا عليها، والتفعيلات السابقة الموجود في المصفوفة Tafs.. فإن أعادت هذه الدالة false بمعنى عدم وجود بحر مشترك، فلا داعي لتضييع الوقت في هذا المسار، وعلينا إنهاء اللفة الحالية من حلقة التكرار والانتقال لدراسة الاحتمال التالي:

```
if not HasBahr(tafID, tafPos, lastTaf) Loop End
```

لاحظ أن هذه الخطوة ستختصر علينا الكثير من الوقت، لأنها ستستبعد مبكرا ستة من الاحتمالات الثمانية التي عرضناها في الجدول السابق فهي لا تمثل أي بحر معروف، وستترك لنا فقط الاحتمالين التاليين:

١. "فَعْلُنْ مَفَاعِيلُنْ فَعَلْ"، ومن الممكن أن تكون صيغة من البحر الطويل (عُولن مَفَاعِيلُنْ فَعُو) بدخول الخَرم على "فعولن" لتحويلها إلى "عولن" (= "فعلن").

٢. "مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ" وهي صيغة بحر الرجز، ويمكن أن تأتي في البحر الكامل كزحاف لـ "مفاعيلن مفاعيلن".

وسنكتب كود الدالة HasBahr بعد قليل.

لو وصلنا إلى هذا السطر ونجونا من كل المقاصل التي تتخلص من التفعيلات غير المقبولة، نستطيع بكل ثقة إضافة التفعيلة التي عثرنا عليها إلى مصفوفة التفعيلات:

```
Tafs + tafID
```

وهنا سيتبقى أمامنا احتمالان:

إما أن هذه هي التفعيلة الأخيرة (lastTaf = true) وعلينا أن نضيف مسار التقطيع الذي حصلنا عليه إلى النتائج التي نحفظها في مصفوفة التقطيع Taqtee3، وإما أننا ما زلنا ننتظر تفعيلات تالية، لهذا يجب على الدالة GetTafs استدعاء نفسها لتحصل على التفعيلة الموجودة في الموضع التالي:

```
if lastTaf
   Taqtee3 + [
      :Tafs = Tafs,
      :Bohor = probBohor[Len(probBohor)]
   ]
else
   GetTafs(endAt + 1)
end
```

لاحظ أن العنصر الجديد الذي أضفناه لمصفوفة التقطيع هو مصفوفة تحتوي على خانتين:

١- التفعيلات التي وضعناها في المصفوفة Tafs.. تذكر أن المعامل = يؤدي عملية نسخ قيمي By Value، أي أنه ينشئ مصفوفة جديدة وينسخ محتويات المصفوفة Tafs إليها.. لهذا ستكون التفعيلات التي أضفناها لمصفوفة نتائج التقطيع آمنة، ولن تتغير مع حذف العناصر من المصفوفة الأصلية Tafs وإضافة عناصر جديدة إليها أثناء فحصنا لمسارات التقطيع الأخرى.

٢- البحور الممكنة التي تنتمي إليها هذه التفعيلات، وهي البحور الموجودة في آخر خانة في المصفوفة probBohor، فهي البحور المشتركة بين التفعيلة الأخيرة وكل التفعيلات السابقة لها.. وسنرى كيف حصلنا على هذه البحور عند كتابة كود الدالة HasBahr.

في هذه اللحظة، نكون قد انتهينا من اللفة الحالية في حلقة التكرار وأنهينا دراسة التفعيلة المحتملة بالطول الحالي، لكن قبل انتقالنا إلى اللفة التالية لدراسة تفعيلة بطول آخر، علينا أولا أن نحذف آخر تفعيلة أضفناها إلى المصفوفة Tafs.. تمنحك رينج الدالة del لحذف خانة من المصفوفة المرسلة إلى المعامل الأول، من الموضع المرسل إلى المعامل الثاني (تذكر أن موضع آخر تفعيلة نتعامل معها محفوظ في المتغير tafPos).. وبالمثل، علينا أن نحذف آخر خانة من تفعيلة البحور المحتملة:

```
        // ……
        del(Tafs, tafPos)
        del(probBohor, Len(probBohor))
    Next
EndFunc
```

وبهذا نكون قد أنهينا كود حلقة التكرار، وكود الدالة GetTafs.

الدالة HasBahr:

هذه الدالة هي التي ستعثر على البحر الذي تنتمي إليه التفعيلات، وهي بهذا تؤدي دورا هاما في تحليل الشطرة عروضيا، لكن لها وظيفة أخرى لا تقل أهمية، فهي تبتر المسارات الخاطئة مبكرا فتمنع البرنامج من دراسة أشجار فرعية طويلة بلا لزوم،

فبمجرد الوصول إلى تفعيلة لا تملك أي بحور مشتركة مع التفعيلات السابقة، ستعيد هذه الدالة False لتخبر الدالة GetTafs بأن هذه التفعيلة لا تصلح، وقد رأينا في المثال كيف أن هذه الدالة ستؤدي إلى بتر ٦ من ٨ مسارات محتملة عند دراسة "مستفعلن مستفعلن" أي أنها وفرت علينا وقت دراسة ٧٥% من الاحتمالات التي لا لزوم لها، ولك أن تتخيل كم عدد الاحتمالات الممكنة عند التعامل مع "مستفعلن مستفعلن مستفعلن" أو عدد تفعيلات أطول من هذا في شعر التفعيلة، حيث تزيد شجرة الاحتمالات بشكل مخيف، وهو أمر سيجعل دراسة عدة أبيات يستهلك وقتا طويلا جدا لو لم تَقُم الدالة HasBahr ببتر شجرة المسارات غير الممكنة من جذورها.

والآن لننظر لكود هذه الدالة، وهي تستقبل ثلاثة معاملات:

- رقم التفعيلة في جدول التفعيلات tafID.
- موضع التفعيلة في الشطرة الحالية tafPos.
- هل التفعيلة هي آخر تفعيلة في الشطرة lastTaf.

```
Func HasBahr(tafID, tafPos, lastTaf)
   probBohor + []
   L = Len(probBohor)
   tafInfo = TafsInfo[tafID]
   // .....
EndFunc
```

في بداية كود الدالة سنضيف خانة جديدة لمصفوفة البحور المحتملة probBohor، هذه الخانة نفسها مصفوفة، ستكون فارغة في البداية ثم سنضيف فيها البحور المشتركة بين التفعيلة الحالية والتفعيلات السابقة.. وسنضع طول المصفوفة probBohor في متغير اسمه L، وسنضع المعلومات الخاصة بالتفعيلة الحالية في متغير اسمه tafInfo لاختصار كتابة الكود.. تذكر أن معلومات التفعيلة الحالية موجودة في الخانة رقم tafID في المصفوفة TafsInfo.. لكن بدلا من أن نكتب مثلا:

```
TafsInfo[tafID][:EllaOf]
```

سيكون أكثر اختصارا أن نستخدم المتغير tafInfo فنكتب:

```
tafInfo[:EllaOf]
```

لكن لاحظ أن عيب هذه الطريقة أنها تؤدي إلى إنشاء مصفوفة جديدة يشير إليها

المتغير tafInfo، ويتم نسخ العناصر من المصفوفة TafsInfo[tafID] إليها، فكما قلنا استخدام المعامل = يعني نسخا قيميا By Value لا مرجعيا By Ref، وهذا سيبطئ من سرعة البرنامج خاصة داخل حلقات التكرار والدوال الارتدادية، كما أن هذا لا يصلح لو كنت تتعامل مع مصفوفة تريد تغيير بعض عناصرها أو إضافة عناصر إليها أو حذفها منها، فكل العمليات التي ستجريها عبر المتغير tafInfo لن تؤثر على TafsInfo[tafID].. إذن فهذه الطريقة تصلح للقراءة فقط.. لكل هذا ستجدني أستخدم المتغير tafInfo هنا لتسهيل الشرح، لكن لن أستخدمه في كود البرنامج ليكون أسرع.

مهمتنا الآن أن نجري عملية التقاطع بين بحور التفعيلة الحالية والتفعيلات السابقة.. هذه المهمة ستؤديها دالة اسمها Intersect سنكتب كودها بعد قليل.

لدينا هنا ٣ حالات للتفعيلة:

١- أن تكون تفعيلة عادية (رقمها أصغر من ٢٥) أي أنها ليست تفعيلة علة، وفي هذه الحالة سنجري عملية تقاطع البحور بغض النظر عن موضع هذه التفعيلة في الشطرة (حتى لو أتت في آخر موضع):

```
if tafID < 25
   Intersect(tafInfo, tafPos, L)
end
```

٢- إذا كانت التفعيلة في الموضع الأخير (بغض النظر عن رقمها سواء أصغر من ٢٥ أم أكبر، فالتفعيلة العادية قد تكون هي نفسها علة لتفعيلة أخرى)، وفي هذه الحالة علينا أن نستخدم الحقل EllaOf الخاص بالتفعيلة، للحصول على تفعيلاتها الأصلية المعتلة، ومن ثم نكتب حلقة تكرار لإجراء تقاطع بين بحور التفعيلات الأصلية، وبحور التفعيلات السابقة في الشطرة الحالية:

```
if lastTaf
   ellaTafs = tafInfo[:EllaOf]
   tL = len(ellaTafs)
   for i = 1 to tL
      ellaTaf = ellaTafs[i]
      Intersect(TafsInfo[ellaTaf], tafPos, L)
   next
end
```

٣- إذا كانت التفعيلة في الموضع الأول من الشطرة، فيمكن أن تكون ناتجة عن دخول زحاف الخرم على تفعيلة أخرى.. لاحظ أننا لن نضع هذا الشرط في مقطع ElseIf مترابط مع الشرط السابق، ففي شعر التفعيلة يمكن أن يأتي سطر فيه تفعيلة واحدة، وبهذا تكون هي التفعيلة الأولى والأخيرة في نفس الوقت، ويمكن أن يدخل عليها الخرم والعلة معا!.. لمعالجة حالة الخرم، سنستخدم الحقل KharmOf للحصول على التفعيلة الأصلية التي حولها الخَرم إلى التفعيلة الحالية، ومن ثم سنستدعي إجراء التقاطع معها:

```
if tafPos = 1
   kharmTaf = tafInfo[:KharmOf]
   if kharmTaf!= ""
      Intersect(TafsInfo[kharmTaf], 1, L)
   end
end
```

أخيرا، سنفحص الخانة الأخيرة في المصفوفة probBohor، فلو كان طولها صفرا أي أنها لا تحتوي على أي بحور، فهذا معناه أن بحور التفعيلة الحالية لا تتقاطع مع بحور التفعيلات السابقة، وبالتالي لا يوجد بحر ممكن لهذا التقطيع، وعلينا أن نحذف الخانة الأخيرة من المصفوفة probBohor، ونجعل الدالة HasBahr تعيد false.. أما إن كانت هناك بحور محتملة، فسنعيد true:

```
if len(probBohor[L]) = 0
   del(probBohor, L)
   return false
else
   return true
end
```

وبهذا يكون كود الدالة HasBahr قد انتهى.

الدالة Intersect:

لن تؤدي هذه الدالة عملية التقاطع مباشرة، ولكن كل ما ستفعله هو أن تحصل على البحور الممكنة للتفعيلة، وترسلها إلى الدالة AddToProbBohor التي ستجري عملية التقاطع.. تذكر أن لكل تفعيلة ثلاثة أنواع من البحور:

- BahrAllN: البحور البسيطة التي تظهر فيها التفعيلة في أي موضع من الشطرة.
- BahrN21 و BahrN20: البحور الثنائية التي تظهر فيها التفعيلة في المواضع الفردية والزوجية.. نستطيع معرفة المواضع الفردية بالتأكد من أن باقي قسمة موضع التفعيلة على ٢ يساوي ١.. وتستخدم رينج % لترمز لعملية باقي القسمة.
- BahrN31 و BahrN32 و BahrN30: البحور الثلاثية التي تظهر فيها التفعيلة في المواضع التي باقي قسمتها على ٣ يساوي ١ و ٢ و ٠ بالترتيب.

