المختصر في "بناء المترجمات"

د.أيمن حمارشه

البرامج، المفسرات والمحولات

Programs, Interpreters and Translators

لغات البرمجة Programming Languages: تتكون لغة البرمجة من مجموعة من الرموز التي تستخدم في وصف العمليات التي يطلب المستخدم من الحاسوب إنجازها. وهي الطريقة الوحيدة التي تتم بواسطتها إعطاء الأوامر للحاسوب ليقوم بإنجاز المهمات والوظائف المحددة. هذا هو ما جعل الاهتمام بلغات البرمجة وتطويرها وإنشاء الجديد منها يأخذ حيزا كبيرا من وقت المبرمجين ومن اهتمامهم.

إن البرامج المكتوبة بلغة التجميع Assembly أو بلغة عالية المستوى High Level Language لا يمكن تنفيذها من قبل الحاسوب إلا بعد تحويلها إلى اللغة الوحيدة التي "يفهمها" الحاسوب وهي لغة الآلة Machine Language. عملية التحويل هذه تسمى ترجمة Compiling والبرنامج الذي يقوم بالترجمة يسمى مترجم Compiler.

معالجات اللغة Language Processors: بناءً على ما سبق يمكن القول أن المترجم هو عبارة عن برنامج يمكنه قراءة البرنامج المكتوب بإحدى اللغات عالية المستوى التي تسمى لغة المصدر Source Language والذي يسمى البرنامج المصدري Source Program وترجمته إلى برنامج مكافئ بلغة الآلة يسمى برنامج الهدف Target Program.

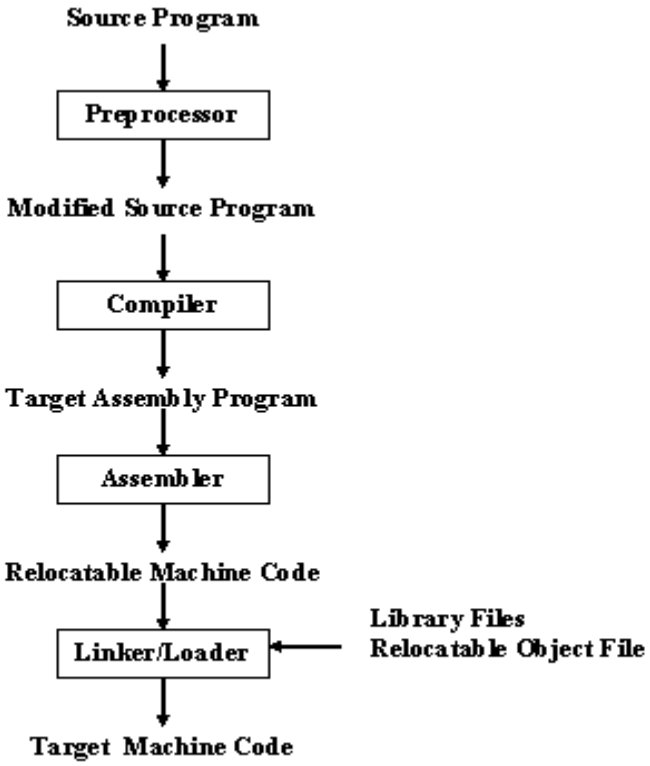
ومن الجدير بالذكر أن مهمة المترجم لا تقتصر فقط على تحويل اللغة وإنما يقوم أيضا خلال عملية الترجمة باكتشاف الأخطاء التي قد يحتوي عليها البرنامج المصدري. عندما يصبح برنامج الهدف على الشكل الذي يكون فيه قابلا للتنفيذ عندها فقط يتم تحويل المدخلات إلى مخرجات ونتائج أي تتم عملية التنفيذ.

هناك نوع شائع آخر من معالجات اللغة يسمى المفسر Interpreter الذي يتلخص عمله في أنه بدلا من إنتاج برنامج الهدف أولا ثم إدخال البيانات اللازمة للتنفيذ بعد ذلك، يقوم المفسر بإدخال البيانات في نفس الوقت الذي تجري فيه ترجمة البرنامج أي بشكل متزامن.

برنامج الهدف الذي يتم إنتاجه بواسطة المترجم يكون عادة أسرع في تحويل المدخلات إلى مخرجات من البرنامج الذي ينتجه المفسر في حين المفسر أفضل في تشخيص الأخطاء لأنه يقوم بتنفيذ البرنامج جملة تلو الأخرى.

في بعض الأحيان تتم عملية الترجمة على شكل خليط من عمل المترجم والمفسر معا كما هو الحال في معالج لغة JAVA الذي يجمع بين الترجمة والتفسير حيث تتم أولا ترجمة البرنامج المكتوب بلغة JAVA إلى برنامج وسيط يسمى bytecodes والذي يتم لاحقا تفسيره بواسطة الآلة الافتراضية Virtual Machine. الفائدة من هذا الترتيب هو أن bytecodes التي تمت ترجمتها على آلة معينة يمكن تفسيرها على آلة أخرى كما يمكن ذلك من خلال شبكة. بالإضافة إلى المترجم فإن هناك مجموعة من البرامج لا بد من توفرها حتى تتم عملية إنتاج برنامج الهدف القابل للتنفيذ.

البرنامج المصدري يمكن تقسيمه إلى وحدات برمجية تسمى Modules يتم تخزينها في ملفات منفصلة وعملية تجميع أجزاء البرنامج المصدري توكل في بعض الأحيان إلى برنامج منفصل يسمى Preprocessor أو المعالج الأولي.



