مقرر مبادئ الأوتومات والمترجمات

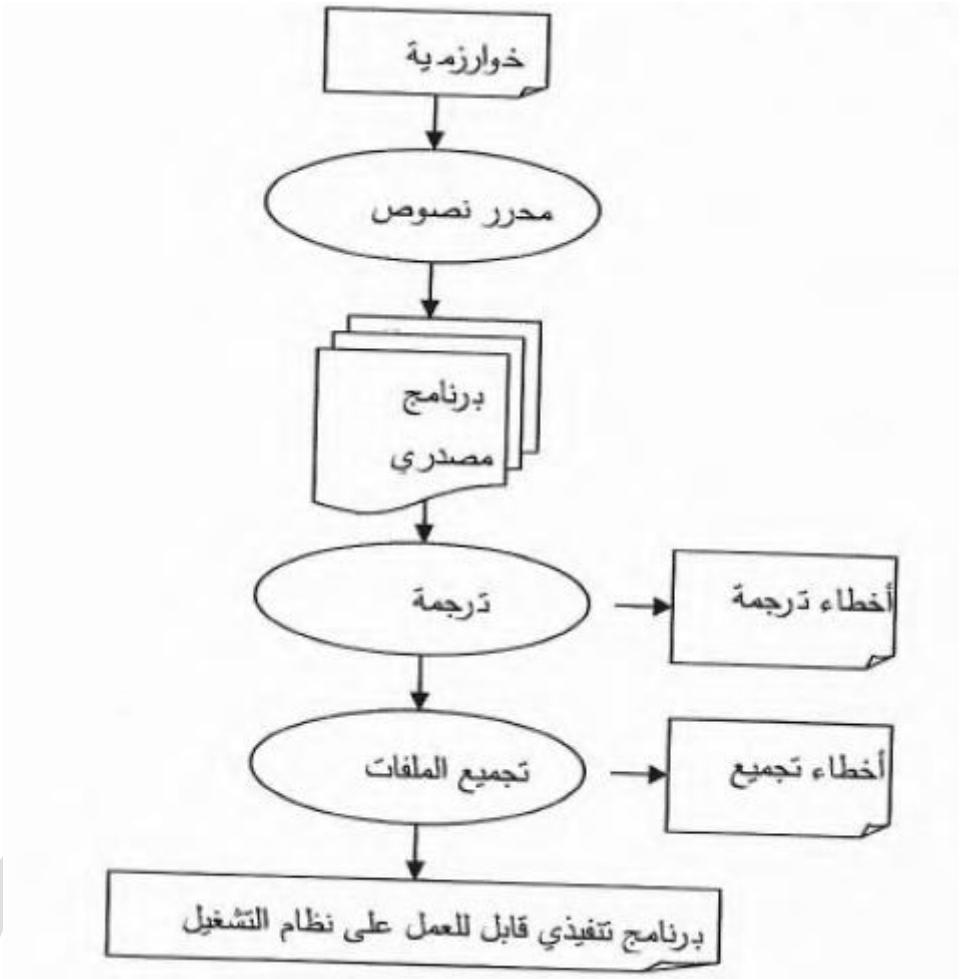
يتضمن المقرر المفاهيم الآتية :

1- مقدمة عن المقرر
2- بنية المترجم
   - مرحلة التحليل (التحليل المفرداتي – التحليل القواعدي – التحليل الدلالي)
   - مرحلة التركيب والتوليد (توليد الرماز)
   - مراحل موازية (إدارة جدول الرموز – إدارة الأخطاء)
3- التحليل اللفظي (التحليل المفرداتي)
   - المفردات
   - االتعابير المنتظمة
   - تنفيذ التحليل المفرداتي
   - أخطاء المفردات
4- اللغات الصورية والأوتومات
   - الأوتومات المنتهي
   - تحويل تعبير منتظم إلى أوتومات منته لاحتمي
   - تحويل أوتومات منته لاحتمي إلى أوتومات منته حتمي
   - تحويل أوتومات منته إلى تعبير منتظم
   - الأوتومات ذات المكدس
5- التحليل القواعدي
   - النحو الصرفي ومفهوم شجرة الاشتقاق
   - تنفيذ التحليل القواعدي
   - الأخطاء القواعدية
6- التحليل الدلالي
   - مجال تعريف ورؤية المتحولات
   - التحقق من الأنماط
7- توليد الرماز
   - البنية الوسيطة
   - تنظيم الذاكرة وتنفيذ عملية الحساب
   - توليد الرماز المقابل للتعليمات