هذا هو كود هذه الدالة:

```
Func Intersect(TafInfo, tafPos, L)
  AddToProbBohor(TafInfo[:BahrAllN], L)

  If TafPos % 2 = 1
    AddToProbBohor(TafInfo[:BahrN21], L)
  Else
    AddToProbBohor(TafInfo[:BahrN20], L)
  End #If

  Switch TafPos % 3
    Case 1
      AddToProbBohor(TafInfo[:BahrN31], L)
    Case 2
      AddToProbBohor(TafInfo[:BahrN32], L)
    Else
      AddToProbBohor(TafInfo[:BahrN30], L)
  End #Switch
EndFunc
```

الدالة AddToProbBohor:

ستحصل هذه الدالة على البحور المشتركة بين التفعيلة الحالية والتفعيلات السابقة.. أبسط وأسرع طريقة لفعل هذا، هو الخوارزمية التالية:

- نضع بحور التفعيلة الأولى في الخانة probBohor[1].. لاحظ أن كل خانى في المصفوفة probBohor ستكون في حد ذاتها مصفوفة.
- لمعرفة البحور المشتركة بين التفعيلة الثانية والتفعيلة الأولى، نجري عملية التقاطع بين بحور التفعيلة الثانية وبين البحور الموجودة في الخانة probBohor[1]، ونضع ناتج التقاطع في الخانة probBohor[2].
- لمعرفة البحور المشتركة بين التفعيلة الثالثة والتفعيلتين الأولى والثانية، نجري عملية التقاطع بين بحور التفعيلة الثالثة وبين probBohor[2] (لأنها تحتوي على ناتج تقاطع الخانتين الأولى والثانية)، ونضع ناتج التقاطع في probBohor[3].
- وبصيغة عامة: لمعرفة البحور المشتركة بين التفعيلة رقم L والتفعيلات السابقة لها، يكفي أن نجري عملية التقاطع بين بحور التفعيلة رقم L، والبحور الموجود في الخانة probBohor[L-1]، ونضع ناتج التقاطع في الخانة probBohor[L].

ولكن كيف نجري عملية التقاطع؟

سنكتب حلقة تكرار تمر عبر بحور التفعيلة، وتبحث عن كل بحر منها في الخانة probBohor[L-1]، فلو كان موجودا فيها نضيفه للخانة probBohor[L].. لكن تذكر أن بحور كل تفعيلة يضاف إليها بحور العلة وبحر الخرم (لهذا نستدعي الإجراء Intersect ثلاث مرات لنفس التفعيلة)، وحتى لا يسبب هذا تكرار أي بحر أكثر من مرة في ناتج التقاطع، سنتأكد أولا أن البحر ليس موجودا في الخانة probBohor[L] قبل إضافته إليها.

هذا هو الكود:

```
Func AddToProbBohor(bohor, L)
   bL = len(bohor)
   for i = 1 to bL
      bahr = bohor[i]
      if (L = 1 or Find(probBohor[L - 1], bahr)) and
         Find(probBohor[L], bahr) = 0
            probBohor[L] + bahr
      end
   next
EndFunc
```

وهكذا نكون قد أنهينا كود هذه الفئة بحمد الله، والخطوة التالية هي أن نحلل نتائج البحور التي حصلنا عليها لنعرف البحر الذي تنتمي غليه القصيدة ونقطع أبياتها عليه. هذا هو موضوع الفصل التالي.

-٨-
تحليل النتائج

فئة تحليل البيانات Analyzer:

لدينا الآن كل المكونات اللازمة لتحليل القصيدة عروضيا، وكل ما علينا هو جمعها معا للحصول على النتائج المطلوبة وعرضها للمستخدم.

أنشئ ملفا جديدا اسمه Analyzer.ring واحفظه في مجلد البرنامج.. في بداية هذا الملف علينا تحميل ملفات الكود التي كتبناها سابقا لأداء وظائف برامج الشاعر:

```
load "aString.ring"
load "SpWords.ring"
load "AroodWrite.ring"
load "Romooz.ring"
load "Taqtee3.ring"
```

بعد هذا سنكتب فئة التحليل Analyzer Class، وهي تستقبل القصيدة التي يكتبها المستخدم (سنرسلها إلى معامل منشئ الفئة Init).. هذا هو الهيكل العام لها:

```
Class Analyzer
   ShatraatCount = 0
   ShataraatInfo = [ ]
   QBohor = [ ]

   Func init(strQaseedah)
      if isAString(strQaseedah)
         Qaseedah = strQaseedah
      else
         Qaseedah = new aString(strQaseedah)
      end

      // سنحلل القصيدة هنا
   EndFunc
EndClass
```

وتمتلك هذه الفئة ثلاث خصائص:

١- عدد شطرات القصيدة ShatraatCount

٢- مصفوفة تحوي المعلومات الناتجة عن تحليل كل شطرة ShataraatInfo.

٣- مصفوفة سنضع فيها بحور القصيدة QBohor.

فلنكتب الكود الذي سيوضع في الدالة Init في موضع التعليق:

```
// سنحلل القصيدة هنا
```

لتحليل القصيدة سنبدأ بتقطيعها إلى شطرات باستخدام الوسيلة aString.GetLines:

```
shataraat = Qaseedah.GetLines( )
```

ثم سنكتب حلقة تكرار لتحليل كل شطرة عروضيا:

```
ShatraatCount = Len(shataraat)
if ShatraatCount = 0 return End
For i = 1 to ShatraatCount
   shatrah = shataraat[i]

   // تقطيع الشطرة إلى كلمات
   words = shatrah.GetWords( )

   // معالجة الكلمات الخاصة إملائيا
   FixSpWords(words)

   // تحويل الكتابة الإملائية إلى كتابة عروضية
   AroodWrite(words)

   // الحصول على الرموز الصوتية المقابلة للكتابة العروضية
   shatrahRomooz = new Romooz(words, true)

   // الحصول على التقطيعات العروضية المحتملة للشطرة
   TaqTee3 = GetTaqtee3(shatrahRomooz.ToString( ))
   //…..
Next
```

لا جديد حتى الآن.. نحن فقط نجمع معا الأجزاء التي صنعناها سابقا لتكوين الصورة الكبيرة.. الجديد هنا سيكون في تحليل التقطيعات التي حصلنا عليها، لنعرف أي البحور أكثر احتمالا (البحر الذي ينتمي له أكبر عدد من الشطرات).. لفعل هذا، سنجمع كل المعلومات التي حصلنا عليها في مصفوفة واحدة اسمها ShataraatInfo،

ويمكننا استخدام الدالة List لنحجز للمصفوفة مسبقا عدد خانات مساويا لعدد شطرات القصيدة.. ضع هذه الجملة قبل بداية حلقة التكرار السابقة:

ShataraatInfo = List(ShatraatCount)

والآن، في موضع التعليق .....// أضف الكود التالي:

```
ShataraatInfo[i] = [
   :Shatrah = shatrah,
   :AroodWrite = words,
   :Romooz = shatrahRomooz,
   :TaqTee3 = TaqTee3
]
```

وبهذا يكون كود حلقة التكرار قد انتهى، ولدينا كل المعلومات عن كل شطرات القصيدة.. لكن بقي شيء واحد فقط: أن نحلل هذه المعلومات، وسنفعل هذا باستدعاء الإجراء AnalyzeBohor في السطر التالي لنهاية حلقة التكرار (بعد Next):

AnalyzeBohor( )

الخطوة التالية إذن أن نكتب كود الإجراء AnalyzeBohor.

ملحوظة:

أضف السطر التالي في نهاية الدالة Init:

Return Self

هذا السطر يجعل الدالة Init تعيد النسخة الحالية من الفئة، وهو ما سيبدو لك بلا فائدة، لأن هذا هو ما يحدث فعلا عند استدعاء الدالة init باستخدام الجملة:

X = New Analyzer("…")

حيث يعيد الأمر New النسخة التي أنشأها من الفئة ويضعها في المتغير X.

هذا صحيح، لكن رينج تعاني من مشكلة طفيفة تمنعك من استخدام صيغة نطاق الكائن في نفس سطر تعريف نسخة جديدة:

New Analyzer("…") { Y = QBohor }

لكن الحل بسيط كما ذكرت: فقط أضف الجملة Return Self إلى نهاية الدالة Init وسيصير بإمكانك استخدام الصيغة السابقة بشكل طبيعي دون اعتراض من رينج.

دالة تحليل البحور AnalyzeBohor:

سنضيف هذه الدالة في الجزء الخاص private في نهاية الفئة:

```
private
   Func AnalyzeBohor( )
      for shInfo in ShataraatInfo
         shBohor = GetShBohor(shInfo)

         for bahr in shBohor
            index = find(QBohor, bahr, 1)
            if index = 0
               QBohor + [bahr, 1]
            else
               QBohor[index][2] += 1
            end
         next
      next
      QBohor = sort(QBohor, 2)
   EndFunc
```

في هذه الدالة كتبنا حلقتي تكرار متداخلتين:

- الخارجية تمر على كل شطرة في القصيدة، حيث نستدعي الدالة GetShBohor للحصول على البحور المحتملة لكل شطرة.
- والداخلية تمر على كل بحر محتمل في الشطرة الحالية، وتضيفه للمصفوفة QBohor، مع ملاحظة أن كل خانة في هذه المصفوفة تحتوي على مصفوفة فيها خانتان:

١. الخانة الأولى نضع فيها البحر.

٢. والخانة الثانية نضع فيها عدد الشطرات التي ظهر فيها هذا البحر في القصيدة.. ولحساب هذا، ، نبحث عن البحر أولا في المصفوفة QBohor قبل إضافته إليها، فإن لم نجده أضفنا خانة جديدة للمصفوفة نضع فيها البحر والرقم ١ لأن هذه أول شطرة يظهر فيها:

```
QBohor + [bahr, 1]
```

وإن وجدناه فستعيد الوسيلة find رقم الخانة التي يوجد فيها البحر حيث ستوضع في المتغير Index، لهذا سنعدّل الخانة QBohor[index] لنزيد ١ على قيمة الخانة الثانية في المصفوفة الداخلية لزيادة عدد الشطرات التي ظهر فيها البحر:

```
QBohor[index][2] += 1
```

لو كان السطر السابق مربكا لك، فيمكنك كتابته على سطرين للتبسيط:

```
value = QBohor[index]
value[2] += 1
```

وبعد انتهاء حلقتي التكرار، ستتبقى أمامنا خطوة أخيرة، وهي ترتيب المصفوفة QBohor على حسب عدد مرات ظهور كل بحر في شطرات القصيدة.. سنفعل هذا باستخدام واحدة من دوال رينج الجاهزة اسمها sort، وهي تستقبل المصفوفة التي تريد ترتيبها، ورقم العمود الداخلي الذي تريد الترتيب على أساسه.. في حالتنا هذه تحتوي المصفوفة QBohor على عمودين (خانتي المصفوفة الداخلية)، ونحن نريد الترتيب على حسب العمود الثاني لأنه يمثل عدد مرات تكرار البحر.. ونظرا لأن الدالة Sort تعيد مصفوفة جديدة مرتبة، فسنضع المصفوفة العائدة في المصفوفة QBohor لنتخلص من المصفوفة القديمة وتتبقى لدينا المصفوفة الجديدة المرتبة:

```
QBohor = Sort(QBohor, 2)
```

لكن لدينا مشكلة صغيرة هنا: هذا الترتيب سيكون تصاعديا، أي أن أول بحر في المصفوفة QBohor هو الأقل ظهورا في شطرات القصيدة، ونحن نحتاج للترتيب التنازلي، لأن المستخدم بالتأكيد يهمه تقطيع القصيدة على البحر الأكثر احتمالا!

لا تقلق، فحل هذه المشكلة سهل، حيث نستطيع أن نمر على هذه المصفوفة بالعكس (من آخر خانة إلى أول خانة) لعرض البحور الأكثر احتمالا أولا.

وهكذا تكون الدالة AnalyzeBohor قد انتهت.