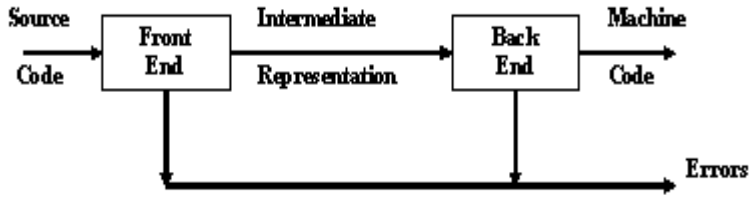
مخطط نظام معالجة اللغة

البرامج الكبيرة عادة تتم ترجمتها في أجزاء وفي مرحلة لاحقة يجب وصل هذه الأجزاء وربطها مع بعضها ومع ملفات المكتبات وملفات هدف أخرى مكونة شفرة برنامج واحد قابل للتنفيذ. البرنامج الذي يقوم الربط يسمى الرابط Linker وهو المسؤول عن أيجاد العناوين في الذاكرة لتخزين هذا البرنامج في حين يقوم برنامج آخر هو Loader أو المُحَمِّل بوضع هذا البرنامج التنفيذي بجميع أجزائه في الذاكرة لتنفيذه.

النموذج التحليلي-التركيبي للترجمة

Analysis-Synthesis Model of Translation

عند تحليل المترجمات نجد أن هناك تباينا كبيرا في تركيبها وبنيتها Structure ولكن مع هذا فإن هناك بنية عامة تشترك فيها أغلب المترجمات وأكثر نماذج الترجمة شيوعا هو النموذج التحليلي-التركيبي المبين أدناه



يبين هذا النموذج أن هناك جزأين رئيسيين للترجمة هما:

1. الجزء التحليلي Analysis Part : ويسمى النهاية الأمامية للمترجم Front End ويقوم هذا الجزء بتحليل أو تفكيك البرنامج المصدري إلى أجزاء رئيسية كما يقوم بفرض بنية قواعدية لهذه الأجزاء. هذا يعني أنه يتم في هذا الجزء تحديد فيما إذا كان البرنامج المصدري قد تم تشكيله قواعديا على نحو خاطئ أو إذا كانت مفرداته غير صحيحة ولا تنتمي إلى لغة البرمجة المستخدمة، وفي هذه الحالة يقوم بإرسال رسالة تفيد بوجود الخطأ Error Message ليقوم المستخدم باتخاذ الإجراءات اللازمة لتصحيح هذا الخطأ.

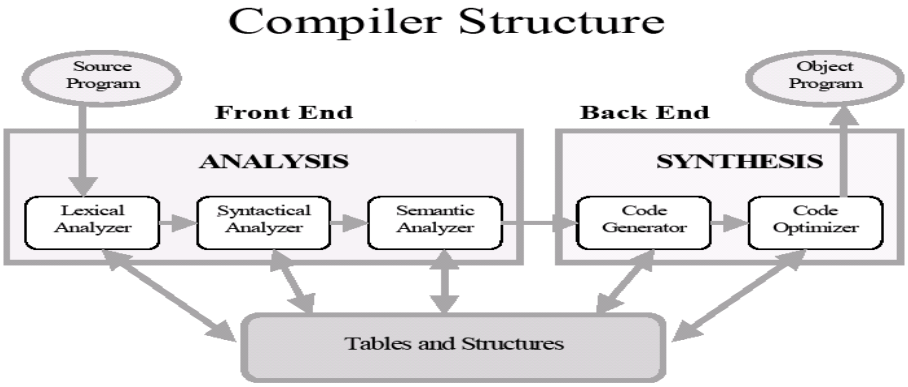
في هذا الجزء أيضا يتم جمع المعلومات التي يحتاجها المترجم خلال عملية الترجمة عن البرنامج المصدري ويتم تخزينها في هيكل بيانات Data Structure يسمى جدول الرموز Symbol Table. خلال عملية التحليل يتم تحديد العمليات التي يتضمنها البرنامج المصدري ويتم تخزينها في بنية هرمية تسمى الشجرة.

في نهاية عمل الجزء التحليلي يتم إنشاء تمثيل وسيط Intermediate Representation للبرنامج المصدري.

2. الجزء التركيبي Synthesis Part: يسمى النهاية الخلفية للمترجم Back End ويقوم هذا الجزء بإنشاء برنامج الهدف المطلوب مستخدما التمثيل الوسيط الناتج عن الجزء السابق ومن المعلومات الموجودة في جدول الرموز.

عند النظر إلى الترجمة بشكل أكثر عمقا سوف نجد أنها عبارة عن عملية متعددة المراحل بحيث يقوم كل جزء من أجزاء المترجم بتنفيذ الجزء الخاص به وينتج تمثيل معين في نهاية المرحلة الخاصة به ليقوم بعدها الجزء التالي بوظيفته منتجا هو الآخر تمثيلا معينا للبرنامج المصدري يختلف عن التمثيل الناتج عن المرحلة السابقة وهكذا حتى تنتهي جميع المراحل ونحصل على البرنامج التنفيذي المطلوب.

جدول الرموز Symbol Table والذي يستخدم لتخزين معلومات كاملة عن البرنامج المصدري يتم استخدامه في جميع مراحل الترجمة .



مخطط هيكلي للمترجم

في الجزء التحليلي Analysis Part من الترجمة تتم عدة عمليات هي:

- تحليل المفردات Lexical Analysis: وهي المرحلة التي يتم فيها تحليل مفردات البرنامج والنظر إليها كشيء متميز عن النحو وتركيب الجمل ويقوم بذلك برنامج يسمى Lexical Analyzer.
- التحليل النحوي Syntax Analysis: في هذه المرحلة يتم تحليل البرنامج والبحث في تركيب وبناء الجمل وترتيب الكلمات أو مفردات اللغة بشكل صحيح في الجملة ويقوم بهذا برنامج Syntactical Analyzer.
- تحليل الألفاظ Semantic Analysis: هذه المرحلة هي مرحلة تحليل الجمل من ناحية معاني ودلالات الألفاظ المختلفة في جمل وعبارات البرنامج وينفذ ذلك برنامج Semantic Analyzer.

