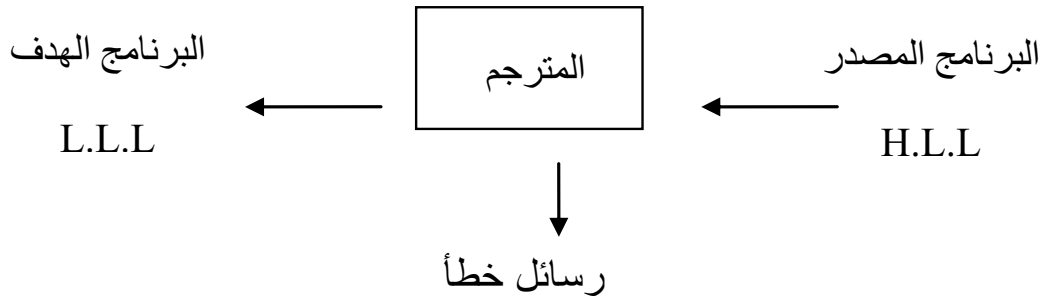
1.1 المترجم (Compiler)

يعرف المترجم (Compiler) بأنه البرنامج الذي يقوم بقراءة برنامج مكتوب بإحدى اللغات البرمجية العليا (اللغة المصدر) [High Level Language HLL] ويترجمه إلى البرنامج المكافئ بلغة أخرى (اللغة الهدف) [Low Level Language LLL] وقد تصدر عن عملية الترجمة هذه أخطاء تسمى الأخطاء القواعدية (Syntax Errors).



وهناك فرق بين المترجمات (Compilers) وبين ما يسمى بالمفسرات (Interpreters) هو أن المترجمات تقوم بترجمة البرنامج بشكل كامل (قطعة واحدة) ،أما المفسرات (interpreters) فإنها تقرا البرنامج إيعاز إيعاز وإذا وجدت خطأ تتوقف عند ذلك الإيعاز ويسمى ذلك الخطأ بالخطأ القواعدي (Syntax Error) وهناك نوع آخر من المترجمات والذي يسمى بالمجمع (Assembler) حيث يقوم بترجمة البرنامج المكتوب بإحدى لغات الاسمبلي (لغات البرمجة الدنيا) ويحولها إلى لغة الآلة كما أن هناك مترجمات في مجالات مختلفة تقوم بتحويل الشفرة من نوع إلى نوع آخر مكافئ حسب الحاجة وهذا الكتاب غير معني بشرح تلك الأنواع.

1.2 الترجمة (Compilation)

إن هذه العملية تشير إلى عمل المترجم في تحويل البرنامج المكتوب بلغة برمجة عليا إلى برنامج مكافئ ولكن بلغة برمجة دنيا وتوجد ثلاث أنواع من اللغات:

1. اللغات الطبيعية (Natural Languages) مثل (العربية، الانكليزية، الفرنسية ... الخ) وهذه اللغات لها معالجة خاصة يطلق عليها معالجة اللغات الطبيعية Natural Language Processing (NLP))).
2. اللغات البرمجية (Programming Languages) مثل (C++، فيجوال بيسك، جافا، باسكال ... الخ) ولكل منها مترجمه الخاص فان للغة باسكال مثلا مترجم خاص يختلف عن مترجم لغة C++ .
3. اللغات الشكلية (Formal Languages) مثل اللغة الشكلية $\{ a^n b^n : n>=1\}$ حيث تتكون هذه اللغة من مجموعة محددة من الكلمات المتكونة من عدد من أحرف الـ a وعدد مماثل له من أحرف الـ b كما يلي :

