اللغات البرمجية للحاسبات الالكترونية

- عصام احمد حبيب

جامعة توهوكو - سنداي - اليابان

- شبيك خليك عمر

(مراجعة لغويه)

مركز حاسبة جامعة الموصل

تعرف لغة البرمجة على انها مجموعة من المعلمات او الارشادات instructions تعطى الى الحاسبة لحل مشكلة معينة . وعندما يكتب البرنامج بلغة الحاسبة machine language فعلى المبرمج ان يأخذ في حسابه معرفة سجلات الحاسبة registres والذاكرة وعناوين مواقعها memory addresses .

وبناء لغة جديدة للحاسبة ، يعني بناء نموذج جديد يسهل من عملية البرمجة بالنسبة للمبرمج وينقذه من الوقوع في قيود مكونات الحاسبة المادية ، وتلك هي وظيفة لغات البرمجة العالية المستوى . وهناك المئات بل الالاف من لغات البرمجة ، لكل لغة خصائص معينة ووظائف معينة ، كما ان لكل لغة تركيب مجرد وتركيب معنوي خاص بها distinctive grammer and syntax.

يتناول هذا المقال لغات البرمجة المختلفة ذات الاستعمال الرياضي والفيزياوي ، ويشرح صفاتها وخصائصها المشتركة وصولا الى ايجاد التطبيقات المناسبة لكل منها ، كما يتناول المقال ايضا اهمية ونوعية تصاميم اللغات المتوازية لما لها من تأثير في زيادة القدرة الحاسبية والسرعة في الانجاز .

في الشكل رقم 1 نماذج برامج مكتوبة بلغة LOGO التي ظهرت في معهد MIT في الولايات المتحدة الاميركية عام 1960. ومن خصائص هذه اللغة قدرتها على السيطرة على اداة تسمى السلحفاة turtule تكون بمثابة جهاز آلي robot يتحرك وفقا لايعازات البرنامج امامـا وخلفـا" ويمينـا" وشمالا" ويدور رافعا او خافضا لقلم مثبت فيها ، فيترك آثار حركته ، كما تترك السلحفاة آثار مسيرها ، وليس بالضرورة ان تكون السلحفاة جهازا آليا ذا حجم فيزياوي بل يمكن . وهو الاغلب حاليا . ان تظهر كمثلث صغير على شاشة التلفزيون تتحرك وفقا لايعازات البرنامج على مساحة الشاشة تاركة خطوطا او نقاطا مضيئة . البرنامج بلغة LOGO اذن ، هو مجموعة من الايعازات التي توجه الى السلحفاة حيث تقوم بتنفيذها .

وكما يلاحظ من الشكل رقم 1 فأن من هذه الايعازات هو الايعاز وضع القلم pendown ورفع القلم penup والتحرك امامـا Forward 50 او يمينـا right 144 ... الخ والرقم المجاور للايعاز تمثل عدد نقاط التقدم او درجة الانحراف .كما نلاحظ ان رسم نجمة يتم برسم خطوط باستعمال Forward 50 والدوران يمينـا (بأتجاه عقارب الساعة) 144 درجة باستعمال right 144 .

وتعتبر لغة LOGO وسيلة كفوءة للمبرمج في تفادي الاعادة والتكرار ، مما يرفع من كفاءة برنامجها ويقلل من حجم الذاكرة المحجوز ، اذ بدلا من استعمال الجملة right 144 Forward 50 خمس مرات لرسم النجمة ، فأن المبرمج يستطيع ان يكتب الجمل الخمس في جملة واحدة هكذا :

Repeat 5 (Forward 50 right 144)

واذا اراد المبرمج رسم نجمة ذات تسعة اذرع ، بطول 80 نقطة للذراع الواحد ، فأن البرنامج يكون :

Repeat 9 (Forward 80 right 160)

ان التغيير فقط هو بعدد مرات الاعادة ، وطول الذراع ، ودرجة الدوران يمينا . فأذا اراد المبرمج ان يكون برنامجها عاما ، فأن هذه الاعداد 9 و 80 و 160 يمكن ان تحول الى متغيرات تستدعي عبر روتين Procedure لها اسم يحددها المبرمج ومن خلال الايعاز To . فاذا كانت وظيفة هذا الروتين رسم نجمة ، ولها الاسم Star على سبيل المثال، فان الايعاز to star سيستدعي الروتين star ويقوم بتنفيذها ونلاحظ في الشكل 1 ان المتغيرين size و points يمثلان ادلة وسيطة لامرار قيمتي طول الذراع ودرجة الانحراف الى داخل الروتين star . وتستخدم ":" للدلالة على الاسماء المتغيرات في لغة LOGO .

ومن خواص القوة التي تتمتع بها لغة LOGO اقترابها من اللغات الطبيعية natural languages من حيث تكوين الكلمات والرموز والجمل واستعمال مفاتيح Keywords ذات معنى ثابت بحيث تظهر الجملة وكأنها تتكون من افعال واسماء كما الحال في اللغات الطبيعية ، فالجمل في اللغات البرمجية تصنف الى جمل اعلانية declarations او جمل تعبيرية statements . ونجد ذلك واضحا في LOGO من خلال برامج الشكل 1 ، اذ ان star يمثل (اسم) العملية ، . و isize و points : المتغيران اللازمان لانجاز العملية star ، اما الجملة التعبيرية ، فهي محتويات الروتين star والتي تمثل في الواقع خوارزمية الحل ولهذا نجد ان repeat (فعل) في الروتين star ، والرقم الذي يتبعها هو اسم الفعل اما محتويات القوسين فتمثل المفعول به .

مقارنة بين ست لغات برمجية :

هناك لغات برمجية شاع استعمالها وطغى على مئات اللغات البرمجية الاخرى ، وسنحاول هنا استخدام ست لغات برمجية ذائعة الصيت لحل مشكلة معينة هي : BASIC و PASCAL و COBOL و FORTH و APL و LISP . فضلا عن لغة سابقة هي لغة الحاسبة machine language وهي – بالطبع – لغة دنيا .

والمشكلة الحسابية المختارة كركيزة للمقارنة بين هذه اللغات هي : ايجاد مجموع الاعداد الفردية من مجموعة من الاعداد الصحيحة . وهذه المشكلة يمكن برمجتها باستعمال اية لغة برمجية ، فضلا عن ان حل مثل هذه المشكلة يستلزم وجود (شرط) محدد ، وهو كون العدد فردي ، كما يستلزم قدرة اللغة البرمجية على التنفيذ التكراري iterative execution .