الجزء التركيبي Synthesis Part من الترجمة يحتوي على المراحل التالية:

- إنتاج الشفرة الوسيطة Intermediate Code Generation: خلال ترجمة البرنامج المصدري إلى شفرة الهدف النهائية يقوم المترجم بإنشاء تمثيل وسيط واحد أو أكثر وهذا التمثيل له أشكال مختلفة. على سبيل المثال الشجرة النحوية Syntax Tree التي تستخدم خلال التحليل النحوي وتحليل الألفاظ هي أحد أشكال هذا التمثيل الوسيط.

بعد انتهاء مراحل التحليل النحوي وتحليل الألفاظ للبرنامج المصدري الكثير من المترجمات تقوم بإنتاج تمثيل وسيط منخفض المستوى يشبه إلى حد كبير شفرة الآلة Code Machine وهذا التمثيل يمكن اعتباره برنامج تنفيذي أو شفرة لآلة مجردة وليست مخصصة لآلة محددة. هذا التمثيل الوسيط له ميزتان مهمتان أن تكون الشفرة الوسيطة سهلة الإنتاج والميزة الأخرى أن يكون سهلا ترجمتها إلى الآلة الهدف.

- **تحسين الشفرة Code Optimization: بشكل عام يقصد بتحسين الشفرة إدخال ميزات جديدة عليها بحيث يتم الوصول إلى أفضل شكل للشفرة الهدف تتميز بسرعة التنفيذ من خلال تصغير حجم هذه الشفرة مثلا.**

قضايا مهمة في تصميم المترجمات:

1. مع أن كتابة مترجم هي عملية صعبة ومعقدة فإن دراسة المترجمات وفهم كيفية بنائها وطريقة عملها يساعد على فهم العلاقة بين البرنامج والآلة التي تنفذ البرنامج وبالتالي على حل الكثير من المشاكل المتعلقة بكتابة البرامج وتطويرها وزيادة سرعة تنفيذها.

2. المترجم هو عبارة عن نظام كبير ومعقد لذلك فإن تصميمه يحتاج إلى توصيف دقيق لهذا النظام واستخدام الأسس النظرية بشكل كامل ودقيق للحصول على نظام يعمل بشكل صحيح.

3. تركيب المترجم يتطلب الربط بين تقنيات مختلفة نظرا للتباين الكبير بين علوم الحاسوب المختلفة التي يرتبط بها تصميم المترجم.

4. يعتبر المترجم تصميما هندسيا له أهداف محددة سلفا لذلك يجب تصميمه مع وضع الأهداف منه ومعرفتها مسبقا.

5. عند بناء المترجم يجب الأخذ بعين الاعتبار وجود عدد كبير من طرق التصميم والقرارات التي يؤثر كل منها على بناء المترجم وعلى المترجم نفسه بعد الانتهاء من تصميمه.

6. يجب تصميم المترجم بشكل يمنع أي اختراق له.

مواصفات لغات البرمجة
Programming Languages Specifications

التعريف النحوي Definition of Syntax:

يمكن تعريف Syntax بأنه علم النحو أو علم قواعد اللغة . في علوم الحاسوب Syntax هو مجموعة القواعد التي تبحث في تركيب وبناء جمل لغات البرمجة المستخدمة في كتابة البرامج وفي ترتيب الكلمات بشكل صحيح في الجملة. مجموعة القواعد توضح أيضا التركيبات المختلفة للرموز symbols التي تشكل الكلمات التي تجعل بناء البرنامج صحيحا ويمكن البحث في Syntax اللغة من خلال الشكل الخارجي للبرنامج.

لغات البرمجة النصية مثل FORTRAN, BASIC, C يتم بناء كلماتها وجملها باستخدام متسلسلات من الرموز التي تنتمي إلى أبجدية هذه اللغة. يتم تعريف Syntax لغات البرمجة النصية باستخدام تراكيب من التعبير المنتظمة Regular Expressions فيما يتعلق بمفردات اللغة، أما في ما يتعلق بالبناء القواعدي فيتم تعريفه من خلال نموذج يسمى Backus-Naur Form. أما اللغات المرئية Visual Languages فيتم بناؤها من خلال الربط بين رموز اللغة وأشكال خاصة أخرى.

Syntax اللغة يصف أو يوضح الشكل الصحيح للغة ولكنه لا يعطي أية معلومات عن معاني الكلمات في البرنامج أو عن النتائج بعد التنفيذ. ويمكن تحديد Syntax أغلب لغات البرمجة باستخدام نوع من القواعد Type-2 Grammar يعتبر من ضمن مجموعة أكبر من القواعد هي Context-Free Grammar وتقوم هذه القواعد بوصف البناء الطبقي لأغلب لغات البرمجة.

على سبيل المثال جملة if-else في لغة JAVA تكون على النحو التالي:

if (expression) statement else

هذه الجملة هي عبارة عن تسلسل Concatenation أو توالي من: الكلمة المحجوزة if ، قوس مفتوح، مقدار جبري (expression)، قوس مغلق، تصريح (statement)، الكلمة المحجوزة else ثم تصريح آخر (statement). إذا تم استخدام المتغير expr للدلالة على expression والمتغير stmt للدلالة على statement فإنه وباستخدام هذه القاعدة الجديدة في بناء الجملة يمكن التعبير عن الجملة السابقة على نحو جديد:

Stmt ⟶ if (expr) stmt else stmt

مع ملاحظة أن السهم يقرأ "يمكن أن تأخذ شكل" نجد أنه يمكن التعبير عن نفس الجملة السابقة بشكل مختلف دون فقدان المعنى الأصلي للجملة أي اختلاف الشكل دون اختلاف المضمون.