{ab, aabb, aaabbb,…}

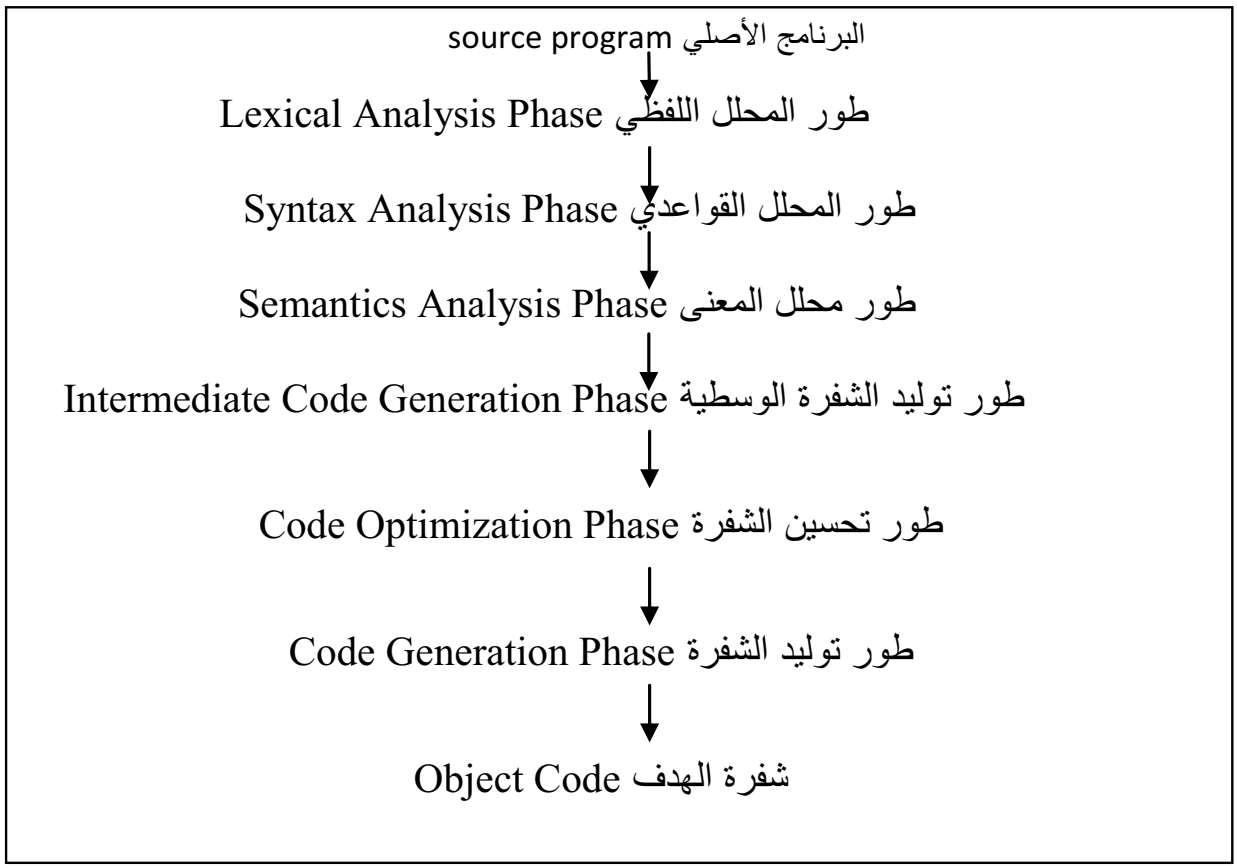
إن عملية الترجمة النموذجية تعتبر معقدة جدا لذا فان هذه العملية تقسم إلى عدة أطوار كما في الشكل (1.1) وأن هذه الأطوار تستخدم جداول الرموز كما سيأتي شرحه في الفصل الثامن وهناك إدارة لتلك الجداول (Symbol Table Manager) وخلال هذه المراحل قد يظهر خطأ ما لذا فهناك معالج للأخطاء (Error Handler) ومن هذه المعالجات إظهار رسائل خطأ للمستخدم تبين فيها موضع ونوع الخطأ ، وهناك نوعين رئيسين من الأخطاء ، النوع الأول يظهر في مرحلة الترجمة (Compilation) ويسمى الخطأ القواعدي (Syntax Error) والآخر يظهر في مرحلة التنفيذ (Run Time) ويسمى بالخطأ التنفيذي (Runtime Error).

لو أخذنا الجملة التالية:

For (int I = 0; I <=10; I ++);

- فان في مرحلة التحليل اللفظي (Lexical Analysis) يتم تقطيع هذه الجملة إلى مقاطع تسمى (Tokens) وفي مثالنا فان المقاطع هي :

{for , (, int, I, = ,0 , ; , <=, 10 , ++,) }



شكل (1.1) المراحل النموذجية للمترجم

- في مرحلة التحليل القواعدي (Syntax Analysis) يتم التدقيق فيما إذا كانت تلك الجملة مقبولة قواعديا أم لا ففي مثالنا فان الجملة مقبولة قواعديا كونها كتبت بشكل قواعدي صحيح تبعا للغة برمجة معينه وليكن لغة C++ أما لو كتبت بالشكل

For (int I = = 0, I = 10, I ++);