BASIC

عرفت لغة BASIC عام 1965 في جامعة Dartmouth الاميركية ، واصبحت شائعة الاستعمال كلغة مجاذبة interactive language ، واشتهر استعمالها في برمجة الحاسبات المايكروية .

تبدأ كل جملة من جمل البرنامج المكتوب بهذه اللغة برقم (الشكل 2) ، اذ أن السيطرة على تنفيذ البرنامج يعتمد على ارقام الجمل لتسهيل عملية الانتقال داخل حدود البرنامج . ويستعمل الايعاز LET لتحديد قيم المتغيرات في جمل الاحلال الحسابي ، كما يستعمل الايعازان FOR و NEXT لتحديد حجم الدارة LOOP التي يتكرر فيها التنفيذ على عدد محدد من الجمل المحصورة بينهما .

PASCAL

ظهرت هذه اللغة عام 1970 على يدي Niklaus Wirth من المعهد التكنولوجي الفدرالي السويسري في زوريخ . وتختلف هذه اللغة عن لغة BASIC في أنها يجب الاعلان عن نوعية كل متغير مستعمل في البرنامج ، كأن يكون متغيرا صحيحا او منظومة صحيحة ...الخ كما تستعمل الكلمات بدل الارقام للوصول الى اية جملة او مقطع داخل البرنامج اثناء الانتقال من جملة الى اخرى (الشكل 3) .

استخدمت لغة PASCAL بشكل خاص لانتاج لغات جديدة اخرى . وقد ظهرت في السنوات الاخيرة لغة باسم (MODULA-2) اعتمدت بالاساس على لغة PASCAL واضافت اليها بأن جعلت البرنامج من لغة (MODULA-2) عبارة عن مجموعة من الوحدات البرمجية المستقلة التي لايعتمد بعضها على البعض الاخر . وقد استخدمت لغة (MODUIA -2) بشكل خاص في عمليات المحاكاة وتصاميم الطائرات وفي البرامج المساعدة في التصميم CAD .

ADA

انتجت لغة ADA لحساب وزارة الدفاع الاميركية ، معتمدة على لغة PASCAL وقد اخذت شهرتها في الاونة الاخيرة ، واصبحت

من اللغات المتميزة والقوية على الرغم من ظهور العديد من التنقيحات على أصل اللغة . وتستخدم بشكل خاص في دوائر الدفاع الامريكية لمحاكاة تصاميم الطائرات ومدى تحملها . وهذه اللغة القابلية على تكوين تراكيب جديدة للغات اخرى . وساترك الحديث عن لغة ADA الى مجال آخر .

COBOL

جاءت لغة COBOL عام 1960 نتيجة لمؤتمر عقد بين مصنعي ومستخدمي الحاسبات الالكترونية . وهذه اللغة تاريخ طويل من الاستعمال في الدوائر الحكومية والبنوك وشركات التأمين ..الخ ويتركز استعمالها في معالجة الكميات الكبيرة من البيانات . ويتكون برنامج COBOL من اربعة اقسام اساسية هي : التعريف والمحيط والبيانات والعمليات . واذا كانت معظم اللغات تتمثل بصيغ رياضية او منطقية ، فأن لغة COBOL تتمثل باستعمال جمل واضحة المعنى شبيه تماما بجمل اللغة الانكليزية . وعلى الرغم من ان قراءة برنامج بلغة COBOL تكون واضحة الا أن كتابة البرامج بهذه اللغة يقود الى الملل احيانا (الشكل 4) .

FORTH

اعلن Charles Moore من موسسة National Radio Astronomy Observatory عن لغة جديدة باسم FORTH عام 1970 ، الهدف منها الاستفادة منها في اعمال السيطرة والتحكم وبالاخص في السيطرة على التلسكوبات . وقد اصبحت هذه اللغة اكثر توسعا وذات انطلاقات اخرى جديدة . كما انها تستخدم بكفاءة في الحاسبات الميني والحاسبات المايكروية لاقتصادها في استعمال الذاكرة (الشكل 5) .

وبالمقارنة مع لغة COBOL ، فأن لغة FORTH صعبة القراءة لوجود العديد من الاشارات التي تقوم مقام المفاتيح واهم خواص لغة FORTH امتلاكها لمكدس stack وهو مساحة من الذاكرة ، واذ تترتب الاعداد بعضها فوق بعض في المكدس ، فأن آخر عدد يدخل المكدس هو اول عدد يخرج منه عند الحاجة . ولايعلن عن نوعية المتغيرات في لغة FORTH ، كما ان كل الحسابات والعمليات تجري على المكدس ، وتعاد النتيجة الى اعلى المكدس ايضا .

APL

للغة APL تركيب اكثر اختصارا من FORTH . وقد عرفت عام 1961 من كتاب صدر عن شركة IBM لمؤلفة Kennth Iverson ، على

انها اداة لتمثيل المشكلات في الرياضيات التطبيقية ، ثم اجريت عليها تحسينات واضافات جديدة تعزز من استعمالاتها في مجال الحاسبات .

واللغة APL صفة متميزة في تعاملها مع المنظومات العددية ومع الاعداد المفردة ، فضلا عن أن ايعازا واحدا يكفي للقيام بالعمليات المتشابهة على مختلف قيم المنظومات (الشكل 6) .

LISP

تعد لغة LISP التي اعلنها John McCartgy من معهد MIT الاميركي اواخر عام 1950 ، من ابسط اللغات التي سبق ذكرها ، اذ انها نوع واحد من العبارات يطلق عليها اسم استدعاء الدالة Function Call ، والاهم من ذلك في لغة LISP هو أن القيمة المعادة من استدعاء الدالة قد تكون استدعاء آخر لدالة اخرى ، وتشتهر لغة LISP في انها اللغة البارزة في الذكاء الاصطناعي ، كما انها اللغة الواعدة لحاسبات الجيل القادم .

جاءت تسمية هذه اللغة اختصارا لـ LIST Processing ، اذ ان كلا من البرنامج والبيانات مركب على شكل قوائم Lists

كما ان البرنامج بهذه اللغة يتم على شكل دارات متكررة iterative loops ، ومن ثم يقع على عاتق المبرمجين عند بناء برامجهم اختيار الاسلوب الذي يستغل خاصية الاستدعاء الذاتي recursive technique التي تشتهر بها لغة LISP ، وخاصية الاستدعاء الذاتي هي ان الدالة تستدعي نفسها عددا من المرات لحين تحقق شرط معين ، وعندئذ تبدأ عملية التعويض لتعيد الدالة قيمة للدالة الاقدم وصولا الى الناتج النهائي .