تمتلك كل جملة في اللغة بنية معينة تحددها مجموعة قواعد، فمثلا في اللغة العربية الجملية الفعلية تتكون من فعل وفاعل حيث يمكن أن يأخذ الفعل والفاعل قيما متعددة فالفعل يمكن أن يكون ماضيا أو مضارعا ..الخ والفاعل يمكن أن يكون اسم إنسان أو جماد وغيره ذلك. لو فرضنا أن للفعل قيمتان هما ذهب/حضر وللفاعل قيمتان أحمد/علي فسوف نكون قادرين على تشكيل أربع جمل صحيحة من هذه القيم:

ذهب أحمد، ذهب علي، حضر أحمد، حضر علي.

مثال آخر على تشكيل الكلمات والجمل هو إنتاج عدد صحيح باستخدام القواعد التالية:

ابجدية الأرقام= {1، 2، 3، .....، 9}

قاعدة تكوين العدد = تكوين متسلسلة من هذه الأرقام

هذه القاعدة تعني أنه باستخدام أي رقم من أبجدية الأرقام وبناءً على قاعدة تكوين العدد فإن المتسلسلات التالية جميعها تحقق القاعدة ويمكن اعتبارها أعداد صحيحة:

00000 ، 65، 67098752314567، 0

نستنتج مما سبق أن أية لغة تحتوي على عدد لا نهائي أو غير محدود من الكلمات والجمل ولكنها ليست بحاجة إلى عدد لا نهائي من الرموز أو من القواعد لتشكيل هذه الجمل بشكل صحيح. يمكن القول أن تعريف أو توصيف أي لغة برمجة يتضمن ما يلي:

1. أبجدية اللغة Alphabet: وهي عبارة عن مجموعة محدودة من الرموز تستخدم في بناء الكلمات والجمل بحيث أن أية كلمة صحيحة يجب أن تكون مشكلة من رموز تنتمي إلى أبجدية اللغة.

2. مجموعة الجمل الصحيحة يمكن كتابتها بالاعتماد قواعد خاصة باللغة وهو ما يسمى Syntax اللغة.

3. هذه الجمل الصحيحة قواعديا لها معاني محددة ومعروفة بناءً على Semantic اللغة.

عند دراسة الخصائص النحوية للغة البرمجة يمكن استخلاص ما يلي:

1. التحليل النحوي Syntactic Analysis للبرنامج المصدري أو الشفرة المصدرية يتطلب تحويل التسلسل الخطي للرموز إلى بُنية طبقية تسمى الشجرة.

2. تركيب الجملة قواعديا بشكل صحيح لا يعني بالضرورة أن يكون لهذه الجملة معنى صحيح بمعنى أن الجملة الصحيحة قواعديا قد لا تحمل معنى مفهوم أو صحيح.

3. لا يجب أن يكون هناك أي غموض Ambiguity في لغة البرمجة سواءَ كان ذلك في الشكل أو المعنى.

4. الفرق بين Syntax وبين Semantics هو أن الكثير من لغات البرمجة لها خصائص مشتركة أي أنها قد تشترك في الألفاظ ذات المعاني المتشابهة ولكن يتم التعبير عنها بشكل مختلف.

Semantics and Pragmatics:

يمكن تعريف Semantics بأنه علم دلالات الألفاظ أو معانيها أما Pragmatics فهي فلسفة الذرائع والأسباب وهي تهتم بالنتائج العملية والفعلية وتعتبرها مقياسا لقيمة الشيء وهما عبارة عن مرحلتين متتابعتين لتحليل الألفاظ والاثنان يهتمان بايجاد معنى الجملة.

في المرحلة الأولى Semantics التمثيل الجزئي للمعنى يتم الحصول عليه بالاعتماد على الأشكال التركيبية المحتملة للجملة وكذلك على معاني الكلمات المختلفة في الجملة ويعتبر هذا أمرا أكثر تعقيدا وصعوبة من كتابة الجمل أي Syntax اللغة.

في المرحلة الثانية Pragmatics يتم توضيح المعنى بشكل موسع أو أكثر دقة بالاعتماد على القرائن في سياق الكلام وعلى معرفة الكلمات وذلك من خلال ربط معنى الكلمة بالسياق العام للجملة أي بالمعنى العام للجملة التي وردت فيها الكلمة بحيث لا يتناقض معنى الكلمة مع المعنى العام للجملة وهذه مسالة ليست سهلة على الإطلاق.

نظرية اللغة الشكلية
Formal Language Theory

يقصد باللغات الشكلية اللغات المتداولة في الحديث والكتابة وتعتبر الأبجديات (Alphabets) والمتسلسلات (Strings) من المفاهيم الأساسية التي تعتمد عليها نظرية اللغات بشكل عام. المجموعة المرتبة الغير خالية تسمى أبجدية إذا كانت عناصرها عبارة عن رموز أو حروف لها إشكال تمثيلية تعرف بها ويعرف الرمز Symbol بأنه أصغر وحدة للتعامل مع اللغة وهو غير قابل للتفكك. أما التوالي المحدود للرموز المأخوذة من الأبجدية فيسمى متسلسلة.

إذا كانت المتسلسلة تحتوي على الحروف المتتالية a1,a2,…..,an فيمكن أن نرمز لها بالرمز a1a2-an . أما المتسلسلة التي تحتوي على صفر من الرموز فإنها تسمى متسلسلة خالية Empty String ويرمز لها بالرمز ε.

مثال: إذا كانت Σ1={a,…..,z} وكانت Σ2={1,…..,9} فإن:

abb هي متسلسلة تنتمي إلى Σ1

123 هي متسلسلة تنتمي إلى Σ2

ba12 ليست متسلسلة تنتمي إلى Σ1 لأنها تحتوي على رموز غير موجودة في Σ1

… 314 ليست متسلسلة لأنها ليست محدودة

المتسلسلة الخالية تنتمي إلى أي أبجدية

المجموعة الخالية Ø لا تعتبر أبجدية لأنها لا تحتوي على أي عنصر

مجموعة الأعداد الطبيعية ليست أبجدية لأنها ليست محدودة.