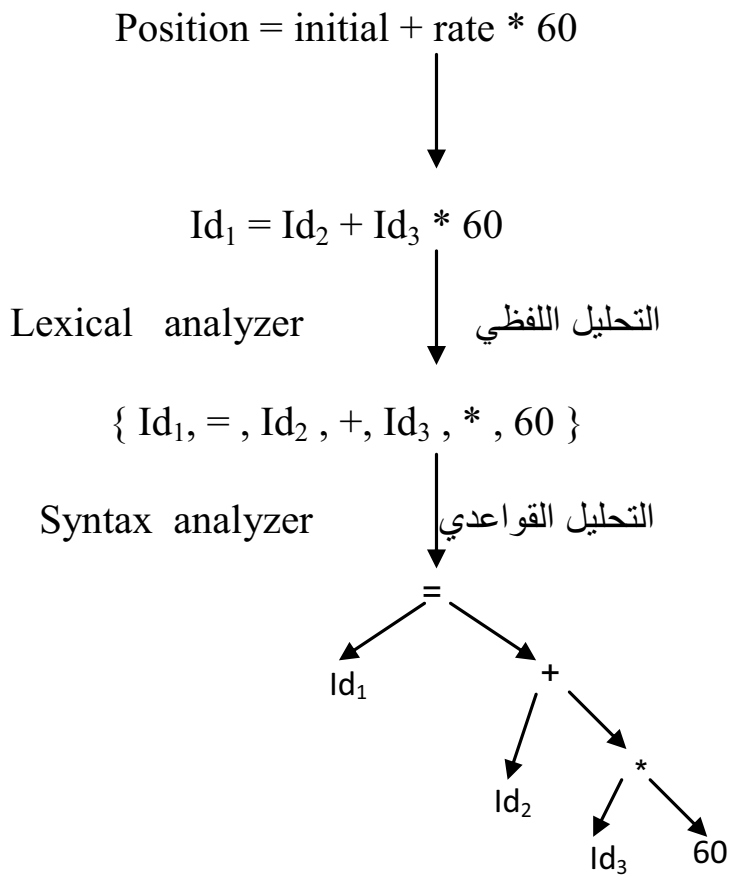
فان هذه الجملة تجتاز مرحلة التحليل اللفظي لكنها لا تجتاز مرحلة التحليل القواعدي حيث تعتبر هذه الجملة غير صحيحة قواعديا ولم تكتب بشكل قواعدي سليم تبعا للغة برمجة معينه وليكن لغة C++.

- في مرحلة تحليل المعنى (Semantic Analysis) فانه يتم التأكد من معنى الجملة فمثلا أن الجملة الثانية من المثال التالي:

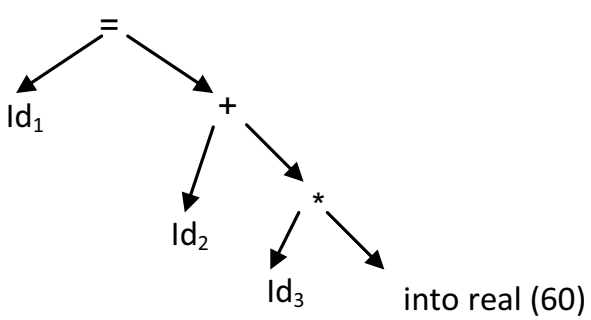
```
int X=5;
X=X + 1.5;
```

تعتبر جملة صحيحة قواعديا وتجتاز مرحلة المحلل اللفظي ولكنها غير صحيحة من ناحية المعنى كون X من نوع عدد صحيح وتم جمعه مع 1.5 الذي يعتبر عدد حقيقي وهذا لا يجوز.

مثال 1.1: لو كانت لدينا الجملة التالية Position = initial + rate * 60 فإنها تمر بالمراحل الآتية:



Semantic analyzer ↓ التحليل المعنوي



Intermediate code generator ↓ توليد الشفرة الوسطية

$Temp_1$:= into real (60)

$Temp_2$:= Id_3 * $Temp_1$

$Temp_3$:= Id_2 + $Temp_2$

Id_1 := $Temp_3$

Code optimizer ↓ تحسين الشفرة

$Temp_1$:= Id_3 * 60.0

Id_1 := Id_2 + $Temp_1$

Code generator ↓ توليد الشفرة

Movf Id_3, R_2

Mulf #60.0, R_2

Movf Id_2, R_1

Addf R_2, R_1

Movf R_1, Id_1