المقدمة :

يستخدم أي مبرمج أداة ضرورية جداً في عملية البرمجة، ندعوها المترجم. يمكننا تعريف المترجم بأنه برنامج حاسوبي يترجم النص البرمجي الذي نكتبه بلغة برمجة عالية المستوى (C++ , C# , Java , ...) إلى مجموعة تعليمات قابلة للتنفيذ من قبل الحاسوب. تكون هذه التعليمات التنفيذية مكتوبة بلغة منخفضة المستوى سواء كانت لغة ثنائية مؤلفة من أصفار وواحدات (0,1) أو لغة تجميع. ويمكن أيضاً بناء مترجمات من لغة غالية المستوى إلى لغة أخرى عالية المستوى وخصوصاً عندما تكون هناك ضرورة لنقل رماز (كود) برمجي من بيئة برمجية إلى أخرى.

عموماً تمر عملية تشغيل برنامج حاسوبي بمجموعة من المراحل التي نمثلها في الشكل التالي :



تجري كتابة النص البرمجي (أو النصوص البرمجية) لأي برنامج حاسوبي باستخدام محرر نصوص وذلك ضمن ملف واحد أو ضمن مجموعة من الملفات. ندعو هذه النصوص البرمجية التي تؤلف برنامج حاسوبي بالبرنامج المصدري (Source Program).

يمر البرنامج المصدري بعد ذلك بمرحلة ترجمة (Compilation) وتجميع (Linking) يجري فيها ربط الملفات الحاوية على البرنامج المصدري الواحد ببعضها البعض وترجمتها إلى مجموعة من التعليمات التنفيذية المكتوبة بلغة منخفضة المستوى.

تشكل التعليمات الناتجة برنامج جديد ندعوه البرنامج الهدف (Destination Program). يأخذ في الحالة العامة شكل ملف تنفيذي قابل للتشغيل مباشرة على نظام التشغيل.

تظهر خلال مراحل الترجمة والتجميع أخطاء ندعوها أخطاء الترجمة (تكون ناجمة عن أخطاء في نصوص البرنامج المصدري) أو أخطاء التجميع (تكون ناجمة عن أخطاء في ربط الملفات الحاوية على النصوص). تؤدي هذه الأخطاء إلى توقف عملية الترجمة حتى يجري تصحيحها من قبل المبرمج. قبل إعادة تشغيل المترجم لتوليد "برنامج هدف" خال من الأخطاء.

تجدر الإشارة إلى أن عملية بناء المترجم تتعلق بعنصرين اثنين بآن واحد:

1. لغة البرمجة المصدرية عالية المستوى التي يستخدمها المبرمج.
2. نظام التشغيل أو البيئة التي سيجري تشغيل البرمامج عليها.