اتحاد الأبجديتين Σ1 و Σ2 (Σ2 U Σ1) يسمى أبجدية فقط في حالة إذا أخذت عناصره ترتيبا معينا.

اللغات Languages:

بشكل عام إذا كانت Σ أبجدية وكانت L مجموعة جزئية Subset من Σ* فأنه يمكن القول أن L هي لغة Language تنتمي إلى Σ وكل عنصر في L يسمى جملة Sentence أو كلمة Word أو متسلسلة String . المجموعات التالية : {0,1}* ,{ε,10} , {0,11,001} هي مجموعات جزئية من {0,1}* لذلك فإنها جميعها تعتبر لغات تنتمي إلى الأبجدية {0,1} .

المجموعة الخالية Ø والمجموعة {ε} هما لغتان تنتميان إلى كل الأبجديات.

Ø هي عبارة عن لغة لا تحتوي على أية متسلسلة

{ε} هي عبارة عن لغة تحتوي على متسلسلة خالية

اتحاد Union اللغتين L1,L2 يمثل على النحو التالي L1 U L2 هي عبارة عن اللغة التي تحتوي على جميع المتسلسلات الموجودة في L1 أو في L2 بحيث تحقق {x | x is in L1or X is in L2} .

تقاطع L1, L2 Intersection والذي يمثل L1∩L2 هو عبارة عن اللغة التي تحتوي على جميع المتسلسلات الموجودة في L1 و L2 في نفس الوقت بحيث تحقق {x | x is in L1 and in L2}

متممة Complementation اللغة L التي تنتمي إلى Σ وتمثل كالتالي L هي عبارة عن جميع المتسلسلات الموجودة في Σ وليست موجودة في L بحيث تحقق {x | x is in Σ* but not in L} .

مثال: إذا كانت L1= {ε, 0,1} , L2= {ε, 01, 11} فإن

L1 U L2 = {ε, 0, 1, 01, 11}

L1 ∩ L2 = {ε}

L1 = {00,01, 10, 11, 000, 001, ….}

Ø U L = L

Ø ∩ L = Ø

Ø = Σ*

Σ* = Ø

الفرق Difference بين L1 و L2 والذي يمثل (L1-L2) هو عبارة عن جميع المتسلسلات الموجودة في L1 ولكنها ليست في L2 حيث تحقق {x | x is in L1 but not in L2} .

حاصل ضرب Cross Product اللغتين L1 و L2 ويمثل ($L_1 \times L_2$) هو عبارة عن المتسلسلة التي تحتوي على جميع الأزواج المرتبة (x,y) بحيث تكون x من L1 وتكون y من L2 وتحقق العلاقة التالية:

{(x, y) | x is in L_1 and y is in L_2 }

تركيب Composition L1 مع L2 الذي يمثل (L1L2) هو عبارة عن لغة تحقق العلاقة التالية:

{ xy | x is in L_1 and y is in L_2 }

مثال: إذا كانت : {ε,1,01,11} =L1 و {1,01,101}=L2 فإن :

L1 – L2 = {ε, 11}

L2 – L1 = {101}

مثال: إذا كانت {ε,0,1}=L1 وكانت {01,11} =L2 فإن:

$L_1 \times L_2 = \{(\varepsilon, 01), (\varepsilon, 11), (0, 01), (0, 11), (1, 01), (1, 11)\}$

$L_1L_2 = \{01, 11, 001, 011, 101, 111\}$

L - Ø = L,

Ø - L = Ø,

ØL = Ø,

{ε}L = L.

تستخدم L^i لتمثيل التركيب Composition المتكون من العدد i من نسخ اللغة L حيث L^0 تعرف على أنها {ε} ,
أما المجموعة $L^0 \cup L^1 \cup L^2 \cup L^3$ ..... فتسمى نهاية كلين Kleene closure أو ببساطة نهاية L وتمثل *L .

أما المجموعة $L^1 \cup L^2 \cup L^3$ ..... فتسمى النهاية الموجبة positive closure للغة L وتمثل +L .

إذا كانت {ε,0,1} =L1 وكانت {01,11}=L2 فإن :

$L_1^{\,2} = \{\varepsilon,0,1,00,01,10,11\}$

$L_2^{\,3} = \{010101,010111,011111,110101,110111,1111101,111111\}$

التحليل اللفظي
Lexical Analysis

هي أول مرحلة من مراحل الترجمة وتسمى أيضا مرحلة المسح Scanning وفيها تتم قراءة الرموز التي يتكون منها البرنامج المصدري من اليسار إلى اليمين رمزا تلو الآخر وبعد ذلك يتم تجميعها في سلاسل لفظية ذات معنى تسمى Lexemes. لكل Lexeme يقوم المحلل اللفظي بإنتاج مخرجات من الوحدات اللفظية تسمى Tokens. يمكن تعريف Token بأنه وحدة لفظية تكون على شكل سلسلة جزئية من البرنامج المصدري يمكن معالجتها والتعامل معها كوحدة واحدة.

يتم التعامل مع الوحدة اللفظية على أنها ثنائية مكونة من جزأين: (token-name, attribute-value).

الجزء الأول (token-name) هو نمط الوحدة اللفظية وهو عبارة عن رمز مجرد يتم استخدامه خلال عملية التحليل اللفظي. أما المكون الثاني (attribute-value) فهو عبارة عن مؤشر يؤشر إلى قيد أو سجل entry في جدول الرموز يحتوي على معلومات عن token كالاسم والنوع. هذه المعلومات يحتاجها محلل المعاني Semantic Analyzer وكذلك مولد الشفرة Code Generator في مراحل لاحقة.