يتكون البرنامج (الشكل 7) من قوائم من الاعداد ، فاذا كانت القائمة فارغة unll فان استدعاء الدالة يعيد القيمة صفر ، أما اذا كانت اول قيمة في القائمة فردية فسوف تضاف الى المجموع ، ثم تستدعي الدالة نفسها مرة اخرى لتكرار العملية نفسها بعد حذف القيمة المستعملة في كل مرة ، الى ان نصل الى نهاية القائمة فيتوقف التنفيذ ونحصل على مجموع الاعداد الفردية .

لغة الحاسبة

من المعروف ان الحاسبة لاتتعامل مع البرنامج – في نهاية المطاف – الا بشكله الثنائي اي ان لغة وحدة المعالجة المركزية والسبلات وغيرها هي اللغة التي تعتمد النظام

الثنائي binary systen الذي يتكون من الصفر والواحد . وان شفرة الحاسب تتكون من تسلسل من الاحاد والاصفار لتمثيل ايعاز instruction او بيانات data او عنوان address .ويمكن كتابة البرامج مباشرة الى الحاسب باستعمال هذه اللغة الثنائية (الشكل 8) الا أن عملية الكتابة بالاحاد والاصفار عملية مملة وكثيرة الاخطاء .

وفي اوائل الخمسينات جرت المحاولات الاولى لبناء البرامج التجميعية assenblers لنقل كتابة البرامج من اللغة الثنائية (الاحاد والاصفار) الى مجموعة من المختصرات التي تخفف من استعمال الاحاد والاصفار ، وبذلك ظهرت الايعازات الاسمية التي تساعد على التذكر مثل ADD كايعاز اضافة و SUB كايعاز طرح و MOVE كايعاز تحويل ... الخ وباستعمال هذه الاسماء المختصرة mnenonics تخلص المبرمج من تذكر التشكيلة اللازمة لكل ايعاز من الاحاد والاصفار ، وبذلك عرفت اللغات التجميعية asserbly languages ولاشك ان تعلم اللغة التجميعية لحاسب ما يعطي المقدرة للمبرمج على الاستعمال المباشر لكل اجزاء الحاسب ، كما يضمن لبرنامجه كفاءة عالية في الاداء وسرعة التنفيذ ، ويطلق على اللغات التجميعية باللغات الدنيا low level langnages لانها قريبة

من لغة الحاسبة ، وعملية التحويل الى لغة الحاسبة الثنائية عملية تحويل مباشر ، فكل ايعاز تجميعي يتحول الى مكافئه الثنائي مباشرة عبر جداول خاصة لهذا الغرض .اوقد كانت عملية التحويل في البداية يدوية ، ثم اصبحت تتم عبر برامج تجميعية بطيعة الية .

ان حل اي مشكلة حسابية أو غير حسابية انما يتم اولا بحلها بشكل عام من خلال خوارزمية تحدد الشكل العام للحل ، ثم يقع على عاتق المبرمج اختيار اللغة المناسبة لبرمجة الخوارزمية . فأن كانت اللغة عالية المستوى ، فأن برمجة مشكلة فيزياوية مثل : القوة = الكتلة x التعجيل ، لاتبدو صعبة . أما استعمال لغة دنيا ، فأن المبرمج يجب ان يكون على معرفة اجزاء الحاسبة وسجلاتها كي يستطيع برمجة الخوارزمية .

وتحتاج البرامج المكتوبة بلغات عالية المستوى الى برامج خاصة تقوم بتحويل او ترجمة هذه البرامج الى لغة الحاسبة وهناك نوعان من هذه البرامج : النوع الاول ، البرامج المترجمة compilars او المترجمات ، والنوع الثاني ، البرامج المفسرة interpreters او المفسارات .

يقوم (المترجم) بتحويل البرنامج (ككل) الى برنامج بلغة الحاسبة ، وتنتفي الحاجة بعد انجاز الترجمة الى النسخة الاصلية من البرنامج المكتوب بلغة عالية المستوى ، وتمر عملية الترجمة في ثلاثة اطوار هي :

1- تكوين نص البرنامج باستخدام Text Editor

2- تحويل النص الى نص بشفرة الحاسبة

3- انجاز صيغة شفرة الحاسبة

أما المفسار فأنه يقوم بتحويل البرنامج سطرا سطرا في كل مرة الى لغة الحاسبة وتنفيذها ، لذا فأن الفرق بين (المترجم) و (المفسار) شبيه بالفرق بين ترجمة نص مكتوب على ورقة ، وترجمة حديث مباشر فقرة فقرة . ففي الحالة الاولى نقرأ النص كاملا ثم نترجمه كاملا . أما في الحالة الثانية فأننا نسمع ثم نترجم فقرة بعد اخرى . وبالتالي فأننا نحتاج الى المفسار باستمرار عند تنفيذ البرنامج ، بينما لانحتاج الى بقاء المترجم في ذاكرة الحاسبة عند تنفيذ البرنامج بعد انجاز مرحلة الترجمة .

من اللغات العالية المستوى التي تستخدم المترجمات لغات FORTRAN و COBOL و PASCAL و C . ومن اللغات التي

تستخدم المفسارات لغات LOGO و FORTH و APL . وهنـــــاك من اللغات من يظهر باستخدام كلا الاسلوبين كلغتي BASIC و LISP .

ويفضل استخدام (المترجم) على (المفسار) في كون الاول سريع الانجاز في التنفيذ . كما يفضل استخدام (المفسار) في مراحل بناء البرامج واختبارها ، لان (المفسار) يشير الـــى الخطأ مباشرة ، كما يقوم بالتنفيذ مباشرة لكل جملة ، ممـــا يتيح للمبرمج تحديد اخطائها بسهولة .

ولتبسيط عمل (المترجم) للقارئ ، فأن عملية الترجمة compilation تتم – كما مـــر سابقا – عبر ثلاثة اطـــــوار :

الطـور الاول : التحليل القاموسي lexical analysis ، وفيـــها يقوم المترجم بتشخيص الرموز المختلفـة في نص البرنامج وتصنيفها كمفاتيـــــح (Key words) وقيم رقمية constants ومتغيـــــرات variables (الشكل 9) .

الطور الثاني : يقوم المترجم بايجاد العلاقات في التركيـــب بين المفاتيح وقواعد اللغة . فعلى سبيـــــل المثال ، يكون الايعاز IF مرتبطا في بعـــــض اللغات بـ THEN .

الطور الثالث : هو طور بناء البرنامج بشفرة الحاسبة machine code .

وهناك قسم من المترجمات ذات طور رابع هو طور التماثل optimization اذ يتم خلال هذا الطور تنقيح البرنامج وصولا الى تحسين كفاءة اداءها .