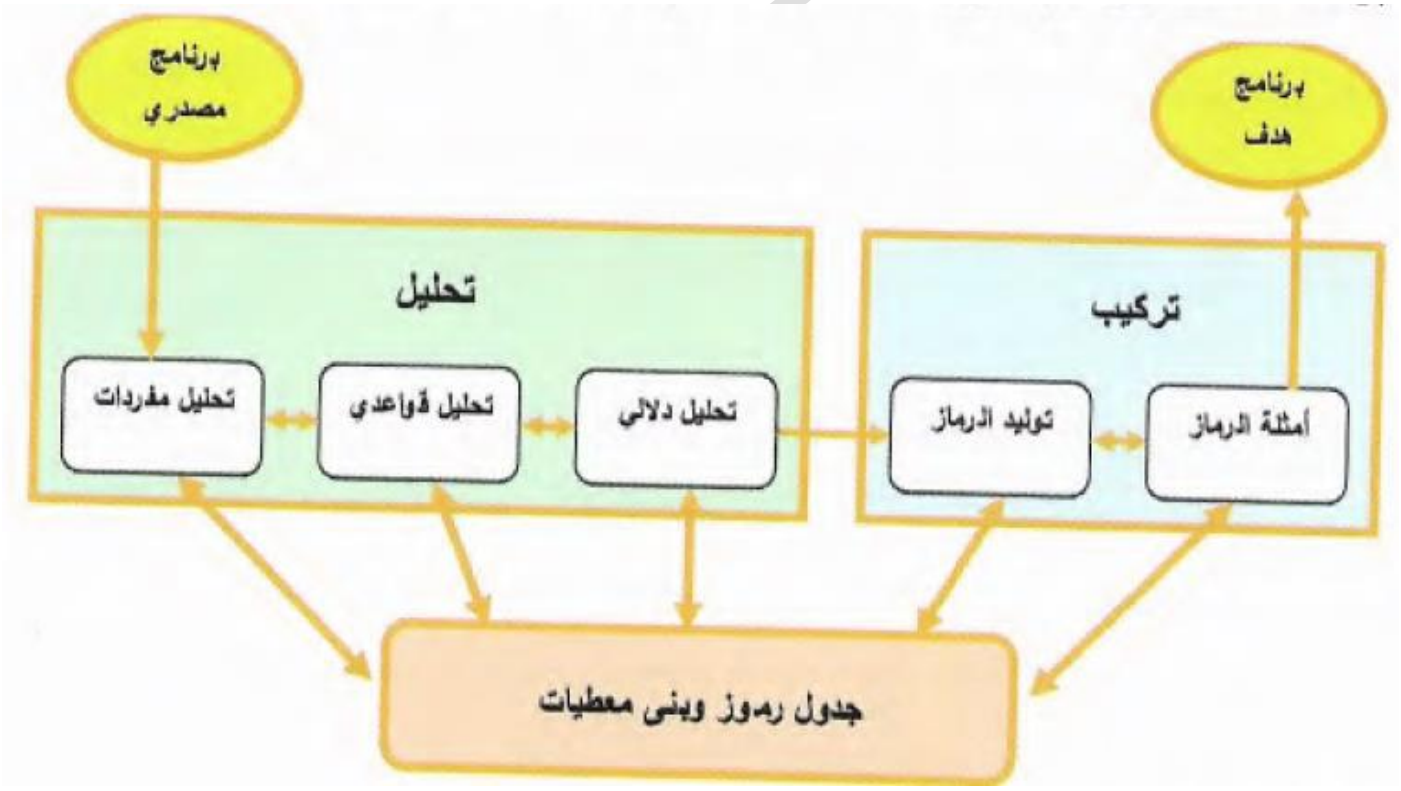
فعلى سبيل المثال، يختلف مترجم لغة C+++ الذي يعمل على نظام Windows عن مترجم لغة C++ الذي يعمل على نظام Linux وذلك نظراً لضرورة توليد تعليمات تشغيل تنفيذية مختلفة في هاتين الحالتين. بالرغم من أننا نتكلم عن نفس اللغة البرمجية وهي لغة C++ في حين يختلف مترجم لغة C++ عن مترجم لغة Java حتى ولو كان المترجمان يعملان على نظام Windows وذلك نظراً لأننا نترجم لغتين برمجيتين مختلفتين.