على سبيل المثال إذا كان البرنامج المصدري يحتوي على جملة التعيين Assignment Statement التالية:

Position = initial + rate*60

الحروف أو الرموز في هذه الجملة يتم تجميعها في Lexemes على النحو التالي:

1. Position هي Lexeme يمكن تحويله إلى token التالي (id, 1) حيث id هو رمز مجرد يدل على المعرف identifier (والمعرف هو اسم يتم إطلاقه على المتغيرات variables والثوابت constants والدوال functions) أما الرقم 1 فهو مؤشر يؤشر على قيد أو سجل في جدول الرموز يحتوي على معلومات خاصة بالمعرف Position مثل اسمه ونوعه.
2. رمز التعيين وهو في هذه الحالة إشارة المساواة = هو أيضا Lexeme يتم تحويله إلى token على الشكل <=> . وبما أن هذا token لا يحتاج إلى إعطائه صفة أو قيمة فإنه يتم حذف المكون الثاني.
3. initial هو Lexeme يتم تحويله إلى token على الشكل (id, 2)
4. الرمز + الذي يدل على الجمع يتم تحويله إلى الشكل <+>
5. rate يتم تحويله إلى (id, 3)
6. الرمز * الذي يدل على عملية الضرب يتم تحويله إلى الشكل <*>
7. العدد الثابت 60 يتم تحويله إلى الشكل <60>

بالنسبة للفراغات التي تفصل بين Lexemes يقوم المحلل اللفظي بتجاهلها وبناءً على ذلك يتم تمثيل جملة التعيين السابق ذكرها على شكل السلسلة اللفظية التالية:

(id, 1) <=> (id, 2) <+> (id, 3) <*> <60>

أما جدول الرموز فسوف يكون على النحو التالي:

1	**Position**	**……..**
2	**initial**	**……..**
3	**rate**	**……..**

كما قلنا سابقا tokens هي سلسلة جزئية من البرنامج المصدري تحتوي عليها لغات البرمجة منها:

- **الكلمات المحجوزة Keywords**
- **الثوابت Constants**
- **المعاملات Operators**
- **علامات التنصيص، الأقواس والفواصل وغيرها**

في هذه المرحلة يتم تحديد إذا كان فيض المدخلات يمكن تقسيمه إلى رموز صحيحة في البرنامج المصدري أم لا ولكن لا يمكن تحديد فيما إذا كانت هذه الرموز واقعة في المكان الصحيح الذي يجب أن توجد فيه أي أنه لا يمكن تحديد هذا النوع من الأخطاء في هذه المرحلة. أما أهم الأخطاء التي يمكن تحديدها في هذه المرحلة فهي إذا كانت الرموز المستخدمة في البرنامج المصدري غير مفهومة أو غير مسموح استخدامها. وفي حالة إذا كانت هناك أخطاء إملائية في كتابة المعرفات أو الكلمات المحجوزة فإنه لن يتم تحديدها.

مثال:

إذا كان البرنامج المصدري يحتوي على الجملة التالية

$$x = y + 1$$

فسوف يتم تقسيم هذه المدخلات إلى tokens على النحو التالي:

IDENTIFIER, EQUALS, IDENTIFIER, PLUS, INTCONSTANT, SEMICOLON

التحليل القواعدي Syntax Analysis

هي المرحلة الثانية من مراحل عمل المترجم وتسمى أيضا مرحلة الإعراب Parsing وفيها يقوم المحلل القواعدي (المحلل النحوي) Syntax Analyzer بالوظائف التالية:

1. التأكد أن الوحدات اللفظية Tokens الموجودة على مدخله والتي هي في الوقت نفسه مخرجات المرحلة السابقة لا تتنافى مع الصيغ المحددة المسموح بها في اللغة المصدرية. على سبيل المثال التعبير الرياضي التالي:

$$Y=A+/B$$

عندما يقوم المحلل القواعدي بمسح scan للتعبير سوف يشير إلى وجود خطأ قواعدي في صياغة التعبير وذلك لوجود الرمز + متبوعا مباشرة بالرمز / وهذه الصيغة لا تتطابق مع الشكل القواعدي الصحيح الذي لا يسمح بوجود عمليتين رياضيتين متتاليتين.

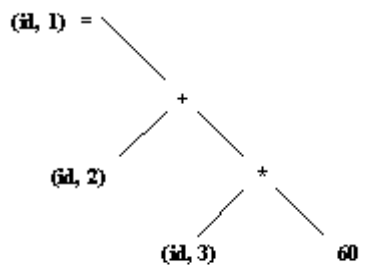
2. تكوين بنية قواعدية على شكل شجرة يتم استخدامها في المراحل اللاحقة حيث يقوم بتوضيح البنية الهيكلية.

عندما يبدأ المحلل القواعدي أو المُعَرِّب Parser عمله فإنه سيقوم باستخدام المكون الأول الذي نتج عن مرحل التحليل اللفظي في تكوين تمثيل وسيط يصف البنية القواعدية للرموز tokens حيث يقوم بتكوين ما يسمى الشجرة القواعدية Syntax Tree . هذا يعني أنه يتم في هذه المرحلة تجميع الوحدات اللفظية الناتجة عن المرحلة السابقة (التحليل اللفظي) في بُنية قواعدية تسمى تعبير Expression ومجموعة التعبيرات يتم تجميعها في بُنية أوسع تسمى عبارة أو جملة Statement . يتم مثيل هذه البنية على شكل شجرة توضح كيفية ترتيب تنفيذ العمليات في المهمة الواحدة وكيفية ارتباطها مع بعضها البعض. مثلا للعبارة

$$Position = initial + rate*60$$

نلاحظ أن عملية الضرب الممثلة بالرمز * ترتبط من جهة مع rate والذي أصبح ممثلا بالمعرف (id, 3) ومن الجهة الأخرى ترتبط مع العدد الثابت الصحيح 60 وهذا يعني أنه يجب أولا القيام بعملية الضرب ثم عملية الجمع بعد ذلك لأن هذه هي القواعد المتبعة في إنجاز العمليات الحسابية.