الدالة GetShBohor:

تحصل هذه الدالة على البحور المحتملة في الشطرة.. لفعل هذا سنمر على التقطيعات المحتملة للشطرة، ونحصل على البحور المحتملة لكل تقطيع، ونضيفها للمصفوفة shBohor، مع ملاحظة أن البحر قد يظهر أكثر من مرة بصيغ تقطيع مختلفة، لهذا منعا للتكرار سنتأكد أولا أن البحر ليس موجودا بالفعل في المصفوفة shBohor قبل إضافته إليها:

```
Func GetShBohor(shInfo)
   shBohor = []
   For taqtee3 in shInfo[:Taqtee3]
      For bahr in taqtee3[:Bohor]
         if not find(shBohor, bahr)
            shBohor + bahr
         end
      Next
   Next
   Return shBohor
EndFunc
```

وهكذا نكون قد حققنا الهدف وانتهينا تماما من تحليل نتائج البرنامج.. لكن بقي أمامنا شيء واحد نضيفه للفئة Analyzer، وهي الدالة GetTafs.

الدالة GetTafs:

سنحتاج لهذه الدالة عند عرض النتائج للمستخدم، ووظيفتها إعطاؤنا بعض المعلومات عن تفعيلات البحر الذي سنعرض تقطيع القصيدة عليه، لهذا سنرسل لهذه الدالة معلومات الشطرة ورقم البحر المراد تقطيعها عليه، وستعيد إلينا مصفوفة فيها خانتان:

١. الخانة الأولى اسمها Lens وتحتوي على مصفوفة فيها أطوال رموز التفعيلات، لنستخدمها في التقطيع العروضي للشطرة لتناسب تفعيلات البحر.
٢. الخانة الثانية اسمها Names وتحتوي على مصفوفة فيها أسماء التفعيلات.

هذا هو كود هذه الدالة، وسنضعها في القسم العام من الفئة (قبل القسم private) حتى يمكننا استدعاؤها من خارج الفئة:

```
Func GetTafs(shInfo, bahrId)
   for taqtee3 in shInfo
      if Find(taqtee3[:Bohor], bahrId)
         TafLens = [ ]
         TafNames = [ ]
         For tafId in taqtee3[:Tafs]
            TafLens + len(TafsInfo[tafId][1])
            TafNames + TafsInfo[tafId][:Name]
         next
         Return [:Lens = TafLens, :Names = TafNames]
      end
   next
EndFunc
```

فكرة الدالة بسيطة، حيث نمر على كل تقطيع محتمل للشطرة، ونبحث في بحور هذا التقطيع عن البحر المطلوب، فإن وجدناه فسنمر على كل تفعيلات هذا التقطيع، لنحصل على طول رموز التفعيلة واسمها (تذكر أن رمز التفعيلة هو أول خانة في مصفوفة معلومات هذه التفعيلة).. وسنضيف هاتين المعلومتين مؤقتا إلى المصفوفتين TafLens و TafNames، وبعد انتهاء حلقة التكرار نضع المصفوفتين في مصفوفة الناتج ونجعل الدالة تعيدها:

```
Return [:Lens = TafLens, :Names = TafNames]
```

لاحظ أن الأمر Return ينتهي تنفيذ الدالة كلها في الحال، وهذا يعني إنهاء حلقة التكرار For Next التي كانت تبحث عن البحر في التقطيعات المحتملة.. تذكر أن البحر قد يظهر في أكثر من تقطيع (كأن تنتهي الشطرة بـ "فعولن فع" وتظهر في تقطيع آخر "مفاعيلن").. إذن فسنأخذ أول احتمال ونتجاهل الاحتمالات الباقية.

وهكذا نكون قد أنهينا تماما كود الفئة Analyzer.

فلنرَ إذن كيف يمكننا استخدامها لعرض النتائج للمستخدم.

الدالة ToConsole:

سنكتب الآن دالة اسمها ToConsole، سنرسل إليها نص القصيدة، لتقوم بعرض تحليلها العروضي على شاشة المخرجات.

أنشئ ملفا جديدا اسمه ToConsole.ring، وأضف فيه الكود التالي:

```
Load "Analyzer.ring"

Func ToConsole(Qaseedah)
   new Analyzer(Qaseedah) {
      كود عرض النتائج //
   }
EndFunc
```

لاحظ أننا نستطيع استخدام القوسين المتعرجين { } بعد اسم أي متغير يحمل كائنا Object، لصنع نطاق خاص بهذا الكائن، حيث نستطيع داخل هذا النطاق التعامل مع عناصر الكائن مباشرة بدون نسبتها إلى اسمه.. وفي الكود السابق عرّفنا نسخة جديدة من الفئة Analyzer لكننا لم نضعها في متغير، لأننا سنتعامل مع خصائصها ووسائلها مباشرة داخل القوسين { } وبدون الحاجة لنسبتها لاسم الفئة.. مثل:

```
new Analyzer(Qaseedah) {
   bCount = len(QBohor)
}
```

حيث إن QBohor هي خاصية تابعة للفئة Analyzer، وفي الكود السابق ستفهم رينج هذا وستعرف أنك تقصد التعامل مع هذه الخاصية في الكائن الجديد الذي أنشأته من الفئة Analyzer.

هذه ميزة قوية جدا في رينج، حيث تستطيع استخدام عناصر الفئة مباشرة داخل القوسين المتعرجين كنوع من الاختصار، وإن خفت من حدوث تضارب بين اسم عنصر من عناصر الفئة واسم متغير عادي، فيمكنك استخدام الكلمة Self للإشارة إلى الكائن الذي تكتب الكود بين قوسيه، مثل:

```
new Analyzer(Qaseedah) {
   bCount = Len(Self.QBohor)
}
```

وقد استفدت من ميزة نطاق الكائن عمليا، ففي البداية كنت قد كتبت الدالة ToConsole داخل الفئة Analyzer وهذا يعني أنني كنت أتعامل مع عناصر الفئة Analyzer بأسمائها مباشرة لأنني داخل الفئة.. ثم قررت أن أخرج الدالة ToConsole من الفئة وأضعها في ملف مستقل، وهذا يستوجب تعريف نسخة من الفئة ووضعها في متغير، وتعديل الكود لأنسب إلى هذا المتغير أسماء عناصر الفئة Analyzer المستخدمة في كود الدالة ToConsole.. لكني وجدت أن الأسهل والأسرع هو وضع كل الكود كما هو بدون تعديل داخل المقطع:

```
new Analyzer(Qaseedah) {
}
```

ليعمل كل شيء كما ينبغي بلا مشاكل.. تسهيل مريح فعلا من رينج.

والآن فلننظر إلى كود عرض النتائج:

في البداية سنعرض رسالة لو لم تكن القصيدة موزونة على أي بحر:

```
bCount = len(QBohor)
n = 1
bahrName = ""
If bCount = 0
   see "لا يوجد بحر" + nl
Else
   // عرض أسماء البحور
End #If
// عرض تقطيع القصيدة
```

في المقطع Else سنعرض للمستخدم أسماء البحور المحتملة، مرتبة تنازليا على حسب البحر الأكثر ظهورا في شطرات القصيدة (كما قلنا سنمر عبر مصفوفة البحور من النهاية إلى البداية لأنها مرتبة تصاعديا).. وسنضع قبل كل اسم بحر رقما مسلسلا، لأننا سنطلب من المستخدم إدخال رقم البحر الذي يريد تقطيع القصيدة عليه.. وسنضع بعد كل بحر النسبة المئوية له، وهي تنتج من قسمة عدد الشطرات التي يظهر فيها على عدد شطرات القصيدة، وضرب الناتج في ١٠٠.

ضع هذا الكود بدلا من التعليق الموجود في المقطع Else:

```
if bCount > 1
   see "البحور المحتملة:" + nl
   for i = bCount to 1 step -1
      bhr = QBohor[i]
      bId = bhr[1]
      ? "" + n + "-" + BohorInfo[bID][:Name] +
      " (" + 100 * (bhr [2] / ShatraatCount) + "%)"
      n++
   next
   see "اكتب رقم البحر الذي تريد تقطيع القصيدة عليه،"
   see "أو اكتب نصا فارغا أو الرقم صفر لإنهاء البرنامج " + nl
   n = GetANumber(bCount)
   if n = 0 return end
end
```

لاحظ أن الأمر See يعرض النص التالي له في نافذة المخرجات لكن لا يضيف سطرا بعده، لهذا استخدمت المتغير nl لإضافة سطر جديد بعد النص.. كان من الممكن أن استخدم الرمز ? مباشرة لعرض النص وإضافة سطر، لكن كتابة نص عربي بعد علامة الاستفهام يفسد عرض الكود في محرر رينج، لهذا كما ذكرنا سابقا، عليك أن تبدأ السطر الذي فيه نص عربي بكلمة إنجليزية ليظهر بشكل صحيح.. الأمر See يؤدي هذا الغرض.. مع ملاحظة أنني أستخدم حيلة أخرى أحيانا، بأن أضع النص العربي في متغير اسمه txt ثم أستخدم الأمر ? لعرض محتوى هذا المتغير، لأتحاشى وضع النص العربي بعد علامة الاستفهام.

ملحوظة:
يؤدي الاسم See إلى بعض الارتباك بين مستخدمي رينج الجدد، فحينما تطلب من رينج أن "ترى"، فهذا يعني ظاهريا أنك تطلب منها أن تنظر لما كتبه المستخدم في نافذة المخرجات، وهذا عكس ما يقوم به الأمر See.. وبسبب هذا الارتباك أضافت رينج أمرا آخر اسمه Put لوضع النص في نافذة المخرجات، وستجدني أستخدم كليهما هنا وهناك.

على كل حال، مترجم الكود compiler لا تعنيه كلمات اللغة الإنجليزية، وإنما الأوامر التي برمج لتنفيذها حينما يرى هذه الكلمات.. ولو أردت بعض الاقتناع عند استخدام الكلمة See، ففكر فيها باعتبارها اختصارا للجملة:

Let user **see** "some text"

وهو ما تستطيع كتابته فعلا في رينج بأكثر من طريقة، منها إعادة تعريف أوامر رينج، ومنها استخدام ميزة معالجة اللغات الطبيعية في رينج، أو بمنتهى البساطة بتعريف متغيرين وهميين اسماها Let و user كالتالي:

```
Let = 0
user = 0
Let user see "some text"
```

جرب هذا الكود وسيعمل كالسحر ☺.

ويمكنك تطبيق نفس الكلام على الأمر Give الذي يستقبل قيمة أدخلها المستخدم في نافذة المخرجات، ويضعها في المتغير التالي له.. فكر فيه أيضا على أنه:

Let user **Give** n

لعلك لاحظت أننا استخدمنا الدالة GetANumber في الكود السابق لقراءة رقم البحر الذي يريد المستخدم تقطيع القصيدة عليه.. في هذه الدالة سنكتب حلقة تكرار تستمر في طلب رقم البحر إلى أن يدخل المستخدم رقما صحيحا (محصورا بين ١ و عدد البحور المحتملة في القصيدة) أو صفرا أو نصا فارغا (للدلالة على إلغاء العملية وإيقاف البرنامج).. يمكنك أن تنظر لهذه الدالة في كود البرنامج المرفق بالكتاب، ولن تجد فيه شيئا يستعصي على فهمك.

لاحظ أيضا أن كل الكود السابق موضوع في مقطع جملة الشرط If bCount >1، فلو كان هناك بحر واحد فقط في أبيات القصيدة، فسنعرض تقطيع القصيدة عليه مباشرة ولا داعي لأن نطلب من المستخدم اختيار البحر فلا يوجد اختيار هنا.. وفي كلتا الحالتين (سواء كان عدد البحور ١ أو أكثر) سنكتب بعد نهاية مقطع جملة الشرط، الكود الذي يعرض تقطيع القصيدة على البحر.. هذا هو السبب في أننا لم نستخدم ElseIf في هذا الكود، واستخدمنا بدلا منها Else داخلها جملة شرط، لأن هناك كودا

مستقلا عن جملة الشرط يجب مواصلة تنفيذه بعد جملة الشرط، وهو الكود التالي:

```
i = bCount - n + 1
bID = QBohor[i][1]
bahrName = BohorInfo[bID][:Name]
txt = "تقطيع القصيدة على البحر "
? txt + bahrName
```

الجزء الهام في الكود السابق هو السطر الأول:

```
i = bCount - n + 1
```

فالرقم n الذي أدخله المستخدم هو رقم البحر بين البحور المعروضة أمامه على الشاشة، لكن هذه البحور معروضة تنازليا على عكس ترتيبها الحقيقي في المصفوفة QBohor.. فلو كتب المستخدم ١، فهو يقصد البحر الأخير في المصفوفة!.. ويمكننا الحصول على الرقم الحقيقي للبحر في المصفوفة، بطرح الرقم الذي أدخله المستخدم من عدد البحور وإضافة ١.