وتجري محاولات جادة لتحسين تصاميم المترجمات ، وبناء قواعد لغوية بسيطة ومتكاملة ، والاستفادة من خاصية الاستدعاء الذاتي recursive technigue لتكوين كل العبارات الممكنة من تلك القواعد . وهناك صيغة جديدة في هذا المجال هي صيغة مترجم - المترجم compiblr- compiler ، الهدف منها بناء لغات جديدة باستخدام مترجمات لهذا الغرض ، مما يسهل على المبرمج استحداث لغة جديدة بعد تقديم المواصفات اللازمة لها الى مترجم - المترجم .

كما تهدف البحوث حاليا الى بناء لغة برمجة عامة وشاملة universal ، وعلى الرغم من ان لغة PL/1 التي بنتها شركة IBM عام 1965 كانت بهذا الاتجاه ، الا انها حققت نجاحات جزئية محدودة ، وان كثيرا من هذه الجهود لم تحقق المطلوب بعد لصعوبات فنية وتعليمية .

في مقابل اللغات البرمجية الروتينية procedural غير الوصفية prescriptive التي نوقشت قبل قليل ، هناك لغات برمجية غير روتينية nonprocedural ووصفية descriptive اصبحت لها اهمية متزايدة واستعمالات كثيرة .اذ ان المستعمل لهذه اللغات لايتدخل في صياغة برامجها الموجودة مسبقا ولا في خوارزمياتها ، بل يقوم بتزويد هذه اللغات بالبيانات اللازمة واختيار البرنامج المطلوب لتنفيذها والحصول على النتائج . من هذه اللغات التي اشتهر استعمالها مع الحاسبات الشخصية VisicCalc و Multiplan و Lotus 1-2-3 فلغة VisicCalc هي اول لغة ظهرت للاستعمال في الحاسبات الصغيرة ، واصبحت ذات فائدة في استعمالات الدوائر والاعمال . وتحوي هذه اللغة برامج معينة تظهر على شكل جداول مرسومة تحوي حقولا تنتظر البيانات من قبل المستعمل ، وتسمى هذه الجداول بـ Spread Sheets وبذلك يتخلص المبرمج تماما من التفكير بكتابة اي برنامج . وتتخصص برامج VisicCalc بالامور المالية بشكل عام .

اما لغة Multiplan فتعمل على الحاسبات المايكروية باحدى نظم التشغيل CP/M او MSDOS او Xenix او UNIX. وبرامج

Multiplan مخزونة على اقراص مرنة floppy disks ، وعند استعمالها فأن البرنامج المطلوب ينتقل من هذه الاقراص الى الذاكرة .

وتتكون Lotus 1-2-3 التي انتجت عام 1983 من ثلاثة مكونات هي :

1- الاوراق الالكترونية Spreadsheets .
2- زخرفة الاعمال Business Graphic .
3- ادارة قواعد البيانات Database Management .

واذا كانت هذه المكونات منفصلة عن بعضها في اللغة السابقة ، فأنها تتكامل مع بعضها في Lotus فضلا عن وجود برامج استخدامات اخرى تعمل على نظام تشغيل MS-DOS وتساعد المبرمج في انجاز مهمته .

وتأتي لغة Symphony لتحمل تحويرات واضافات جديدة الى Lotus ، الا أنها تحتاج الى مالايقل عن 320 كيلو بايت ذاكرات RAM ، ويعود السبب الى وجود مكونات جديدة في هذه اللغة لم تكن موجودة في Lotus مثل الاتصالات Communications والبرمجة الماكروية Macroprogramming ومعالج النصوص Word processor واستمارات بيانات Database/forms filter

يتم استعمالها بسهولة وباستخدام مفتاح واحد من مفاتيح لوحة الحاسبة .

ومن لغات البرمجة الحديثة لغة Prolog القريبة من اللغات الطبيعية ، والتي تعد من لغات الذكاء الاصطناعي ، اذ تحوي اللغة هذه صيغا ثابتة بل علاقات تربط بين الاشياء قيما لتكوين البرنامج (الشكل 10) .

تتكون لغة Prolog من معلنات declaration فقط وليس لها جملا statements كما هو الحال في اللغات المعروفة . لذا فأن العلاقة (Product hight with area) تكون بمثابة المساواة بين Area= Hight x width من دون تحديد قيمة الارتفاع او العرض ولانوعية المساحة المراد حسابها .

وثمة تطويرات جديدة في لغات البرمجة ذات اهمية كبيرة قادت الى بناء لغات الاشياء الموجهة object-oriented language: اذ تظهر باستعمالها الحاسبة كانها قد جزئت الى مجموعة من الاشياء التي يمكن لها ان تعنون بشكل منفصل فضلا عن ان هذه الاشياء يمكن لها أن تتصل مع بعضها عبر رسائل فيما بينها (الشكل 11) .

وقد قدمت لغة Simula 67 المشتقة من لغة Algol 60 دلالات الى هذه الاشياء الموجهة لم تلق في البداية اهتماما كبيرا

لحين طور لغة Smalltalk عام 1970 من مركز Xerox Palo Research Centre . وتتكون اللغات التي تعتمد هذه الدلالات على هياكل البيانات والخوارزمية ، ومن ثم يبقى على المبرمج ان يتعامل مع الحاسبة بهذه اللغة في ادخال البيانات واستخراج النتائج ، وما عدا ذلك يبدو الامر كصندوق مقفل . وكما ان لكثير من الحيوانات مثل البطريق والجمل والثعبان وغيرها طرق مختلفة في المشي ، كذلك فأن لغات الاشياء الموجه تتطلب من المبرمج معلومات متكونة من اعداد واعداد مركبة ومنظومات لانجاز ما هو مطلوب من قبل الحاسبة كاجراء عمليات اضافة مثلا .

وتمارس الزخارف البيانية graphics دورا كبيرا في الكثير من مثل هذه اللغات ، وبالاخص فيما يتعلق بالالعاب الالكترونية .

وثمة اتجاه آخر لتصميم لغات البرمجة يعتمد على الحساب المتوازي الموزع distributed paxallel computation ونعني به ان الحاسبات الضخمة قد تتكون من مئة وحدة معالجة ، وبالتالي فأن توزيع العمل الواحد لتنفيذه على مئة وحدة معالجة سيزيد من سرعة التنفيذ بالنسبة نفسها فيما لو نفذ البرنامج على حاسبة اعتيادية ذات وحدة معالجة واحدة ، مع

ملاحظة طبيعة اللغة المستعملة وقدراتها على التنفيذ المتوازي .