جذر الشجرة القواعدية يعتبر عملية المساواة الممثلة بالرمز = وهذا يعني أنه يجب تخزين ناتج عملية الجمع في الموقع الخاص بالمعرف Position. هذا الترتيب في تنفيذ العمليات وعلى هذا النحو يتطابق مع ما هو متعارف عليه من أولويات تنفيذ العمليات الحسابية. في النهاية سوف تتم إعادة تمثيل مدخلات هذه المرحلة لتصبح مخرجات وعلى النحو التالي:



تحليل المعاني

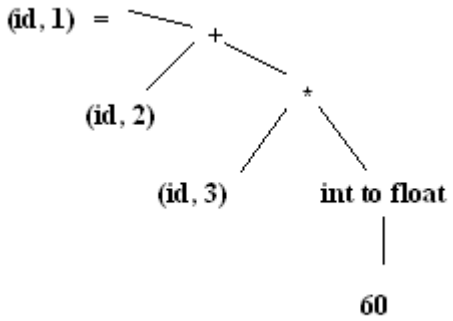
Semantic Analysis

في هذه المرحلة يتم الربط بين التعريف والاستخدام حيث يتم التحقق من أن كل تعبير Expression يمتلك صيغة صحيحة معنويا (المعنى صحيح حسب مواصفات لغة البرمجة المستخدمة) ليتم بعدها إنتاج تمثيل مناسب لتوليد برنامج بلغة الآلة على المخرج.

يقوم المحلل المعنوي باستخدام شجرة القواعد Syntax-Tree وكذلك المعلومات الموجودة في جدول الرموز لفحص مدى تطابق شفرة البرنامج المصدري مع تعريف لغة البرمجة كما يقوم بتجميع معلومات عن نوع البيانات ويقوم بحفظ هذه المعلومات إما في شجرة القواعد أو في جدول الرموز ليتم استخدامها لاحقا في مرحلة إنتاج الشفرة الوسيطة Intermediate Code. بالإضافة إلى ذلك يقوم المحلل المعنوي بإحضار معلومات عن المُعرف من جدول الرموز وكذلك إضافة معلومات عنه إلى جدول الرموز.

مواصفات بعض لغات البرمجة تسمح بالقيام ببعض أنواع التحويلات التي تسمى تحويلات قسرية أو إجبارية منها مثلا عدم السماح باستخدام عدد حقيقي كفهرس Index في مصفوفة حيث يجب أن يكون العدد صحيح وفي حالات أخرى يتم إجراء بعض التحويلات مثل تحويل العدد الصحيح إلى حقيقي عند إجراء عملية جمع لهذا العدد مع عدد آخر حقيقي.

لو عدنا للمثال وفرضنا أن rate, initial, Position قد تم تعريفها على أنها أعداد حقيقية والعدد 60 تم تعريفه كعدد صحيح فإنه عند فحص النوع Checking Type سيكتشف المحلل المعنوي بأن معامل الضرب * قد تم استخدامه لإجراء هذه العملية بين العدد الحقيقي rate والعدد الصحيح 60 وفي هذه الحالة يجب تحويل العدد الصحيح إلى صيغة عدد حقيقي وعليه ستكون مخرجات المحلل المعنوي على النحو التالي:



توليد الشفرة الوسيطة

Intermediate Code Generation

خلال ترجمة البرنامج المصدري إلى شفرة الهدف فإن المترجم قد يقوم بتكوين واحد أو أكثر من التمثيلات الوسيطة والتي يمكن أن تأخذ أكثر من شكل. شجرة القواعد Syntax Tree هي أحد أشكال التمثيل الوسيط وتُستخدم عادة خلال عمليات التحليل اللفظي والمعنوي.

بعد الانتهاء من عمليات التحليل القواعدي والمعنوي للبرنامج المصدري فإن الكثير من المترجمات تقوم بشكل صريح بتوليد أو إنتاج Generate تمثيل من المستوى المنخفض أو ما يمكن اعتباره تمثيلا يشبه لغة الآلة أو شفرة آلة مجردة بمعنى أن هذا التمثيل ليس مخصصا لآلة معينة. هذا التمثيل الوسيط يجب أن يمتلك خاصيتين مهمتين هما:

1. أن يكون توليد هذا التمثيل سهلا
2. أن تكون ترجمته إلى لغة الهدف أمرا سهلا

عادة يكون التمثيل الوسيط عبارة عن سلسلة من التعليمات البسيطة. هناك شكل من أشكال هذا التمثيل يسمى الشفرة ثلاثية العنوان Three-address Code يتم فيها استخدام ثلاثة معاملات operands (يمكن القول أن المعاملات هي المدخلات التي تُستخدم في العمليات المختلفة) وهي تحتوي على سلسلة من التعليمات الشبيهة بتعليمات لغة التجميع على النحو التالي:

```
t1=inttofloat(60)
t2=id3*t1
t3=id2+t2
id1=t3
```

في هذا النوع من التمثيل هناك عدة نقاط مهمة يجب التركيز عليها وهي:

1. كل تعليمة من تعليمات جملة التعيين assignment ثلاثية العنوان يمتلك على الأكثر معامل واحد في الجانب الأيمن وبالتالي فإن هذه التعليمات تقوم بتحديد الترتيب الذي يجب أن تُنفذ فيه العمليات أي تحديد أولويات التنفيذ فتنفيذ عملية الضرب مثلا يسبق عملية تنفيذ عملية الجمع في البرنامج المصدري.
2. يجب على المترجم توليد اسم مؤقت لتخزين أو حفظ القيمة التي تم إيجادها بواسطة التعليمة ثلاثية العنوان.
3. بعض التعليمات ثلاثية العنوان مثل التعليمة الأولى والتعليمة الأخيرة في مجموعة التعليمات السابقة تمتلك أقل من ثلاثة معاملات.