بنية المترجم :

تتألف عملية الترجمة من مرحلتين أساسيتين:

- مرحلة التحليل والتي يجري فيها تفسيم النص البرمجي إلى كلمات وجمل والتأكد من صحتها ودلالتها.
- ومرحلة التركيب التي يجري فيها تركيب نص برمجي جديد بنفس دلالة النص المصدري ولكن بلغة أخرى هي عموماً اللغة التي تتألف منها التعليمات التنفيذية التي يفهمها نظام أو بيئة التشغيل.

يوضح الشكل التالي مكونات المرحلتين:



ولنتعرف الآن على هاتين المرحلتين.

1. مرحلة التحليل

التحليل اللفظي أو التحليل المفرداتي (Lexical Analysis)

تجري في هذه المرحلة عملية التعرف على أنواع الكلمات المؤلفة للنص البرمجي المصدري. لذا تجري قراءة النص البرمجي المصدري حرفاً حرفاً وتجميع الحروف لتشكيل كلمات. يتولى التحليل اللفظي المهام التالية:

1. حذف كافة المحارف التي لاتدخل في صلب النص البرمجي مثل الفراغات والتعليقات ... ألخ.
2. تجميع المحارف في كلمات وتحديد نوع كل كلمة:
   - كلمة مفتاحية من لغة البرمجة keyword
   - متحول Variable
   - ثابت Constant
   - رمز Symbol
   - قيمة عددية Numeric Value
   - عملية حسابية أو عملية منطقية. وغيرها...

التحليل القواعدي (Syntax Analysis)

يمكن أيضاً تسميته بالتحليل الصرفي، إذ يجري خلال التحليل القواعدي جمع الكلمات الناتجة عن التحليل المفرداتي في جمل وبنى قواعدية تشكل بنية النص البرمجي. يقوم المحلل القواعدي بالتأكد من سلامة الجمل المبنية بالنسبة لقواعد صرفية خاصة بكل لغة برمجة.

على سبيل المثال، تحدد القواعد الصرفية طريقة بناء الجملة الشرطية في لغة البرمجة (مثل لغة C++) أو طريقة بناء حلقة تكرار فيها. (مثلاً عندما نجد كلمة if يجب أن يأتي بعدها قوس ثم يأتي الشرط الواجب تحققه وبعده يأتي قوس الإغلاق)

تشبه عملية التحليل القواعدي للغة البرمجة عملية الإعراب في اللغات الطبيعية، إذ يمكن لجملة عربية أن تكون فعلية مؤلفة من فعل وفاعل ومفعول به أو جملة اسمية مؤلفة من مبتدأ وخبر. كذلك هو الحال في جملة شرطية مكتوبة بلغة C++ حيث يجب أن تبدأ هذه الجملة بكلمة if يليها تعبير شرطي يعبر عن شرط معين مثل X>0 ومن ثم مجموعة من العمليات التي يجب تنفيذها عند تحقق الشرط. ويمكن أن تتبع هذه العمليات كلمة else لتعريف العمليات التي تجري عند انتفاء الشرط. أو يمكن أن تنتهي الجملة الشرطية دون كلمة else

التحليل الدلالي (Semantic Analysis)

تجري في هذه المرحلة عملية التحقق من دلالة الجمل المركبة بعد أن تم التأكد من سلامتها قواعدياً. فعلى سبيل المثال تعتبر عملية التحقق من الأنماط مرحلة أساسية من مراحل التحليل الدلالي. حيث يجري فيها التأكد من أن عملية مثل X*Y لها دلالة أم لا وذلك إذا كانت كل من X و Y تعبر عن قيمتين رقميتين. في حين تصبح هذه العملية دون دلالة في حال كانت X تعبر عن قيمة رقمية وكانت Y تعبر عن سلسلة محارف.

2. مرحلة التركيب والتوليد

توليد الرماز (Code Generation)

تجري في هذه المرحلة عملية توليد التعليمات التنفيذية التي لها نفس دلالة تعليمات النص البرمجي المصدري ولكن بلغة مفهومة من كل من بيئة ونظام التشغيل اللذين سيجري تشغيل البرنامج عليهما.