ولكن، أين هي البيانات التي سنعرضها للمستخدم؟.. كل ما عرضناه حتى الآن هو الجملة: تقطيع البحور على البحر الفلاني.. فأين باقي المعلومات؟

لا تقلق.. سنكتب هذا الكود حالا، وسنضعه بعد نهاية جملة الشرط الخارجية، في موضع التعليق:

```
عرض تقطيع القصيدة //
```

السبب في هذا أنه حتى لو كانت القصيدة غير موزونة على بحر، فيجب أن نعرض للمستخدم الكتابة العروضية لكل شطراتها لعل هذا يساعده في معرفة موضع الكسر في الأبيات، أو ربما يكتشف أنه نسي وضع علامة سكون أو شدة أو تنوين فوق بعض الحروف.

ونفس هذا سنفعله مع الشطرات المكسورة التي لا تنتمي للبحر الذي اختاره المستخدم.. ولن نعرض الكتابة العروضية للشطرات الصحيحة، لأن الكتابة العروضية تظهر ضمنيا في التقطيع.

والكود الذي سنستخدمه بسيط وواضح ولا جديد فيه، سوى استدعاء الدالة Join لتشبيك كل خانات المصفوفة في نص واحد ووضع فاصل بينها.. في حالتنا هذه سنستخدم

النص " – " كفاصل عند عرض تقطيع القصيدة وعند عرض رموزها كما في:

```
? txt + Join(shTaqtee3, " - ")
```

هذه إذن حلقة التكرار التي تعرض تقطيع الشطرات:

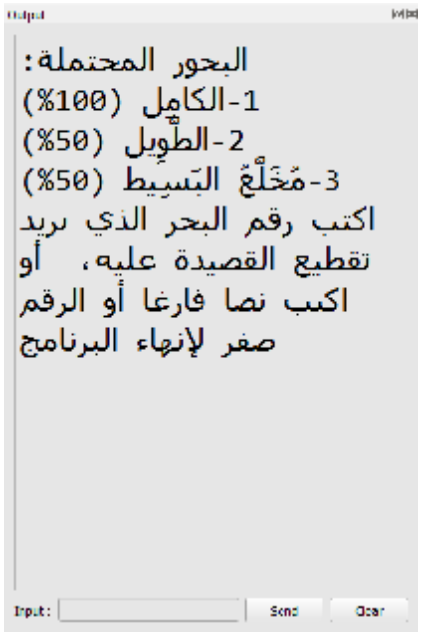
```
For shInfo in ShataraatInfo
   txt = "الشطرة: "
   ? txt + shInfo[:Shatrah].Text
   If bCount = 0
      txt = "الكتابة العروضية: "
      ? txt + Join(shInfo[:AroodWrite], " ")
      Loop
   End

   tafs = GetTafs(shInfo[:Taqtee3], bID)
   if isNull(tafs)
      see "هذه الشطرة غير موزونة على بحر " + bahrName + nl
      txt = "الكتابة العروضية: "
      ? txt + Join(shInfo[:AroodWrite], " ")
   else
      txt = "التقطيع: "
      tafLens = tafs[:Lens]
      shTaqtee3 = shInfo[:Romooz].Horoof(tafLens)
      ? txt + Join(shTaqtee3, " - ")
      txt = "الرموز : "
      romooz = shInfo[:Romooz].Harakaat(tafLens)
      ? txt + Join(romooz, " - ")
      txt = "التفعيلات: "
      ? txt + Join(tafs[:Names], " - ")
   end
   ? "---------------"
next
```

كما ذكرنا، مهمة الدالة Join المرور عبر المصفوفة وتشبيك النص، وهي قادرة على التعامل مع نصوص رينج العادية والنصوص الخاصة بنا من النوع aString.. وقد أضفت هذه الدالة للملف aString.ring (وضعتها خارج الفئة aString، في الجزء العام الموجود أعلى الملف) لأنها دالة عامة يمكننا الاستفادة منها في مواضع كثيرة.. ولا جديد في كودها يحتاج للتوضيح هنا.

وهكذا نكون قد أنهينا المهمة، ولم يبق أمامنا إلا تجربة الكود.. أضف الكود التالي في الملف قبل تعريف الدالة ToConsole:

```
ToConsole(
    "هلّا سألْت القوم يا ابنة مالكٍ"
    "إن كنْت جاهلةً بما لمْ تعْلمي
)
```

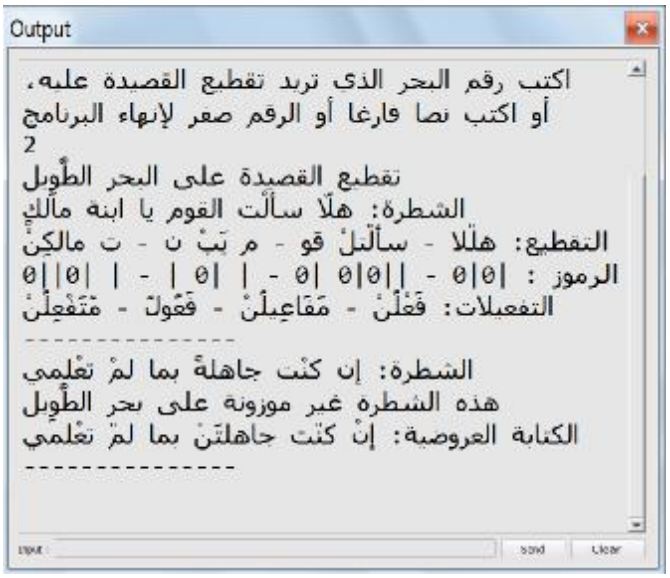


واضغط زر تشغيل البرنامج.. ستعرض شاشة المخرجات النص الموضح في الصورة.

ولإدخال الرقم المطلوب، اكتبه في مربع النص Input الموجود أسفل نافذة المخرجات واضغط الزر Send المجاور له، مع العلم أنك لو تركت مربع النص فارغا (أو كتبت صفرا) وضغطت Send فسيتم إنهاء البرنامج.

واضح أن البيت ينتمي إلى البحر الكامل لأنه موجود بنسبة ١٠٠%.. لرؤية تقطيع القصيدة على هذا البحر اكتب ١ واضغط Send.. هكذا ستبدو لك النتائج:

ويمكنك إعادة تشغيل البرنامج واختيار البحر رقم ٢ لرؤية كيف تظهر نتائجه:



تدريب:
يمكنك تعديل الكود الذي كتبناه، للسماح للمستخدم باختيار بحر آخر بعد عرض تقطيع القصيدة على أحد البحور، بدلا من إغلاق البرنامج وإعادة تشغيله.. سأترك لك التفكير في حل لهذه المسألة لتختبر قدراتك البرمجية بعد كل ما تعلمته في هذا الكتاب.

هذا جميل، لقد نجحنا بالفعل في تقطيع القصيدة، ويمكنك أن تجرب أبياتا من بحور أخرى لاختبار برنامج الشاعر.

لكني لا أظنك راضيا عن هذه الطريقة في عرض النتائج فمظهرها غير مريح، كما أنها تصلح للتجريب فقط، ولا تصلح لتوزيع البرنامج للمستخدمين، فشاشة المخرجات تخص محرر رينج فقط، ولن تظهر للمستخدم.

لحل هذه المشكلة يمكنك عرض نتائج البرنامج في شاشة سطور الأوامر Console الخاصة بنظام التشغيل، وهي التي ستظهر للمستخدم فعلا لو أعطيته الملف التنفيذي exe للبرنامج.. ويمكنك معاينة هذا من محرر رينج بفتح القائمة الرئيسية Program وضغط الأمر Debug أو ضغط Ctrl+D مباشرة من لوحة المفاتيح.. لكنك للأسف ستفاجأ بأن شاشة سطور الأوامر عاجزة عن عرض الحروف العربية، وستعرض بدلا منها نقوشا عجيبة غير مفهومة!

وهذا سبب آخر يدفعنا للبحث عن طريقة أفضل لعرض نتائج البرنامج.. ألن يكون رائعا لو عرضنا للمستخدم نافذة فيها مربع نص يكتب فيه القصيدة، وزر يضغطه لتحليل الأبيات، لتظهر له النتائج على النافذة بشكل منظم ومريح؟

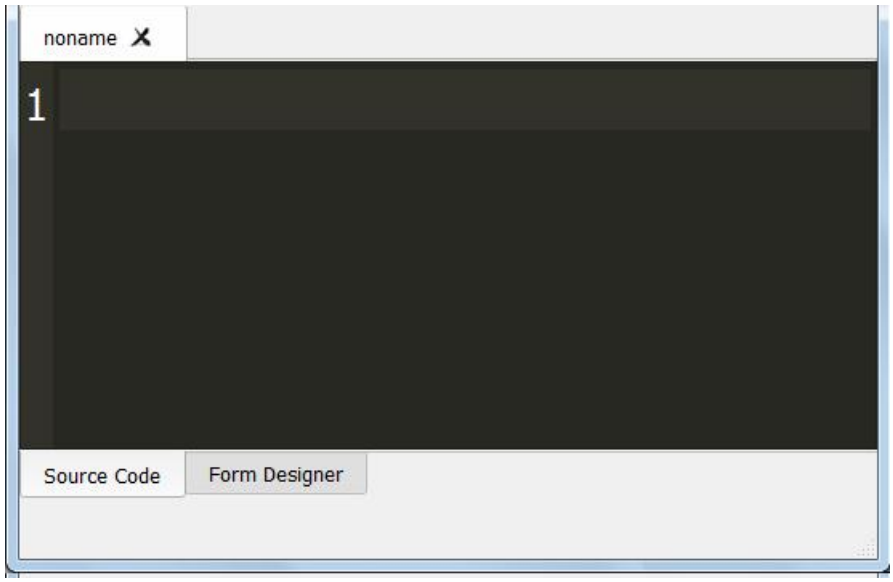
إذا كانت الإجابة بلى، فانتقل إلى الفصل التالي من الكتاب، لنتعلم معا كيف نستخدم مصمم نوافذ رينج لإنشاء مثل هذه النافذة.

– ٩ –

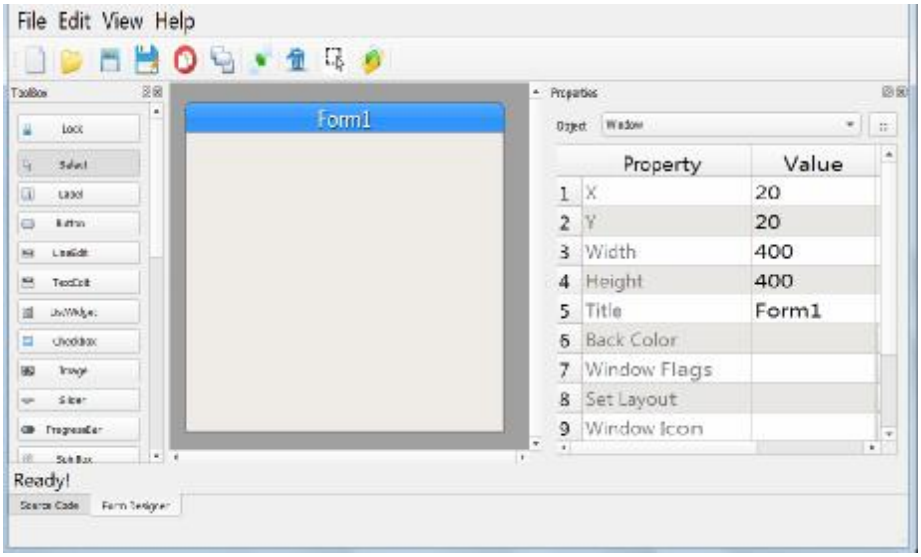
تصميم واجهة مرئية للبرنامج

مصمم النماذج Form Designer:

يحتوى محرر رينج على شاشة تستخدم لتصميم النوافذ، ويمكنك عرضها من القائمة الرئيسية View بضغط الأمر From Designer Window.. لكن حينما تفعل هذا أثناء عرض إحدى نوافذ الكود في محرر رينج، لن يظهر مصمم النوافذ مباشرة، وإنما سيظهر شريط جديد أسفل نافذة الكود مكتوب عليه Form Designer:



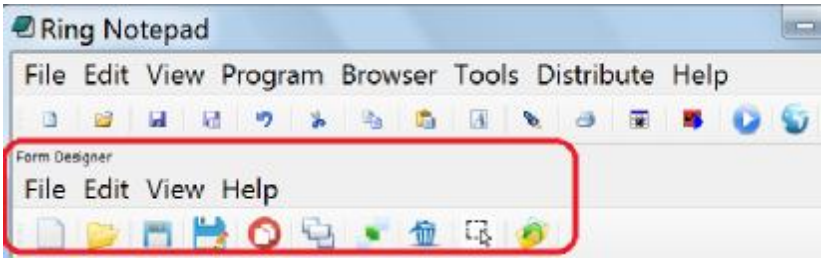
اضغط هذا الشريط بالفأرة لعرض مصمم النوافذ:



هذه الشاشة تتكون من أربعة أقسام رئيسية:

١. القائمة الرئيسية Main Menu وشريط الأوامر Command Bar:

يعرضان كل الأوامر اللازمة لفتح نافذة جديدة وحفظها وتنسيق الأدوات عليها وغير ذلك.. انتبه إلى أن القائمة الرئيسية الخاصة بمصمم النوافذ وشريط الأوامر الموجود تحتها يختلفان عن القائمة الرئيسية وشريط الأوامر الخاصين بمحرر رينج، وأنك سترى هذه العناصر الأربعة معا في نفس النافذة، فلا ترتبك بينها.