تحسين الكود

Code Optimization

في هذه المرحلة يتم تحسين شفرة البرنامج الذي يتميز باعتماده على الآلة للحصول على أفضل النتائج فيما يتعلق بشفرة الهدف وعادة كلمة "أفضل" تعني أسرع ولكن هذا لا يعني الوصول إلى خصائص أخرى كالحصول على كود قصير مثلا لذلك فإن توليد شفرة وسيطة بسيطة بعد أن يتم تحسين هذه الشفرة هو بدون شك السبب في الوصول إلى شفرة هدف جيدة.

إن مُحَسِّن الكود Code Optimizer يمكن أن يستدل أو أن يستنتج بأن تحويل القيمة 60 من عدد صحيح إلى عدد حقيقي يتم مرة واحدة طول فترة الترجمة لذلك فإن عملية التحويل inttofloat يمكن إزالتها والتخلص منها وذلك باستبدالها بالقيمة 60.0 . هناك أيضا t3 الذي يستخدم مرة واحدة فقط وذلك لنقل قيمتها إلى id1 لذلك يقوم محسن الكود بتحويل العبارات السابقة إلى تسلسل أقصر كالتالي:

t1=id3*60.0

id1=id2+t1

هناك اختلاف كبير في الدرجة أو في القدر الذي يتم فيه تحسين الكود وهذا يعتمد على المترجم الذي يقوم بذلك فمثلا المترجمات التي تسمى Optimizing Compilers تستغرق منها هذه المرحلة وقتا طويلا لذلك يتم إجراء بعض التحسينات البسيطة التي تُحسن بشكل كبير من وقت تنفيذ Run-Time برنامج الهدف ودون أن يبطئ ذلك من عملية إنشاء برنامج الهدف نفسه.

توليد الشفرة

Code Generator

يقوم مولد الشفرة Code Generator بأخذ الشفرة الوسيطة على أنها مدخلات البرنامج المصدري ويحولها إلى شفرة مكتوبة بلغة الهدف وإذا كانت شفرة الهدف هي نفسها شفرة الآلة فسوف يتم اختيار مسجلات ومواقع في الذاكرة لتخزين كل متغير يستخدمه البرنامج. بعد ذلك يتم ترجمة تعليمات الشفرة الوسيطة إلى سلاسل من تعليمات الآلة التي تقوم بإنجاز نفس المهمات.

الجانب الحاسم في توليد هذه الشفرة هو التوظيف الصحيح والمناسب للمسجلات عند تخزين المتغيرات. على سبيل المثال إذا تم استخدام المسجلين R2, R1 فإن الشفرة الوسيطة السابقة يمكن تحويلها إلى شفرة الآلة التالية:

```
LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1
```

المعامل الأول في كل واحدة من هذه التعليمات يحدد الوجهة أو الهدف أما F في كل تعليمة فيعني أنه يتم التعامل مع أعداد حقيقية. تبين هذه الشفرة أيضا أن محتويات العنوان id3 قد تم تحميلها إلى المسجل R2 ثم بعد ذلك تم ضرب هذه المحتويات بالعدد الحقيقي الثابت 60.0 أما الرمز # فيوضح أن 60.0 قد تم التعامل معه على اعتبار أنه ثابت فوري Immediate Constant.

التعليمة الثالثة تقوم بنقل id2 إلى المسجل R1 في حين التعليمة الرابعة تقوم بإضافة القيمة التي تم حسابها سابقا في المسجل R2 إليها. أخيرا القيمة الموجودة في المسجل R1 قد تم تخزينها في العنوان id1 وبذلك يمكن القول أن تعليمات البرنامج المصدري قد تم تنفيذها بشكل صحيح.

إدارة جدول الرموز

Symbol Table Management

يمكن تعريف جدول الرموز بأنه عبارة عن هيكل بيانات Data Structure يحتوي على سجلات لأسماء المتغيرات المستخدمة في البرنامج المصدري كما يحتوي على معلومات عن الخصائص المختلفة لكل اسم من أسماء المتغيرات. هيكل البيانات هذا يتم تصميمه بشكل يسمح للمترجم بإيجاد السجل المطلوب وكذلك حفظ السجل واسترجاعه بسرعة كبيرة. أما المعلومات التي يتم جمعها عن الاسم فتتضمن ما يلي:

1. **سلسلة المحارف التي تشكل الاسم**
2. **نوع الاسم (عدد صحيح، عدد حقيقي أو سلسلة نصية ..الخ)**
3. **صيغة الاسم**
4. **موقع الاسم في الذاكرة**
5. **بعد المواصفات التي تختلف من لغة لأخرى**

إن كل سطر في جدول الرموز هو عبارة عن ثنائية على الشكل (اسم، معلومات) وفي كل مرة يتم فيها إدخال اسم فإنه يجب إجراء بحث في الجدول لمعرفة فيما إذا كان الاسم المُدخل موجود سابقا أم لا وإذا كان إدخاله لأول مرة فسوف يتم أيضا إدخال المعلومات المتعلقة بهذا الاسم لاستخدامها في مراحل الترجمة المختلفة.

المعلومات المُجَمعة في جدول الرموز يمكن أن تستخدم خلال مراحل متعددة فمثلا يمكن استخدام هذه المعلومات في مرحلة تحليل المعاني وذلك للتأكد من التوافق بين التصريح عن الاسم واستخدام هذا الاسم أو في مرحلة توليد الشفرة لمعرفة كمية ونوعية الذاكرة التي يجب تخصيصها للإسم.