يمكن في بعض الأحيان توليد الرماز بلغة أخرى عالية المستوى (في حال كان المطلوب هو نقل نصوص برمجية مكتوبة بلغة برمجة قديمة إلى نصوص برمجية مكتوبة بلغة برمجة أحدث وذلك بهدف تسهيل الربط مع برامج أخرى).

كما يمكن توليد الرماز بلغة خاصة بآلة افتراضية تعمل على نظام تشغيل كما هو الحال في آلة Java الافتراضية JVM اختصاراً لـ Java Virtual Machine والتي تعمل على نظام التشغيل Windows أو Linux وغيرها.

أو يمكن توليد الرماز بلغة آلة خاصة بنوع معين من المعالجات وضمن بيئة نظام تشغيل كما هو الحال في بيئة التشغيل الخاصة بنظام Windows والتي تعمل على معالجات الحواسب الشخصية X86

أمثلة الرماز Code Optimization

تهتم هذه المرحلة باختصار وتحسين الرماز المولد وذلك بهدف تسريع عمل البرنامج التنفيذي الناتج وضمان تنفيذ سريع وفعال له عند تشغيله. يجري في هذه المرحلة حذف تعليمات لا معنى لها مثل تعريف متحول وعدم استخدامه أو ضرب قيمة ما بـ 1 كذلك جمع قيمة ما مع 0 وغيرها من التعليمات التي لا معنى لها.

3. مراحل موازية

إدارة جدول الرموز (Symbol Table)

يشكل جدول الرموز أحد أهم بنى المعطيات المستخدمة في المترجمات. إذ يجري تخزين المتحولات المعرفة ضمن النص البرمجي المصدري في جدول الرموز. كما يجري تخزين أنماطها التي جرى الإعلان عنها في النص البرمجي.

يُستخدم جدول الرموز أيضاً لتحديد مجال رؤية المتحول. وهو مجموعة مقاطع النص البرمجي وإجرائياته التي يمكن فيها استخدام هذا المتحول وفقاً لتعريفه في جدول الرموز. فعلى سبيل المثال في بعض لغات البرمجة وعند الإعلان عن متحول ما x كمتحول محلي من النمط Integer ضمن الإجرائية Proc فلا يمكن للمبرمج استخدام هذا المتحول ضمن النص البرمجي خارج نطاق هذه الإجرائية. لذا يقوم المترجم بالاعتماد على جدول الرموز لتخزين اسم المتحول ونمطه ومجال رؤيته بهدف التأكد من عدم استخدام هذا المتحول خارج هذه الإجرائية Proc.

إدارة الأخطاء (Error Handling)

تنتج عن عمليات التحليل والتركيب السابقة أخطاء ارتكبها المبرمج أثناء كتابته للنص البرمجي منها أخطاء في كتابة الكلمات ومنها أخطاء في بناء الجمل وبعضها أخطاء دلالية.

يجري التعامل مع كل نوع من أنواع الأخطاء بشكل مختلف ولكن يبقى الهدف الأول والأخير لمعالجة الأخطاء هو إعطاء المبرمج إشارة إلى الخطأ وتوضيح سبب الخطأ قدر الإمكان ومحاولة تجميع أكبر عدد من الأخطاء الناجمة عن خطأ أو وإظهارها بآن واحد وذلك بهف تسريع عملية الترجمة.

التحليل اللفظي أو التحليل المفرداتي (Lexical Analysis)

يشكل المحلل اللفظي الجزء الأول من المترجم وينفذ المرحلة الأولى من عملية الترجمة. تتلخص المهمة الأساسية للمحلل اللفظي بتجميع محارف الدخل الآتية من النص البرمجي المصدري بهدف توليد مجموعة من الكلمات التي ستؤلف بدورها جمل يعالجها المحلل القواعدي.

ينفذ المحلل المفرداتي أيضاً مجموعة من المهام الثانوية من أهمها: حذف المحارف التي لا دور لها كالفراغات والتعليقات كما يتولى مهمة حفظ أرقام أسطر النص البرمجي التي يمكن الاستعانة بها للإشارة إلة مكان الأخطاء الموجودة في النص البرمجي.