٢. صندوق الأدوات ToolBox:

يظهر على يسار الشاشة، ويحتوى على مجموعة أزرار مكتوب على كل منها اسم أداة يمكن رسمها على النافذة، مثل الزر Button ومربع النص TextBox.

٣. سطح تصميم النافذة:

يظهر في منتصف الشاشة، وترى فيه نافذة فارغة، يمكنك أن ترسم عليها الأدوات الموجودة في صندوق الأدوات كما سنرى بعد قليل.

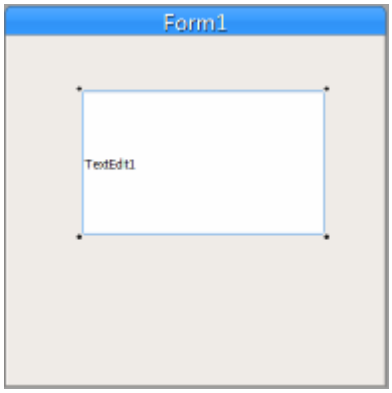
٤. نافذة الخصائص Properties Window:

تظهر على يمين الشاشة، وهي جدول يعرض أسماء وقيم خصائص النافذة أو الأداة المحددة عليها Selected Control، حيث يمكنك تغيير هذه القيم بالضغط فوق خانة القيمة مرتين بالفأرة Double-click وبدء الكتابة.. دعنا نغير خصائص النموذج (النافذة):

	Property	Value
1	X	0
2	Y	-84
3	Width	900
4	Height	570
5	Title	الشاعر
6	Back Color	

- غير قيمة الخاصية Title إلى: الشاعر.. هذا هو النص الذي سيظهر على الشريط العلوي للنافذة.
- غير عرض النافذة بوضع القيمة ٩٠٠ في الخاصية Width.
- غير ارتفاع النافذة بوضع القيمة ٥٧٠ في الخاصية Height.

سترى تأثير هذه القيم على شكل النافذة المعروضة في المصمم في الحال.



والآن، دعنا نضع بعض الأدوات على النافذة: في صندوق الأدوات اضغط الأداة TextEdit ضغطة واحدة بالفأرة، وتحرك بالفأرة فوق النموذج إلى موضع يناسبك.. ستلاحظ أن مؤشر الفأرة قد تحول إلى العلامة + دلالة على أنك تستطيع استخدام الفأرة للرسم فوق النموذج.. لفعل هذا اضغط زر الفارة الأيسر Left-Click ولا تتركه، وتحرك بالفارة فوق النموذج لتحديد المستطيل الذي تريد رسم الأداة داخله، ثم اترك زر الفأرة.. سيتم رسم الأداة داخل المساحة التي حددتها.

لاحظ أن الأداة TextEdit تؤدي وظيفة مربع النص متعدد الأسطر، وهي أداة يستطيع المستخدم أن يكتب فيها نصا مكونا من عدة سطور.. وهناك أداة شبيهة لكنها تسمح بكتابة سطر واحد فقط هي الأداة LineEdit.

الآن بعد أن رسمت مربع النص على النموذج، يمكنك ضغطه بزر الفأرة الأيسر وسحبه (أثناء الاستمرار في ضغط زر الفأرة) لتغيير موضعه.. كما يمكنك التحليق بالفأرة فوق حواف مربع النص إلى أن يتحول مؤشرها إلى شكل السهم، ثم تضغط زرها الأيسر وتسحب الحافة لتغيير مساحة مربع النص.

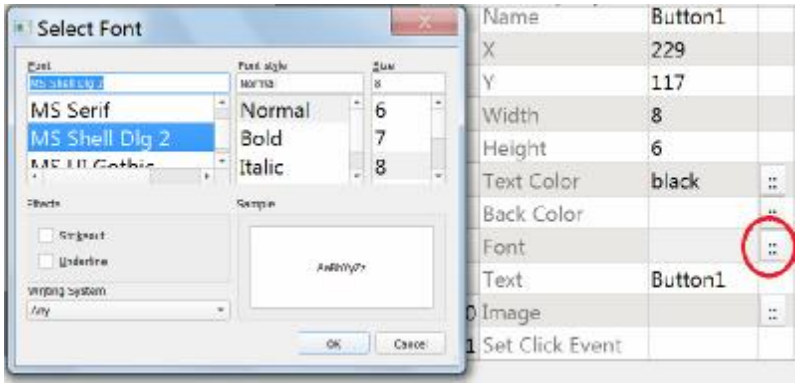
ويمكنك تحديد أي أداة بضغطها مرة بزر الفأرة الأيسر، وينطبق هذا أيضا على النموذج.. ويؤدي تحديد الأداة إلى ظهور مربعات صغيرة زرقاء على رؤوسها الأربعة، مع تحول لون إطارها إلى الأزرق.. وكما قلنا، تظهر خصائص الأداة المحددة SelectedControl في نافذة الخصائص.

حدد مربع النص، ودعنا نغير قيم بعض خصائصه في نافذة الخصائص كما هو موضح في الجدول:

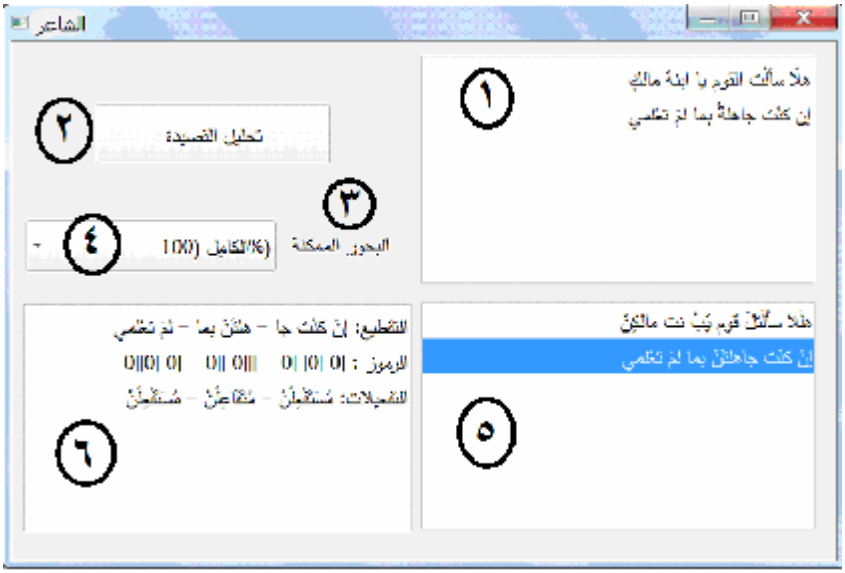
الخاصية	القيمة	ملحوظة
Name	txtQaseedah	الاسم البرمجي للأداة.
X	٤٤٥	الموضع الأفقي، وهو المسافة بين الحافة اليسرى للأداة والحافة اليسرى للنموذج.
Y	١٦	الموضع الرأسي، وهو المسافة بين الحافة العليا للأداة والحافة العليا للنموذج.
Width	٤٣٠	عرض الأداة.
Height	٢٥٠	ارتفاع الأداة.
Text Color	Black	لون النص المكتوب في مربع النص.
Back Color	White	لون أرضية مربع النص.

الخاصية	القيمة	ملحوظة
Font	Simplified Arabic,14, -1,5,50,0, 0,0,0,0	خصائص الخط، مثل اسمه وحجمه وهل هو سميك أو مائل.
Text		النص الذي تعرضه الأداة، وسنتركه هنا فارغا، لأن المستخدم سيكتب القصيدة في مربع النص.

وتستطيع اختيار اللون المناسب للخاصيتين Text Color و Back Color من نافذة الألوان، ويمكنك عرضها بضغط الزر المجاور لخانة القيمة في نافذة الخصائص.. وينطبق نفس الأمر على مواصفات الخط، فلو ضغطت الزر المجاور للخاصية Font فستظهر نافذة اختيار الخط.. ويمكن ألا ترى هذا الزر لو كان عرض نافذة الخصائص صغيرا، ويمكنك تكبيره بالتحرك بالفأرة فوق الحافة اليسرى لنافذة الخصائص إلى أن ترى سهم تغيير الحجم، ثم ضغط الفأرة وسحب الحافة إلى اليسار.



وبنفس هذه الطريقة في تصميم الأدوات، يمكنك تصميم النافذة لتبدو كما في الصورة أثناء تشغيل البرنامج وعرض النتائج:



ويلخص الجدول التالي أنواع الأدوات الموجودة على النافذة ووظائفها، وهي مرقمة بنفس الأرقام الموضحة على الصورة:

م	نوع الأداة	اسمها البرمجي Name	وظيفتها
١	TextEdit	txtQaseedah	مربع نص نكتب فيها أبيات الشعر.
٢	Button	btnExecute	الزر "تحليل القصيدة".
٣	Label	Label1	اللافتة "البحور الممكنة".
٤	ComboBox	cmbBohoor	قائمة منسدلة تعرض أسماء البحور.
٥	ListWidget	lstShataraat	قائمة عناصر تعرض الكتابة العروضية.
٦	TextEdit	txtDetails	مربع نص يعرض تفاصيل الشطرة المحددة في قائمة الكتابة العروضية.

لاحظ أن العمود الثالث يعرض الاسم البرمجي للأداة.. هذا الاسم يجب أن تكتبه في الخاصية Name في نافذة الخصائص (ستجدها أول عنصر في النافذة لأهميتها)،

وتنطبق على هذا الاسم نفس قواعد تسمية المتغيرات في لغة رينج، لأننا سنستخدمه بالفعل في الكود كمتغير يحمل الكائن Object الذي يمثل الأداة.. وقد التزمنا في تسمية الأدوات بوضع بادئة من ثلاثة حروف في أول الاسم هي اختصار لنوع الأداة حتى يسهل علينا فهم الكود، فمثلا استخدمنا البادئة txt مع مربع النص TextEdit، والبادئة btn مع الزر Button... وهكذا.