وتعرف بعض لغات البرمجة المتوازية اسلوبا في تنفيذ المهمات يعتمد على توضيح العمليات التي يمكن ان تنجز بالتوزاي مسبقا ، وذلك مثل لغة Occam التي انتجتها شركة Britfish Semicandrctor Manufacturer . وهناك لغات برمجة متوازية اخرى تعتمد على مترجمها لاكتشاف الحسابات المتوازية مثل لغة COMPEL وهي مختصر لـ compute Paralcl . وقد استفدت من هذه اللغة لبناء نظام جديد لانجاز المشكلات التي تتطلب سرعة عالية بشكل متواز . فالبرنامج بلغة COMPEL مثلا يتكون من جمل احلال assigment statement التي ليس من الضروري انجازها بالتسلسل الذي كتبت به كما هو الحال في الحاسبات الاعتيادية . لذا يقوم المترجم بانشاء تنظيم جديد لهذه الجمل لتنفيذها بشكل متواز . وقد اعتمدت في بحثي على طريقة تكاد تكون جديدة لتمثيل البرنامج قبل تنفيذه بشكل نستطيع به ايجاد الاماكن والجمل التي نستطيع جمعها لتنفيذها بالتوازي .

ان تصميم الحاسبات الالكترونية وحتى الضوئية لها مشكلتان اساسيتان في التركيب هما :

- كيفية جعل العوامل المكونة للمشكلة متصلة فيما بينها لنقل الاوامر وانجاز البرنامج .
- كيفية تقسيم البرنامج الى اجزاء تنجز بوحدات معالجة عديدة .

وعلى هذا الاساس تقسم الحاسبات المتوازية الى :

1- التوازي الخشن Coarse grain parallelism : وفيه يستخدم عدد قليل من وحدات المعالجة ، ذات سرعة عالية ، لايتجاوز عدد وحدات المعالجة 16 وحدة .

2- التوازي المتوسط Medium grain parallelism : ويستخدم فيه من 32 الى 1000 وحدة معالجة ، كل واحدة منها تمثل حاسبة شخصية . مثال ذلك حاسبة Hybercube Architecture التي تستعمل Intels Cosmic Cube الذي يحتوي 256 وحدة معالجة ولان هذه الوحدات مرتبة على شكل معين ، لهذا فأن وحدات المعالجة الواقعة على اطراف المكعبات تأخذ وقتا طويلا - نسبيا - لاجل الاتصال فيما بينها مما يجعل هذه الحاسبة بطيئة عند وجود اتصالات كثيرة بين اطراف المكعبات . ولهذا فأن البرنامج يحتاج الى خوارزمية خاصة للتخلص من الاتصالات البطيئة .

3- التوازي الدقيق Fine grain parallelism : ويحوي من 1024 الى 256000 وحدة معالجة تعمل في آن واحـــــد . لذا فأن المبرمج يجب ان يفكر عند كتابة برنامجا في كيفية جعـــل حساباته تنجز باستخدام اكبر عدد من وحدات المعالجـــــة بصورة آنية كي يختصر في زمن تنفيذ برنامجا ، ان وحـــدات المعالجة هذا يجب ان تكون لها القدرة على التنـــــوع في الارتباط حسب التطبيقات والتي تضمن زيادة السرعـــــة في التنفيذ . واقوم حاليا بتكوين تصاميم لها القابلية على تكوين تركيبات مختلفة وعديدة ، تحت اسم :

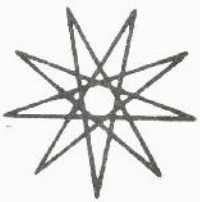
Reconfigured and Dynamic Architecture لها القدرة على تكوين تركيب جديد واتصالات جديدة حسب التطبيق المستخدم. ومن الامثلة على هذه التصاميم التصميم المسمـــــــــى Connection Machine الذي صمم في MIT عام 1983 واستعمـــل 65536 وحدة معالجة . ومن التطبيقات المفيدة لهـــــــذه الحاسبة حاسبات الرؤيـــة vision computers والتحكـــــم بالمرور الجوي Air Traffic Control واجهزة الانسان الالـــي التي تتطلب استجابة سريعة مقاربة للاستجابة الحقيقيـــــة real time فضلا عن حدوث تغييرات معينة تجعل من الصعب على الحاسبات ذات التركيبات الثابتة انجازها بشكل سريع .

ولاشك ان التقدم في بناء الرقاقات الالكترونية ، وبالاخص رقاقات التكامل الكبير جدا (VLSI) سيزيد من امكانيات بناء تصاميم جديدة ذات كفاءة عالية ، غير ان مشكلة بناء خوارزميات كفوءة تواكب هذا التطور التقني تبقى المشكلة التي تتطلب بحوثا كثيرة ، لازالت بطيئة لحد الان .

```
Pendown
forward 50 right 144
forward 50 right 144
forward 50 right 144
forward 50 right 144
forward 50 right 144
forward 50 right 144
penup
```

```
pendown
repeat 5(forward 50 right 144)
penup
```

```
pendown
repeat 9(forward 80 right 160)
penup
```



```
to star:size:points
   repeat:points(forward: size right720/: points)
   penup
```

star 80 5

```
to star :size :points
   if :points   2
   (pendown
   repeat:points(forward size right720/: points)
   penupl)
```

star 80 9

شكل -1- برامج بلغة LOGO

```
100 DIM T(100)
200 READ N
300 FOR I=1 to N
400 READ T(I)
500 NEXT I
600 GOSUB 1100
700 PRINTS
800 GOTO 2000
900 DATA 4
1000 DATA 23,34,7,9
1100 REM MAKES THE SUM OF THE ODD ELEMENTS IN ARRAY T(1...)
1200 LET S=0
1300 FOR I=1 TO N
1400 IF NOT ODD (T(I) THEN GO TO 1600
1500 LET S=S+T(I)
1600 NEXT I
1700 RETURN
2000 END
```

شكل -2- برنامج بلغة BASIC

يمثل برنامجا في BASIC ، ويظهر الروتين في الحروف الاكثر سوادا ، ولكن نلاحظ بأن ليس له اسم يشير اليه وانما يشار بواسطة رقم العبارة (1100 GOSUP) ان البرنامج في لغة BASIC ليس له ادلة وسيطة (paramters) وانما تحدد القيم بالمتغيرات العامة (Global variables) وبذلك يستطيع الروتين الفرعي استخدامها . المتغير في البرنامج ليس من الضروري ان يعلن (declare) مالم يكن منظومة array .