إن مسألة تصميم محلل لفظي تكافئ مسألة تصميم برنامج ينفذ عمليات على سلاسل من المحارف ويهتم بالتعرف على أشكال وصيغ محددة لهذه السلاسل. وهي مسألة تصب في مجال بناء المنظومات التي تساعد في البحث عن المعلومات وتشكل صلب محركات البحث.

المفردات

في كل نص برمجي تشكل كل مجموعة من الأحرف المتتالية كلمة word حيث يكون لكل كلمة من الكلمات المستخدمة في النص البرمجي شكل يحدد انتماءها إلى نوع من أنواع الكلمات أو ما ندعوه نموذج Model

ونستخدم لمجموعة الكلمات التي لها نموذج محدد اسم ندعوه مفردة Token

فعلى سبيل المثال: تعبر الكلمات + - * / % عن عمليات حسابية

أو بصيغة أخرى فإن الرمز + هو عبارة عن كلمة وهذه الكلمة تعبر عن عملية حسابية هي عملية الجمع. وهكذا ...

كما تعبر الكلمات (Toto , Y , Counter , X) عن متحولات variable يستخدمها المبرمج. ونرمز لها عادة بـ ID

كما يكون للمتحولات نموذج يعبر عن شكل المتحول فعلى سبيل المثال يفرض النموذج الذي يعرف شكل المتحولات عدم استخدام أرقام لتعريف متحول إذا لا يمكن للمحلل اللفظي اعتبار 55 متحول بل يعتبر المحلل اللفظي أن 55 قيمة رقمية.

التعابير المنتظمة

تعريف: ندعو "أبجدية" (Alphabet)مجموعة منتهية غير خالية Σ من الرموز. كما ندعو "كلمة" (Word) كل سلسلة منتهية من عناصر Σ

نستخدم الرمز ε للدلالة على الكلمة الفارغة. كما نستخدم الرمز Σ^* للدلالة على المجموعة غير المنتهية التي تضم جميع الكلمات الممكنة المبنية اعتباراً من الأبجدية Σ بما في ذلك الكلمة الفارغة. بينما نستخدم الرمز Σ^+ للدلالة على مجموعة الكلمات غير الفارغة التي يمكن أن نبنيها اعتباراً من Σ

بالنتيجة يكون: $\Sigma^+ = \Sigma^* - \{\varepsilon\}$

يشير الرمز |m| إلى طول الكلمة m وهو يعبر عن عدد الأحرف المنتمية إلى الأبجدية Σ والتي تشكل الكلمة m

ونستخدم الرمز Σ^n للدلالة على مجموعة الكلمات المنتمية إلى Σ^* والتي طولها n

وبالتالي يمكننا أن نستنتج أن: $\Sigma * = \sum_{n=0}^{\infty} \Sigma n$

تعريف: نعرف عملية "الدمج التسلسلي" أو "التتابع" (Concatenation) والتي نرمز لها بالرمز " . " بأنها عملية نطبقها على كلمتين كالتالي:
لتكن لدينا الكلمة $u = u_1...u_n$ حيث $u_i \in \Sigma^*$ والكلمة $v = v_1...v_p$ حيث $v_i \in \Sigma^*$ يكون حاصل الدمج التسلسلي للكلمتين u و v هو الكلمة $u.v = u_1...u_n\, v_1...v_p$
بشكل عام يمكن إهمال رمز الدمج التسلسلي وبالتالي يمكن كتابة uv بدلاً عن u.v

فعلى سبيل المثال لنأخذ الأبجدية $\Sigma = \{ a , b , c \}$ والكلمات u , v , t , w حيث:

$u = aaba$ $v = bbbacbb$ $w = c$ $t = \varepsilon$

هي كلمات تنتمي إلى Σ^* وبأطوال 4 , 7 , 1 , 0 على التسلسل.