بعد انتهائك من تصميم النموذج بالشكل المطلوب (وهو موجود ضمن كود المشروع المرفق بالكتاب على كل حال)، اضغط زر الحفظ على شريط أدوات مصمم النماذج، وفي خانة الاسم غير اسم النموذج إلى FrmQaseedah.. لاحظ أن رينج تحفظ النموذج في ثلاثة ملفات، هي:

- FrmQaseedah.rform: يحتوي على خصائص النموذج والأدوات كما حددتها في مصمم النماذج.. لا تغير محتويات هذا الملف يدويا.
- FrmQaseedahView.ring: يحتوي على الكود اللازم لاستخدام مكتبة QT لرسم النموذج والأدوات.. هذا الكود هو الذي يرسم النموذج والأدوات أثناء تشغيل البرنامج.. لا تغير محتويات هذا الملف يدويا.
- FrmQaseedahController.ring: ملف التحكم Controller وهو الذي نكتب فيه الكود الذي يتحكم في وظائف الأزرار ومربعات النص.. هذا هو الملف الذي يعنينا عمليا، وهو الذي سنفتحه في محرر كود رينج.

وحتى يكون مشروعك أكثر تنظيما، أنصحك أن تنشئ مجلدا لكل نموذج لتضع فيه ملفاته الثلاثة.. في حالتنا هذه احفظ ملفات النموذج في مجلد اسمه FrmQaseedah.

وتستطيع إعادة فتح أي نموذج في مصمم النماذج باستخدام الأمر Open من القائمة الرئيسة File من شريط قوائم مصمم النماذج، واختيار الملف ذي الامتداد rForm. .. ويمكنك عرض ملف التحكم الخاص بالنموذج بضغط الشريط Source Code الموجود أسفل مصمم النماذج للانتقال إلى محرر الكود، حيث ستجد الملف مفتوحا، وإن أغلقته بنفسك فيمكنك إعادة فتحه بفتح الملف FrmQaseedahController.ring كما تفتح

أي ملف رينج عادي.

وستجد في هذا الملف بعض الكود، وسترى فيه تعريف فئة التحكم كالتالي:

```
Class frmQaseedahController from windowsControllerParent
  oView = new frmQaseedahView
EndClass
```

وسنكتب كل الكود الخاص بنا داخل الفئة frmQaseedahController.. هذه الفئة ترث الفئة windowsControllerParent (لاحظ وجود الكلمة From بعد اسم الفئة.. هذه هي الكلمة التي تستخدمها رينج لجعل الفئة ترث من فئة أخرى).

ويظهر في الفئة تعريف خاصية اسمها oView حيث البادئة o هي اختصار للكلمة object دلالة على أن هذا هو نوع هذه الخاصية، وهي تحمل نسخة من الفئة frmQaseedahView التي تم تعريفها في الملف FrmQaseedahView.ring، وهي الفئة المسئولة على إنشاء النموذج والأدوات فعليا، لهذا سنتعامل مع كائن النموذج وكائنات الأدوات من خلالها.. ولتسهيل هذا وجعل الكود أكثر اختصارا، دعنا نعرّف مجموعة خصائص لها نفس أسماء الأدوات وتحمل الكائنات الخاصة بها، وسنسمى النموذج هنا win (اختصارا لـ window):

```
Class frmQaseedahController from windowsControllerParent
  oView = new frmQaseedahView
  win = oView.win
  txtQaseedah = oView.txtqaseedah
  cmbBohoor = oView.cmbBohoor
  lstShataraat = oView.lstShataraat
  txtDetails = oView.txtDetails


EndClass
```

نحن جاهزون الآن لكتابة وظيفة الأدوات.

الاستجابة لأحداث الأدوات Handling Events:

تعتمد برمجة النوافذ على الأحداث Events لهذا تسمى البرمجة الموجهة بالأحداث Event-Driven Programming.

والحدث Event هو فعل معين قام به مستخدم البرنامج، مثل الكتابة في مربع النص أو ضغط الزر أو اختيار عنصر من القائمة، ويترتب على هذا الفعل أن يرسل نظام التشغيل (كالويندوز مثلا) رسالة للبرنامج يخبره فيها بما فعله المستخدم، وتقوم كل أداة بقراءة رسائل الويندوز لمعرفة الرسائل (الأحداث) التي تخصها والاستجابة لها، سواء بكتابة الحروف في مربع النص، أو تغيير شكل الزر لإشعارك بأنه صار مضغوطا، أو تلوين العنصر الذي اخترته في القائمة... إلخ.

بعد هذا، يمكن أن تمرر الأداة الرسالة لنا نحن المبرمجين إذا كان من المهم أن نكتب كودا إضافيا للاستجابة للحدث.. لهذا تطلق كل أداة بعض الأحداث الهامة:

- فمربع النص TextEdit يطلق الحدث TextChangedEvent بعد حدوث أي تغيير في النص المكتوب في مربع النص.
- والزر Button يطلق الحدث ClickEvent بعد ضغط الزر.
- والقائمة ListWidget تطلق الحدث CurrentItemChangedEvent عندما يتغير العنصر المحدد حاليا في القائمة.
- والقائمة المركبة ComboBox تطلق الحدث CurrentIndexChangedEvent عندما يتغير العنصر المحدد حاليا في القائمة المنسدلة.

وهذه هي الأحداث التي سنستخدمها لكتابة برنامجنا، حيث سنكتب معالجا Handler لكل حدث.. **معالج الحدث Event Handler** هو دالة سيتم استدعاؤها عندما ينطلق الحدث، لهذا يجب علينا أن نربط كل حدث بالدالة التي ستعالجه (ستستجيب له).. ويمكن فعل هذا ببساطة من نافذة الخصائص فهي تعرض أحداث الأداة في الأسفل (بعد الخصائص)، وكل ما عليك هو أن تنقر مرتين بالفأرة على خانة القيمة المجاورة لاسم الحدث، وتكتب اسم الدالة.. والجدول التالي يلخص أسماء الدوال التي

سنستخدمها لمعالجة أحداث الأدوات:

اسم الأداة	اسم الحدث	اسم المعالج (الدالة)
btnExecute	ClickEvent	btnExecute_Click
cmbBohoor	CurrentIndexChangedEvent	cmbBohoor_IndexChanged
lstShataraat	CurrentItemChangedEvent	LstShataraat_ItemChanged

ومن الأفضل أن تلتزم في تسمية معالج الحدث بالصيغة:

ControlName_EventName

حتى يكون اسم الدالة مميزا في ملف الكود ووظيفتها واضحة بمجرد النظر.

بعد أن تكتب أسماء هذه المعالجات في نافذة الخصائص، انتقل إلى ملف كود التحكم FrmQaseedahController.ring، واكتب كود المعالجات (الدوال).

لقد أضفنا بالفعل خصائص تمثل أسماء الأدوات ليسهل علينا كتابة الكود، وسنضيف الخاصية أخرى اسمها results نضع فيها نتائج تحليل القصيدة التي سنحصل عليها بعد ضغط الزر، لنستخدمها مع باقي الأدوات.. أضف هذا السطر بعد تعريف خصائص الفئة:

results

من المهم أيضا أن نجعل النموذج والأدوات تعرض النصوص من اليمين إلى اليسار لمناسبة اللغة العربية.. يمكن فعل هذا باستدعاء الدالة SetLayoutDirection وإرسال المعامل ١ إليها:

win.SetLayoutDirection(1)

يمكن أن تكتب الجملة السابقة في الدالة Init الخاصة بفئة التحكم، أو كاختصار، يمكنك أن تكتبها بعد تعريف الخصائص مباشرة، لأن الكود الذي يظهر في هذه المنطقة يتم تنفيذه عند إنشاء الفئة مباشرة مثلما يحدث مع الدالة Init.

والآن فلنكتب كود معالجات الأحداث.. وسيكون شبيها جدا بالكود الذي كتبناه لعرض النتائج في نافذة المخرجات، لكن الاختلافات هنا أننا سنأخذ المدخلات من مربع النص، ونعرض النتائج في الأدوات الأخرى التي وضعناها على النافذة.

زر "تحليل القصيدة":

في حث ضغط هذا الزر، سنقرأ نص القصيدة من مربع النص ونرسله إلى الفئة Analyzer لتحليله، وسنضع الناتج في الخاصية Results التي عرّفناها على مستوى فئة التحكم، لنستخدمها في كتابة كود باقي الأدوات.

لاحظ أن الأداة TextEdit تستخدم الوسيلة ToPlainText لقراءة النص المكتوب فيها، على خلاف الأداة LineEdit التي تستخدم الوسيلة GetText لقراءة النص!

```
Func btnExecute_Click( )
   results = new Analyzer(txtQaseedah.ToPlainText( ))
   // باقي الكود
EndFunc
```

بعد تحليل القصيدة، علينا إفراغ الأدوات التي تعرض النتائج من أي نتائج سابقة معروضة فيها، فمن الممكن أن يكتب المستخدم قصيدة أخرى ويضغط الزر:

```
cmbBohoor.Clear( )
lstShataraat.Clear( )
txtDetails.Clear( )
```

والآن سنضع أسماء البحور في القائمة المركّبة (القائمة المنسدلة) cmbBohoor.. لفعل هذا سنستخدم الوسيلة AddItem، وهي تستقبل معاملين، أولهما نص العنصر الذي تريد إضافته، والثاني رقم العمود الذي تريد عرض العنصر فيه.. لدينا هنا عمود واحد فقط، لهذا سنرسل إلى هذا المعامل القيمة صفر.. لكن يجب أن نتأكد أولا أن عدد البحور لا يساوي صفرا، وإلا فسنعرض في القائمة العنصر "لا يوجد بحر":

```
bCount = len(results.QBohor)
if bCount = 0
   cmbBohoor.AddItem("لا يوجد بحر", 0)
else
   for i = bCount to 1 step -1
      bhr = results.QBohor[i]
      bId = bhr[1]
      cmbBohoor.AddItem(BohorInfo[bID][:Name] +
         " (" + 100 * (bhr[2] / results.ShatraatCount) + "%)", 0)
    next
end
```

هذا هو تقريبا نفس الكود الذي استخدمناه لعرض أسماء البحور في نافذة المخرجات، لكننا هنا نضيف أسماء البحور للقائمة بدلا من كتابتها في نافذة المخرجات.

حدث تغير العنصر المحدد في القائمة المنسدلة:

عندما يختار المستخدم بحرا من قائمة البحور، سنستخدم الحدث IndexChanged لعرض الكتابة العروضية لشطرات هذا البحر في القائمة lstShataraat.

لاحظ أن القائمة المنسدلة ستحدد أول عنصر فيها تلقائيا بعد ملئها بأسماء البحور، لهذا سينطلق الحدث IndexChanged تلقائيا بعد أن نملأ القائمة بأسماء البحور.

في معالج هذا الحدث، سنبدأ بإفراغ قائمة التقطيع العروضي ومربع نص المعلومات العروضية من أي نتائج سابقة:

```
Func cmbBohoor_IndexChanged( )
  lstShataraat.Clear( )
  txtDetails.Clear( )
  باقي الكود //
EndFunc
```

بعد هذا سنتأكد أن هناك عنصرا محددا في القائمة.. لاحظ أن أول عنصر في القائمة رقمه صفر، لأن هذه الأدوات ليست مكتوبة بلغة رينج، وإنما قادمة من مكتبة QT.