في هذا البرنامج نلاحظ ان DIM T(100) تحدد قيم المنظومة بـ 100 قيمة اما (الدارة) فأنها تتحدد بواسطة العبارة (FOR-NEXT) اما العبارة الشرطية فانها تتحدد بواسطة (IF-THEN) .

```
Program SUMOdd Numbers;
type TermIndex=1.....100;
     Term array =array(TermIndex)of Integer;

VAR  myTerms: TermArray;

Function sumOdds(n: TermIndex;Terms: TermArray):interger;
        Var i:=TermIndex;
            Sum:integer;
        begin
          Sum:=0;
            fori;=1 to n do
               if odd(terms [i])then
                     Sum:=Sum+terms [i] ;
        SumOdds:=Sum;
        end;
begin
    my Terms[1]:=23;myTerms [2]:=34;myTerms[3]:=7;myTerms[4]:=9;
        Write Ln (Sum Odds(4,myTerms));
end.
```

شكل -3- برنامج بلغة PASCAL

برنامج بلغة PASCAL لجمع الاعداد الفردية في منظومة صحيحة . الدالة التي تقوم بجمع الاعداد الفردية باسم SUMODDS . لاحظ ضرورة الاعلان عن نوعية المنظومة والدالة والمتغيرات . لاحظ ايضا أن عملية ايجاد المجموع يتم عبر جملة الدارة (FOR-DO) كما ان اختبار العدد يتم باستعمال جملة (IF-THEN) . أما المنظومة الصحيحة فهي (my Terms) .

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 NUMERIC-VARIABLE USAGE IS COMPUTATIONAL.
     02 TERMS PICTURE 9999 OCCURS 100 TIMES INDEXED BY I.

     02 N-PICTURE 999.
     02 SUM PICTURE 999999.
     02 HALF-TERM PICTURE 9999.
     RMDR PICTURE 9.

PROCEDURE DIVISION.
EXAMPLE.
     MOVE 23 TO TERMS (1)
     MOVE 34 TO TERMS (2)
     MOVE 7  TO TERMS (3)
     MOVE 9  TO TERMS (4)
     MOVE 4  TO N.
     PERFORM SUM-ODD.
SUM-ODDS.
  MOVE 0 TO SUM.
  PERFORM CONSIDER-ONE-TERM VARYING I FROM 1 BY 1
UNTIL I  N.
CONSIDER-ONE-TERM.
  DIVIDE 2 INTO TERMS(I) GIVING HALF-TERM REMAINDER RMDR.
  IF RMDR IS EQUAL TO 1;ADD TERMS (I) TO SUM.
```

شكل -4- برنامج بلغة COBOL

البرنامج لايجاد مجموع الاعداد الفردية باستخدام SUM-ODDS والتي تستدعي عملية اخرى تسمى (CONSIDER-ONE- TERM) . ان هذه اللغة لاتحتوي على ادلة وسيطة Paramters لهذا تحدد قيم المنظومة مسبقا . ان الدارة تتمثل في PERFORM-VARYING واختبار القيم بالجملة IF .

```
:SUMODDS
 0 SWAP0
   DO
    SWAPDUP2MOD
        IF+
             ELSE DROP
              THEN
 ;        LOOP
 23 34 7 9 4 SUMODDS
```

WORD	STACK	COMMENT
23	23	
34	23 34	
7	23 34 7	
9	23 34 7 9	
SUMODDS	23 34 7 9 4	callSUOMDDS
0	23 34 7 9 4 0	
SWAP	23 34 7 9 0 4	
0	23 34 7 9 0 4 0	
DO	23 34 7 9 0	Remove loop control values
SWAP	23 34 7 0 9 9	
DUP	23 34 7 0 9 9 2	
2	23 34 7 0 9 1	
MOD	23 34 7 0 9	
IF	23 34 7 0 9	TOS=1;do IF to ELSE
+	23 34 7 9	
LESE	23 34 7 9	skip past THEN
DROP	23 34 7 9	skipped
THEN	23 34 7 9	skipped
LOOP	23 34 7 9	Return to DO
DO	23 34 7 9	
SWAP	23 34 9 7	
DUP	23 34 9 7 7	
2	23 34 9 7 2	
MOD	23 34 9 1	
IF	23 34 9 7	TOS=1;do IF to ELSE
+	23 34 16	

WORD	STACK	COMMENT
ELSE	23 34 16	skip past then
DROP	23 34 16	skipped
THEN	23 34 16	skipped
LOOP	23 34 16	return to DO
DO	23 34 16	
SWAP	23 16 34	
DUP	23 16 34 34 2	
2	23 16 34 34	
MOD	23 16 34 0	
IF	23 16 34	TOS=0;do ELSE to THEN
+	23 16 34	skipped
ELSE	23 16 34	
DROP	23 16	
THEN	23 16	
LOOP	23 16	Return to DO
DO	23 16	
SWAP	16 23	
DUP	16 23 23	
2	16 23 23	
MOD	16 23 1	
IF	16 23	TOS= 1;do IF to ELSE.
+	39	
ELSE	39	skip past THEN.
DROP	39	skipped.
THEN	39	skipped.
LOOP	39	no more iterations.
		Return from SUMONS.
	EMPTY STACK	Print the result.

شكل -5- برنامج بلغة FORTH

يقوم البرنامج بلغة FORTH بجمع الاعداد الفرديـــــة في منظومة تترتب عناصرها في مكدس . ويبدأ التعامل مع الاعـــداد

العليا من المكدس . يستخدم الايعاز SWAP لتبديل اول قيمتين مع بعضهما في المكدس اما الايعاز DUP فيكرر القيمة العليا في المكدس ويخزنها . اما الايعاز DROP فيحذف القيمة العليا في المكدس . العاملان + و MOD يقومان بتبديل الناتج بعد انجاز العملية محل القيمتين التي اجريت عليهما العملية . ان الايعاز DO يقوم بازالة القيمتين من المكدس ، وتكرار ذلك العدد المطلوب من المرات . اما الايعاز IF فينجز العملية عندما تكون القيمة في اعلى المكدس لاتساوي صفرا والا تم انجاز مابين ELSE و THEN .

```
SUM  -SUMODDSTERMS
SUM  -+ /(2/ TERMS)/TERMS

SUMODDS 23 34 7 9
   TERMS -          23  34   79  Initial value assignment,
(2 | TERMS) -        1   0    1     Array of remainders.