تكون الكلمة $u.v = aababbbacbb$ حاصل الدمج التسلسلي للكلمتين u و v

خصائص:	$\|u.v\| = \|u\| + \|v\|$
	$(u.v).w = u.(v.w)$
	$u.\varepsilon = \varepsilon.u = u$

تعريف: ندعو لغة مبنية على أبجدية Σ ، كل مجموعة جزئية من Σ^*

على سبيل المثال يمكن اعتباراً من الأبجدية $\Sigma = \{ a , b , c \}$ تعريف لغة غير منتهية L_1 حيث تتألف هذه اللغة مثلاً من الكلمات: $\{ \varepsilon , a , b , aab , bbcaa , bbcccacccaaaabbbcaa , ... \}$

تعريف: العمليات على اللغات

الاجتماع: $L_1 \cup L_2 = \{ w : w \in L_1 \text{ OR } w \in L_2 \}$

التقاطع: $L_1 \cap L_2 = \{ w : w \in L_1 \text{ AND } w \in L_2 \}$

الدمج التسلسلي أو التتابع: $L_1L_2 = \{ w = w_1w_2 : w_1 \in L_1 \text{ AND } w_2 \in L_2 \}$

الإغلاق: $L^* = \mathbf{U}_{n\geq 0} L^n$

هنا نطرح السؤال التالي:

في حال كان لدينا لغة ما L فكيف يمكن توصيف الكلمات المنتمية إلى اللغة؟ وكيف يمكن توصيف هذه اللغة؟

عموماً توجد عدة أنماط من اللغات ولكن سنركز الآن على اللغات التي ندعوها لغات منتظمة.

تعريف: نستطيع تعريف **لغة منتظمة** L (Regular Language) على أبجدية Σ بشكل عودي كما يلي:

1) { ε } هي لغة منتظمة على Σ
2) إذا كان a حرف من حروف الأبجدية Σ تكون { a } لغة منتظمة على Σ
3) إذا كانت L لغة منتظمة على Σ تكون كل من L^n حيث $n \geq 0$ و L^* لغات منتظمة على Σ
4) إذا كانت L لغة منتظمة على Σ تكون متممة L بالنسبة للمجموعة الكلية Σ^* هي لغة منتظمة ونرمز لها $\overline{L}$
5) إذا كان كل من L_1 و L_2 لغات منتظمة على Σ تكون كل من $L_1 \cup L_2$ و $L_1 \cap L_2$ و L_1L_2 لغات منتظمة على Σ

تعريف: يمكن توصيف اللغات المنتظمة باستخدام أداة ندعوها التعابير المنتظمة (Regular Expressions)
نعطي فيما يلي تعريف عودي للتعابير المنتظمة:

1) ε هو تعبير منتظم يوصف اللغة { ε }
2) إذا كان a حرف من حروف الأبجدية Σ يكون a تعبير منتظم يوصف اللغة { a }
3) إذا كان r تعبير منتظم يوصف اللغة L فإن كلاً من $(r)^*$ و $(r)^+$ عبارة عن تعبيرين منتظمين يوصفان كلاً من اللغتين L^* و L^+ على الترتيب.
4) إذا كان r_1 و r_2 تعبيرين منتظمين يوصفان اللغتين L_1 و L_2 على الترتيب فإن كلاً من $r_1 + r_2$ و r_1r_2 عبارة عن تعبيرين منتظمين يوصفان اللغتين $L_1 \cup L_2$ و L_1L_2 على الترتيب.

أمثلة عن التعابير المنتظمة:

- ✓ يوصف $(a + b)^*$ مجموعة الكلمات (اللغة) المؤلفة من a و b أو الكلمة الفارغة.
- ✓ يكون $(a + b)^* = (b + a)^*$
- ✓ يوصف $(a + b)^*bbb(a + b)^*$ مجموعة الكلمات (اللغة) التي تحتوي على السلسلة bbb فيها.