ويمكننا معرفة رقم العنصر المحدد باستدعاء الوسيلة CurrentIndex (نعم هي دالة وليست خاصية، ويجب أن تضيف قوسين بعد اسمها!.. مكتبة QT تتعامل بهذه الطريقة العجيبة!).. فإن كان رقم العنصر المحدد أقل من صفر، أو عدد البحور صفرا، فنستخدم الأمر Return لمغادرة الدالة في الحال وعدم تنفيذ باقي الكود:

```
If cmbBohoor.CurrentIndex( ) < 0 return End
```

فإذا لم يتحقق الشرط السابق، وواصلنا تنفيذ كود الدالة، فسنكتب حلقة تكرار لملء القائمة lstShataraat بالكتابة العروضية للشطرات:

```
for shInfo in Results.ShataraatInfo
   lstShataraat.AddItem(Join(shInfo[:AroodWrite], " "))
next
```

لاحظ أن قائمة العناصر ListWidget لا تحدد أول عنصر فيها تلقائيا، لهذا سنستدعي الوسيلة SetCurrentRow بأنفسنا، وهي تستقبل معاملين، أولهما رقم

العنصر المراد تحديده (علما بأن أول عنصر رقمه صفر)، والثاني يتحكم في خصائص تحديد عناصر القائمة، وسيكون صفرا دائما في برنامجنا هذا للسماح للمستخدم بتحديد عنصر واحد فقط في القائمة:

```
lstShataraat.SetCurrentRow(0, 0)
```

حدث تغير العنصر المحدد في قائمة الكتابة العروضية:

عندما يختار المستخدم شطرة من القائمة lstShataraat، فسنستخدم الحدث ItemChanged لعرض التفاصيل العروضية لهذه الشطرة في مربع النص txtDetails.. وسنبدأ كالعادة بإفراغ مربع النص من أي نتائج سابقة:

```
Func lstShataraat_ItemChanged( )
   txtDetails.Clear( )
   باقي الكود //
EndFunc
```

بعد هذا سنستخدم الوسيلة CurrentRow لنحصل على رقم العنصر المحدد حاليا في القائمة، وسنجمع عليه ١ لأن عناصر القائمة تبدأ بالرقم صفر، بينما عناصر المصفوفة التي نحتفظ فيها بمعلومات الشطرات تبدأ بالرقم ١.. فإذا كان الرقم الذي حصلنا عليه أصغر من ١ (لا يوجد عنصر محدد) فسنغادر الإجراء في الحال، وإلا فسنحصل على معلومات الشطرة الحالية من نتائج تحليل القصيدة:

```
index = lstShataraat.CurrentRow( ) + 1
If index < 1 return End
shInfo = results.ShataraatInfo[index]
```

الآن أمامنا احتمالان: قائمة البحور تعرض "لا يوجد بحر" أو تعرض بحرا معينا.

لو لم تكن القصيدة موزونة على أي بحر، فسنعرض في مربع النص "هذه الشطرة ليست موزونة على أي بحر" ونعرض رموزها، ونغادر الإجراء:

```
bCount = len(results.QBohor)
if bCount = 0
   txtDetails.SetText("هذه الشطرة غير موزونة على أي بحر" +
      nl + "الرموز : " +
      Join(shInfo[:Romooz].GetOrgHarakaat( ), " ")
   )
   return
End
```

فإذا لم يتحقق الشرط السابق، فسنواصل عرض النتائج على البحر الذي اختاره المستخدم من قائمة البحور، كالتالي:

في البداية سنحصل على رقم البحر الذي نعرض نتائجه، وسنجري هنا نفس العملية الحسابية التي شرحناها عند عرض النتائج في نافذة المخرجات، فنحن نعرض البحور في القائمة بترتيب معكوس لترتيبها في مصفوفة النتائج:

```
i = bCount - cmbBohoor.CurrentIndex( )
bID = results.QBohor[i][1]
```

الجزء الباقي من الكود سيكون شبيها للغاية بكود عرض التفاصيل العروضية في نافذة المخرجات، لكننا هنا نستخدم الوسيلتين SetText و Append لكتابة النص في مربع النص txtDetails، حيث:

- الوسيلة SetText تمحو كل النص الموجود في مربع النص، وتكتب النص المرسل إليها بدلا منه.
- الوسيلة Append لا تمحو النص المكتوب حاليا في مربع النص، ولكن تضيف إليه النص الذي نرسله إليها.

هذا هو الكود:

```
tafs = results.GetTafs(shInfo[:Taqtee3], bID)
if isNull(tafs)
  txtDetails.SetText("هذه الشطرة غير موزونة على بحر " +
  BohorInfo[bID][:Name] + nl + "الرموز : " +
  Join(shInfo[:Romooz].GetOrgHarakaat(), " "))
else
  shTaqtee3 = shInfo[:Romooz].Horoof(tafs[:Lens])
  txtDetails.Append("التقطيع: " + Join(shTaqtee3, " - "))

  romooz = shInfo[:Romooz].Harakaat(tafs[:Lens])
  txtDetails.Append("الرموز : " + Join(romooz, " - "))

  txtDetails.Append("التفعيلات: " + Join(tafs[:Names], " - "))
end
```

وبهذا نكون قد أتممنا عرض النتائج في النافذة.. يمكنك الآن ضغط زر تشغيل البرنامج وتجربته.

إنشاء الملف التنفيذي لبرنامج الشاعر:

نريد الآن أن ننشئ برنامج الشاعر ليعمل من خارج لغة رينج.. واضح طبعا أننا سنستخدم النافذة التي أنشأناها لتكون هي واجهة البرنامج، وكل ما نريد فعله هو إنشاء ملف تنفيذي (.exe) Executable File يبدأ التشغيل بعرض هذه النافذة.. لفعل هذا اتبع الخطوات التالية:

الخطوة الأولى: إنشاء الدالة الرئيسية للبرنامج Main:

في محرر كود رينج، أنشئ ملفا جديدا اسمه AlSha3er.ring، واحفظه في المجلد الرئيسي لبرنامج الشاعر، وأضف فيه الكود التالي:

```
load "FrmQaseedah\FrmQaseedahController.ring"

Func Main( )
  ShowfrmQaseedah( )
EndFunc
```

في هذا الملف أضفنا الدالة الرئيسية Main، وهي الدالة التي يتم استدعاؤها في بداية تشغيل الملفات التنفيذية exe. للبرامج.. في هذه الدالة كتبنا سطرا واحدا يستدعي الدالة ShowfrmQaseedah ومهمتها عرض نافذة البرنامج.. هذه الدالة سنضيفها في فئة التحكم الخاصة بالنافذة frmQaseedah، وهذا هو سبب تحميلنا لفئة المتحكم باستخدام الأمر:

```
load "FrmQaseedah\FrmQaseedahController.ring"
```

علينا إذن إضافة الدالة ShowfrmQaseedah داخل فئة المتحكم.. افتح الملف FrmQaseedahController.ring واكتب الدالة التالية داخل الجزء العام (قبل تعريف الفئة frmQaseedahController):

```
Func ShowfrmQaseedah( )
  new App {
    StyleFusion( )
    open_window(:frmQaseedahController)
    exec( )
  }
EndFunc
```

لاحظ أن كود هذه الدالة هو نفس الكود المكتوب داخل الشرط:

```
if IsMainSourceFile() {
  // .....
}
```

الذي يضيفه مصمم نماذج رينج لملف التحكم تلقائيا.. لهذا يمكنك أن تختصر هذا التكرار وتعدل ذلك الشرط ليصير:

```
if IsMainSourceFile( )
   ShowfrmQaseedah( )
End
```

الخطوة الثانية: إنشاء الملف التنفيذي للبرنامج:

افتح الملف AlSha3er.ring في محرر كود رينج، ومن القائمة الرئيسية "توزيع" Distribute اضغط الأمر:

Ring2Exe (Distribute application – use all runtime – hide console window)

ستظهر نافذة أوامر سوداء تعرض رسالة أن الأمر يتم تنفيذه، ثم بعد قليل سيتم فتح المجلد الذي تم إنشاء ملفات البرنامج فيه (وهو نفس مجلد برنامج الشاعر، لكن الملفات ستضاف في مجلد فرعي اسمه Targets)، وبعد انتهاء العملية ستختفي الشاشة السوداء.

في المجلد Targets ستجد مجلدا باسم نظام التشغيل الذي تعمل عليه مثل windows.. في داخل هذا المجلد ستجد كل الملفات اللازمة لتشغيل البرنامج، ومن بينها الملف التنفيذي AlSha3er.exe.. يمكنك أن تنقره مرتين بالفأرة لفتحه.. ستظهر النافذة الرئيسة للبرنامج، ويمكنك استخدام برنامج الشاعر كما تريد ☺.

وهكذا نكون قد أنجزنا المهمة بحمد الله.

في الفصل التالي سنخطو خطوة أبعد، ونجعل برنامج الشاعر يصل إلى العالمية ويعمل على موقع ويب ☺.. فهيا بنا.

-١٠-
إنشاء موقع ويب لبرنامج الشاعر

في الفصول السابقة من الكتاب أنشأنا برنامج الشاعر واستطعنا عرض نتائجه في شاشة المخرجات، وأيضا صممنا له واجهة مرئية وشغلناه كملف تنفيذي يعمل على سطح المكتب.. وكلا الأمرين كان سهلا، لأننا كتبنا وظيفة البرنامج نفسها مستقلة عن طريقة عرضه، وهذا يعني أننا نستطيع إنشاء واجهات مختلفة لبرنامج الشاعر تستخدم نفس الكود بلا تغيير، وهذا يمتد ليشمل تصميم موقع ويب لبرنامج الشاعر، وكذلك تصميم تطبيق جوال لجعل البرنامج يعمل على الهواتف المحمولة والأجهزة اللوحية!
وفي هذا الفصل، سنرى كيف يمكن استخدام رينج لإنشاء موقع ويب لبرنامج الشاعر بمنتهى السهولة.

تجهيز نسخة الويب Web من المشروع:

تدعم لغة رينج إطار العمل Qt Framework والذى يتيح لنا توزيع نفس المشروع لكل من سطح المكتب Desktop ومواقع الإنترنت Web والأجهزة المحمولة Mobile.. وتستخدم Qt تقنية WebAssembly لتشغيل تطبيقات الويب في المتصفحات، بدون الحاجة إلى جافا سكربت Java Script، وهذا يعني أننا نستطيع تشغيل كود رينج مباشرة في المتصفح من خلال هذه التقنية.
تعال نجرب هذا عمليا:
افتح الملف AlSha3er.ring في محرر كود رينج، ومن القائمة الرئيسية Distribute

اضغط الأمر:

Ring2EXE (Prepare Qt Project – Distribute for Web Browsers using WebAssembly)

وانتظر إلى حين انتهاء العملية وإنتاج ملفات المشروع المطلوبة، والتي ستوضع في المجلد Target/webassembly/qtproject داخل مجلد برنامج الشاعر.

في هذا المجلد ستجد ملفا اسمه project.pro.. سنحتاج لفتح هذا الملف ببرنامج اسمه Qt Creator (الذي سيقوم بإنتاج نسخة الويب من المشروع)، لكن عليك أولا تنزيل هذا البرنامج وإعداده على جهازك.. وستجد تفاصيل تنزيل مكتبة Qt والأدوات المطلوبة لدعم تقنية WebAssembly في الرابط التالي على موقع لغة رينج:

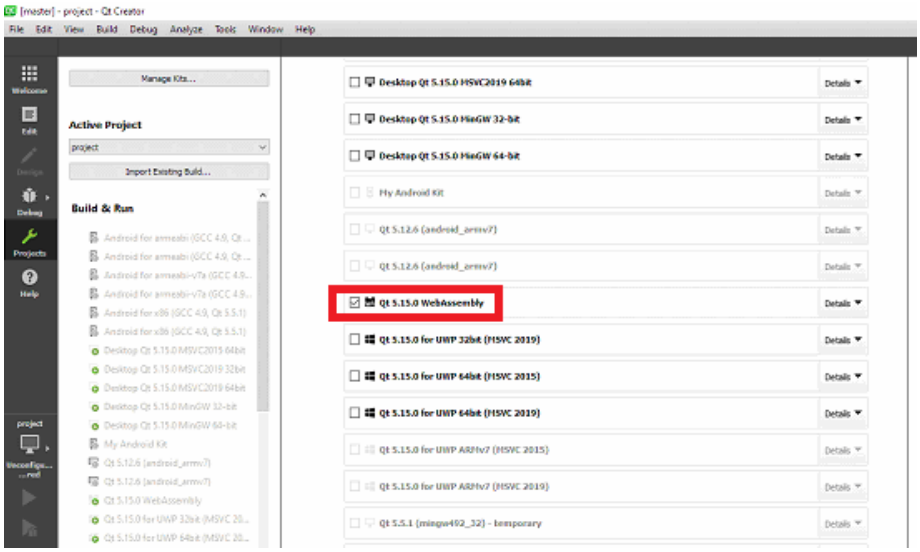
https://ring-lang.github.io/doc1.14/qtwebassembly.html

سنجد في بداية هذه الصفحة رابطا ينقلك إلى موقع QT وفيها تفاصيل كاملة عن كيفية إعداد مكتبة الويب أسمبلي الخاصة بها، وهو أمر لن يكون سهلا على المبتدئين، لهذا لو شق عليك الأمر فتجاهله الآن لأنه يحتاج منك لبعض القراءة ومشاهدة فيدوهات تشرح هذه الخطوات على يوتيوب.. ويمكنك مواصلة قراءة الصفحات القليلة المتبقية من هذا الفصل لأخذ فكرة عن الأمر، وستجد في نهاية الفصل رابطا لموقع نشرنا عليه برنامج الشاعر، حيث يمكنك تجربته مباشرة.