(2 |TERMS/TERMS)  -  23                Compression of two arrays.
+/(2 |TERMS)/TERMS  - 23 + 7    + 9   Reduction by addition.
             SUM  -  39                 Assignment of result
```

شكل -6- برنامج في لغة APL

هذا البرنامج يحسب مجموع الاعداد الفردية في منظومة واحدة وتعرف المنظومة TERMS في سطر واحد من دون الحاجة الى اعطاء عدد القيم التي تتكون منها المنظومة عبارات APL تنجز من اليمين الى اليسار ما عدا اذا وجدت اقواس فانها تغير اتجاه التحليل .

في هذا المثال العبارة (2|TERMS) تنجز اولا والتي تحسب الباقي بعد قسمة كل قيمة موجودة في المنظومة على 2 وتكوين صف بنفس الحجم الذي يملكه (TERMS) . ان الرمز (/) يمثل نوعين من العمليات ، ففي العبارة (2 |TERMS)/ TERMS," / ") تمثل ضغط (Compression) العاملين (Operators) التي تكون منظومة جديدة والذي فيه كل قيمة من (TERMS) تظهر فقط اذا كانت المفردات المطبقة في (2|TERMS) ليست صفرا . في الرمز ("+/, ""/") يمثل تقليل (reduction) العاملين والتي تقلل المنظومة الى رقم مفرد بواسطة ادخال(+) بين كل زوج من القيم .

```
DEFUN SUMODDS
  (LAMBDA (TERMS)
   (COND
    ((NULL TERMS ))0)
      ((ODD (CARTERMS)) (PLUS(CARTERMS)(SUMODDS(CDRTERMS))))
        (T(SUMODDS((CDR TERMS ))))))
(SUMODDS, (23 34 7 9 ))

(SUMODDS, ( 23 34 7 9 ))
        = ( PLUS 23(SUMODDS, (34 7 9 )))
        = ( PLUS 23(SUMODDS,   ( 7 9 )))
          = (PLUS 23 (PLUS 7 (PLUS 9(SUMODDS,()))))
          = (PLUS 23 (PLUS 7(PLUS 9 0)))
      = (PLUS 23 (PLUS 7 9 ))
       = (PLUS 23 16 )
=39
```

شكل -7- برنامج في لغة LISP

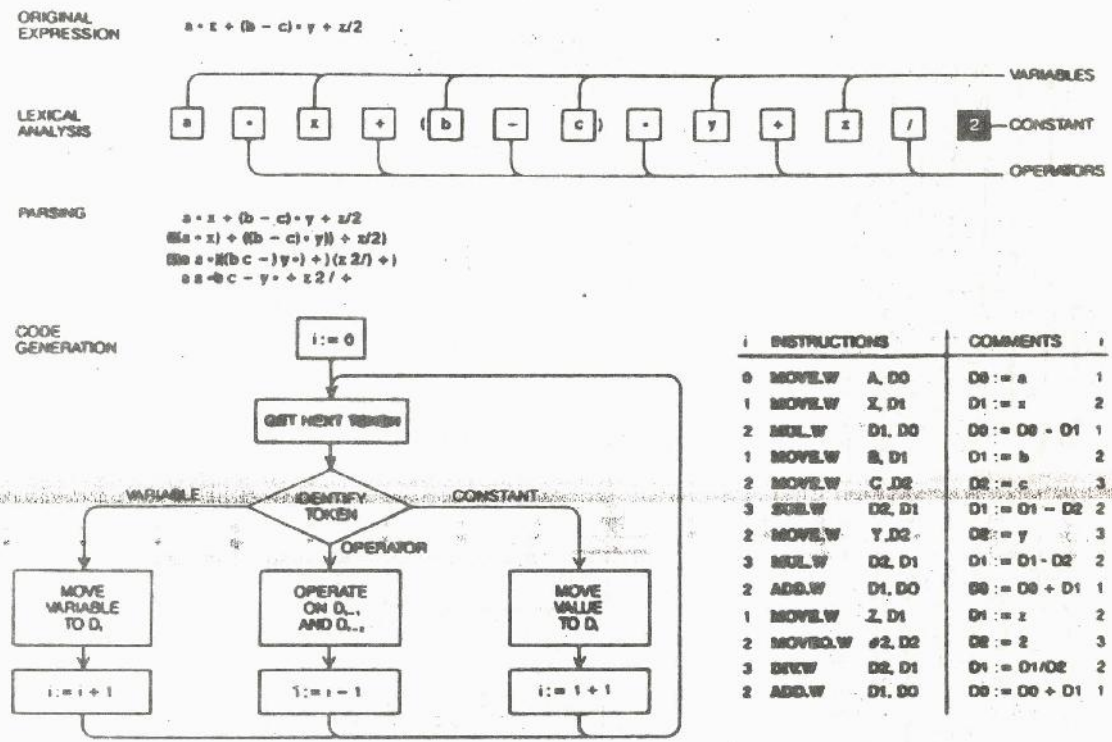
يقوم البرنامج بحساب مجموع الاعداد الفردية لمنظومة صحيحة . اذ تستدعي الدالة SUMODDS نفسها ذاتيا . تمثل CAR القيمة الاولى في القائمة , بينما تمثل CDR بقية القيم . ويستعمل المصطلح DEFUN لتعرف الدالة , أما LAMBDA فلتعريف الادلة الوسيطة في هذا البرنامج هناك دليل وسيط واحد هو TERM. COND وظيفته تحليل CAR , فأن كان الناتج حقيقيا true فان CDR سوف تحلل وفقا لخوارزمية البرنامج , واذا كان الناتج زائفا false فأن COND يحول الى القائمة التالية . ثمة احتمالات ثلاثة تواجه تنفيذ البرنامج : اما ان تكون TERMS قائمة فارغة فيعيد SUMODDS الصفر , او حقيقيا فيضيف القيمة الى المجموع SUMODDS , أو زائفا فتهمل القيمة .

AssEMBLY CODE

MACHINE CODE	LABLS	INSTRUCTIONS	COMMENTS
00100100 01011111	SUMODDS	MOVE.L (A7)+ A2	Pop return address from the stack into A2
00100010 01011111		MOVE.L (A7)+A1	Pop address of first term into A1
00110010 00011111		MOVE.W (A7)+D2	Pop n into D1
01000010 01000010		CLR.W D2	Assign a value of to the sum in D2
01001110 11111010 00000000 00001110		JMP COUNT	Jump to the end of the loop to test if n=0
00001000 00101001 00000000 00000000 000000000 00000001	LOOP	BTST 0,1(A1)	if the term addressed by A1 is even .....
01100111 00000010		BEQ.S NEXT	...then go to NEXT
11010100 01010001		ADD.W (A1),D2	otherwise add the term to the sum in D2
01010100 01001001	NEXT	ADDQ.W #2,A1	Set A1 to the address of the next term
01010001 11001001 11111111 11110010	COUNT	DBF D1,LOOP	Decrement D1: unless it is-1,go to LOOP
00111110 10000010		MOVE.W D2-(A7)	Push the sum from D2 onto the stack
01001110 11010010		JMP (A2)	Go to the return address

شكل -8- برنامج بلغة الحاسة

يقوم البرنامج بحساب مجموع الاعداد الفردية باستخدام لغة الحاسبة . وتبدو الحاجة هنا الى معرفة مكونات الحاسبة لكي تتم برمجتها بالشكل الصحيح . ورموز هذا البرنامج تعتمد على وحدة معالجة مايكروبة نوع (Motorola 68000) . اما الخوارزمية فهي بصيغة مشابهة للبرنامج المكتوب بلغة PASCAL وبصيغة افضل . ويلاحظ في البرنامج استخدام المكدس stack والعناوين addresses . كما نلاحظ استخدام اللغة التجميعية بايعازاتها المساعدة على التذكر . كما يظهر البرنامج (اقص اليسار) بشكله الثنائي ، وهو الشكل الذي تتعامل معه الحاسبة آخر المطاف .



i	INSTRUCTIONS		COMMENTS	i
0	MOVE.W	A, D0	D0 := a	1
1	MOVE.W	X, D1	D1 := x	2
2	MUL.W	D1, D0	D0 := D0 * D1	1
1	MOVE.W	B, D1	D1 := b	2
2	MOVE.W	C ,D2	D2 := c	3
3	SUB.W	D2, D1	D1 := D1 - D2	2
2	MOVE.W	Y ,D2	D2 := y	3
3	MUL.W	D2, D1	D1 := D1 * D2	2
2	ADD.W	D1, D0	D0 := D0 + D1	1
1	MOVE.W	Z, D1	D1 := z	2
2	MOVEQ.W	#2, D2	D2 := 2	3
3	DIV.W	D2, D1	D1 := D1/D2	2
2	ADD.W	D1, D0	D0 := D0 + D1	1

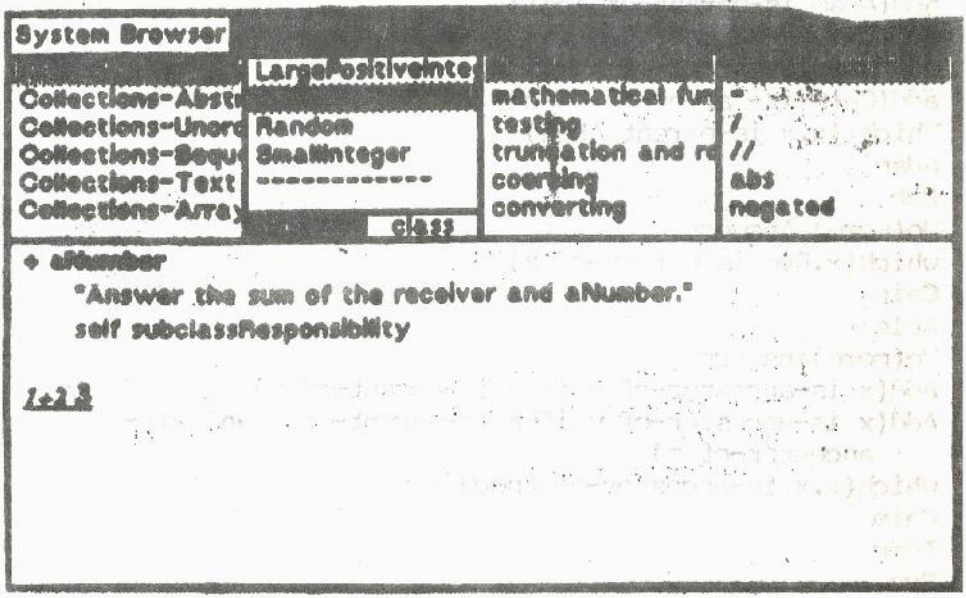
شكل -9- مثال لعمل المترجم PASCAL

يقوم المترجم بتحليل التعبير الحسابي مارا بثلاثة اطوار . في الطور الاول يتم تحليل العلامات والرموز . وفي الطور الثاني يتم معرفة معاني العلاقات الحسابية والمعاملات والعمليات الحسابية ووفق سلم تنفيذ العمليات الحسابية . ومن ثم يتحول التعبير الحسابي الى صيغة جديدة وفق احدى الصيغ المستعملة مثل التدوين البولوني ، كما يتحول التعبير في الطور الثالث الى صيغته النهائية .

```
add(Adam is-parent-of Cain)
add(Adam is-parent-of Abel)
add(Eve is-parent-of Cain)
add(Eve is-parent-of Abel)
add(Cain is-parent-of Enoch)
Which (x.x is-parent Abel)
Adam
Eve
No(more) Answers
which(x.Eve is parent-of x)
Cain
Able
No(more)answers
Add(x is-ancester-of y if x is-parent-of y)
Add(x is-ancester-of y if z is-parent-of y and xis-
    ancestor-of z)
which(x.x is-ancestor-of Enoch )
Cain
Adam
Eve
No(more)answers
which(x.Adam is-ancestor-of x)
Cain
Abel
Enoch
No(more)answers
```

شكل -10- برنامج في لغة PROLOG

هذه اللغة ليس لها عبارات ولكنها متكونة بشكل كامل من بيانات (declarations) ان برنامجا بهذه اللغة يعطي معلمات غير واضحة (Not explicit instructions) لانجاز العمليات وانما تحدد فقط العلاقات وتكون الاستنتاجات فيما بينها مبينه على تلك العلاقات , هذا البرنامج مكون بواسطة (Micro-prolog) ان اول خمسة بيانات تحدد العلاقة بين الولد والوالدين فالنظام ممكن ان يجيب عن الاسئلة التي تتعلق بتلك الحقيقة , هناك قاعدتان تقودان الى منطق الاستنتاج (Logical Inference) واللتان استخدمتا لتحديد العلاقة بين السلف بدلالة الوالدين , النظام يمكن ان يطبق هذه القواعد لايجاد كل الاسلاف او الاخلاف للافراد .



شكل -11- برنامج Smaltalk

يمثل هذا البرنامج جمع رقم 1 الى رقم 2 والناتج هو 3 بالحقيقة ان الرقم 1 هو ليس رقما وانما يمثل شيئا محددا بعض الاشياء التي تحددت برقم واحد ،وهو الشيء نفسه بالنسبة الى رقم 2 ، لهذا فاننا قمنا بدمج شيء 1 مع شيء 2 لتكوين شيء 3 ، يجب ان نلاحظ بأن المبرمج لايستطيع ان يرى ما بداخل شيء 1 ولا في باقي الاشياء الاخرى .