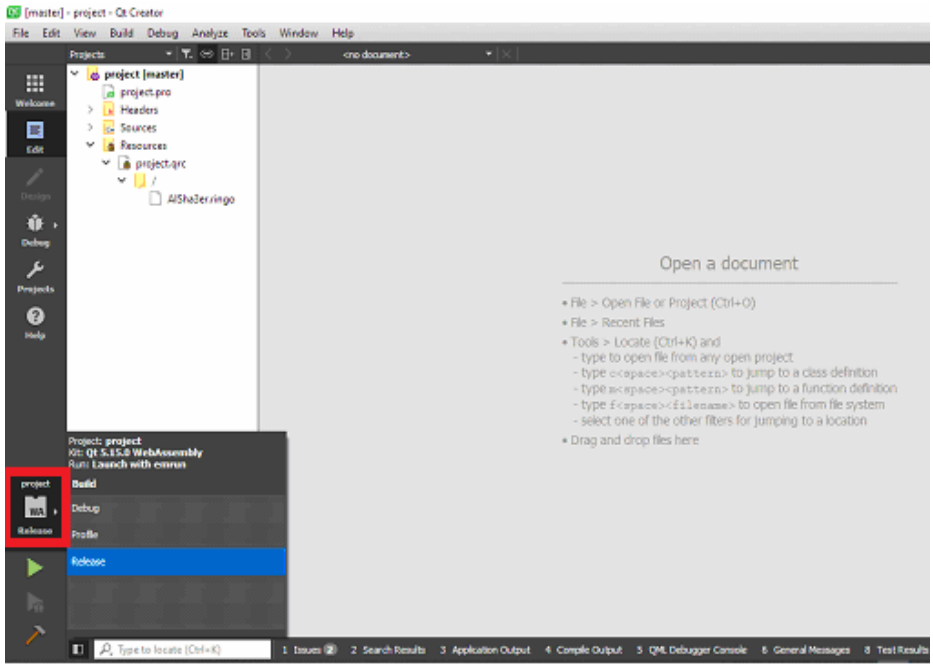
إنشاء تطبيق الويب:

بعد إعداد برنامج Qt Creator على جهازك، يمكنك النقر مرتين بالفأرة على الملف project.pro لفتحه في هذا البرنامج.. ستظهر لك نافذة ضبط خصائص المشروع ومنها يمكنك اختيار المنصة التي نريد بناء المشروع ليعمل عليها.. اختر:

Qt 5.15.0 WebAssembly



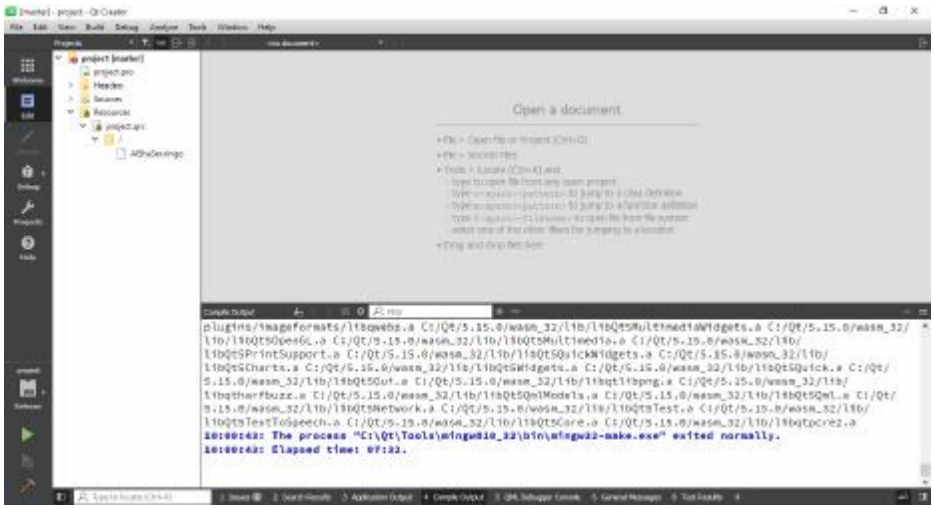
الخطوة التالية هي بناء المشروع.. من الهامش الأيسر (الأسود) اضغط الشريط Project ومن القائمة التي ستظهر اختر الأمر Release، كما في الصورة:



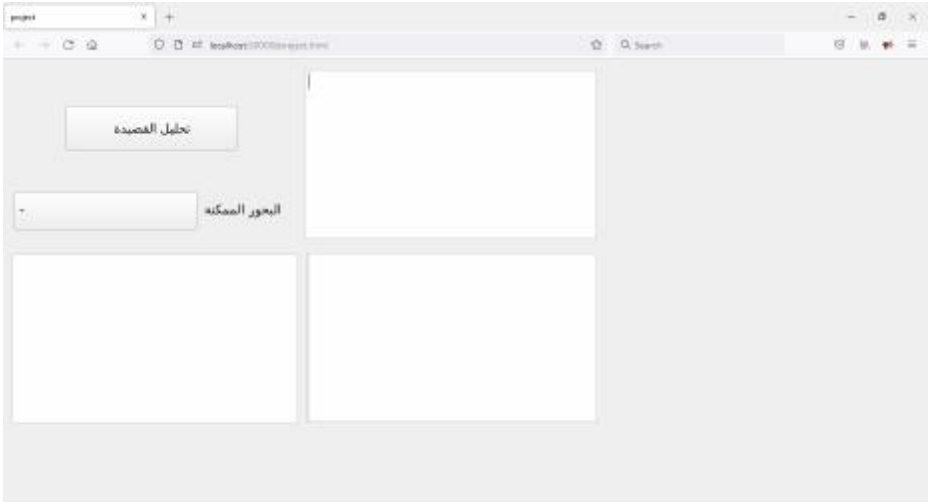
ثم اضغط الشريط Run للتشغيل (أو اضغط Ctrl+R من لوحة المفاتيح)

e

وانتظر لحين الانتهاء من بناء التطبيق.. سيستغرق هذا عدة دقائق في المرة الأولى.



وبمجرد انتهاء عملية البناء سيظهر التطبيق الخاص ببرنامج الشاعر تلقائيا داخل المتصفح Web Browser الافتراضي الذي تستخدمه على جهازك، وستلاحظ أنها نفس الواجهة التي أنشأناها لتعمل على سطح المكتب، حيث يمكنك كتابة أبيات من الشعر وضغط زر تحليل القصيدة لعرض معلومات عروضية عنها!



نشر تطبيق الويب:

ينتج عن عملية البناء مجلد جديد اسمه webassembly ستجده داخل المجلد target، وهو يحتوى على ملفات كثيرة لكن ما يهمنا منها هو الملفات الخمسة التالية:

Project.html
Project.js
Project.wasm
qtloader.js
qtlogo.svg

أما باقي الملفات فهي ملفات مؤقتة نتجت أثناء عملية البناء ولا نحتاج لها عند نشر تطبيق الشاعر على موقع الويب.

هذه الملفات الخمسة هي ملفات HTML و JavaScript و WebAssembly، ومهمتها أن تعمل على جهاز العميل Client Side في المتصفح الخاص به، وكل المطلوب منك أن تنشر هذه الملفات على أي خادم ويب Web Server (ستحتاج بالطبع لشراء استضافة وعنوان لموقعك.. هذه تفاصيل لا تعنينا هنا في هذا الكتاب) ولا يشترط أن يدعم خادم الويب لغة برمجة معينة.. المهم أن يكون متصفح الإنترنت الذي يستخدم لفتح الموقع حديثا ومتطورا ويدعم الويب أسمبلي، وهذا أمر سهل فأشهر

المتصفحات مثل Edge و Chrome وغيرهما تدعم هذه التقنية حاليا على مختلف أنظمة التشغيل الشهيرة.

لاحظ أنك تستطيع إجراء تعديلات على الملف project.html مثل إضافة صورة للمجلد وعرضها في صفحة الويب أثناء تحميل البرنامج، وكتابة اسم التطبيق "الشاعر" وأي تعديلات أخرى تريدها.

و يمكنك تجربة برنامج الشاعر على موقع الويب من خلال الرابط التالي (سأضعه أيضا في مجلد أمثلة الكتاب لتضغطه مباشرة من هناك):

https://ring-lang.github.io/web/poet/project.html

مع ملاحظة أن فتح الرابط لأول مرة قد يستغرق بعض الوقت لأن تقنية ويب أسمبلي تحتاج لتنزيل الملفات الخاصة بالمشروع إلى جهاز المستخدم في أول مرة.. كما أن هناك احتمالا ألا يعمل هذا الموقع على أجهزة ويندوز ٣٢ بت، لهذا من الأفضل أن تفتحه على نظام تشغيل يعمل بنظام ٦٤ بت.. ربما يكون هذا عيبا في تقنية QT، لكني أطمئنك أن رينج تدعم تصميم مواقع الويب باستخدام تقنيات أخرى لن تواجه معها هذه المشكلة.. لكن شرح هذا خارج عن نطاق هذا الكتاب، ويمكنك الاستزادة عنه من خلال الروابط الموجودة في المجلد "مصادر تعلم رينج" المرفق بأمثلة الكتاب.

وهكذا تكون رحلتنا في هذا الكتاب قد انتهت، لكني أرجو أن تكون رحلتك في عالم البرمجة قد بدأت للتو، وأن تستمتع فيها بصحبة رينج.

تم بحمد الله

محمد حمدي غانم

٢٠٢١/٩/٢

عن الكاتب

- مهندس محمد حمدي غانم.
- من مواليد محافظة دمياط ١٩٧٧.
- خريج هندسة الاتصالات، جامعة القاهرة.
- عمل مبرمجا وكاتبا تقنيا.
- أنشأ لغة Small Visual Basic وهي نخة مطورة من لغة MS Small Basic لتعليم البرمجة للأطفال والمبتدئين، ونشر سلسلة درو مرئية لشرحها على يوتيوب.
- له كتب متخصصة في البرمجة تشرح لغتي VB.NET و C# وهي:
  - فيجيوال بيزيك وسي شارب: طريقك المختصر للانتقال بين اللغتين.
  - المبرمج الصغير: تعلم البرمجة بفيجيوال بيزيك دوت نت.
  - المدخل العملي السريع إلى: سي شارب.
  - المدخل العملي السريع إلى: فيجيوال بيزيك دوت نت.
  - من الصفر إلى الاحتراف: سي شارب.
  - من الصفر إلى الاحتراف: فيجيوال بيزيك دوت نت.
  - من الصفر إلى الاحتراف: برمجة إطار العمل.
  - من الصفر إلى الاحتراف: برمجة نماذج الويندوز.
  - من الصفر إلى الاحتراف: برمجة قواعد البيانات في سي شارب.
  - من الصفر إلى الاحتراف: برمجة قواعد البيانات في VB.NET.
  - أساسيات Wpf لمبرمجي سي شارب.
  - أساسيات Wpf لمبرمجي فيجوال بيزيك دوت نت.
- له إصدارات أدبية أخرى منها:
  - مجرّد طريقة للتفكير، مسرحية (العدد ١٦ من "آفاق المسرح" من إصدارات قصور الثقافة، ٢٠٠٠).
  - انتهاك حدود اللحظة، ديوان شعر فصيح (مكتبة دار المعرفة، ٢٠١٠).
  - دلال الورد، ديوان شعر فصيح (قصر ثقافة دمياط، ٢٠١٣).
- بريد الالكتروني:

msvbnet@hotmail.com

الفهـــــــرس

مقدمة

-١-
تنزيل وإعداد لغة رينج

-٢-
التعامل مع النصوص العربية في رينج

-٣-
تقطيع النص إلى سطور وكلمات

-٤-
كلمات خاصة إملائيا

-٥-
الكتابة العروضية

-٦-
الرموز الصوتية

-٧-
التفعيلات والبحور

-٨-
تحليل النتائج

-٩-
تصميم واجهة مرئية للبرنامج

-١٠-
إنشاء موقع ويب للبرنامج