Compilers Techniques
Lecture 10
المحاضرة العاشرة

Intermediate Code Gengeration توليد الشيفرة الوسيطية

Optimization (Instruction selection) تحسين الشيفرة

Reference: Aho Pages 357-421

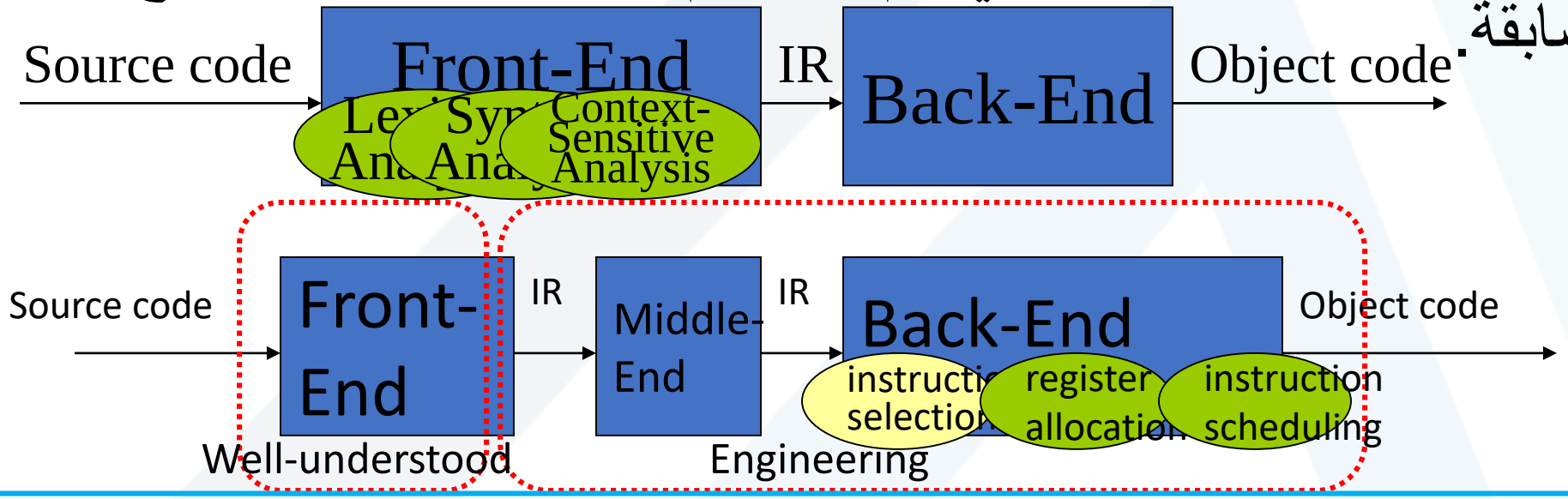
السنة الرابعة – المستوى السابع- الهندسة المعلوماتية

تعرفنا في المحاضرات السابقة على مراحل Front-End وهي التحليل المعجمي والتحليل القواعدي والتحليل المعنوي

محاضرة اليوم:

Intermediate Code Generation توليد الشيفرة الوسيطة

يتم في هذه المرحلة ترميز كل المعرفة التي قام المترجم بتجميعها حول البرنامج المصدري خلال المراحل السابقة.

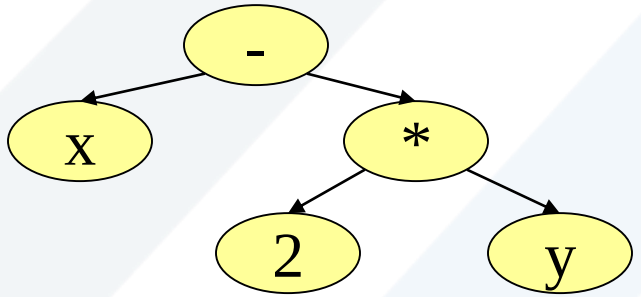


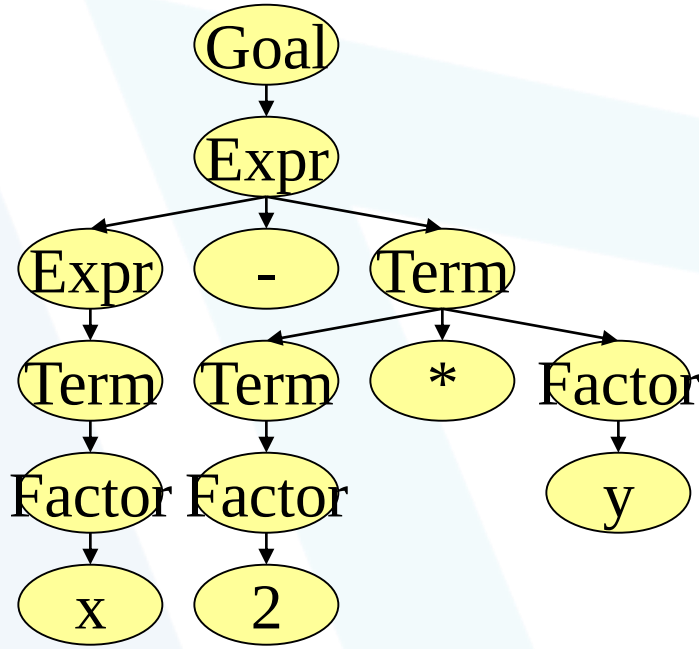
From Parse Trees to Abstract Syntax Trees (AST)

توليد الشيفرة الوسيطة
Intermediate Code Generation

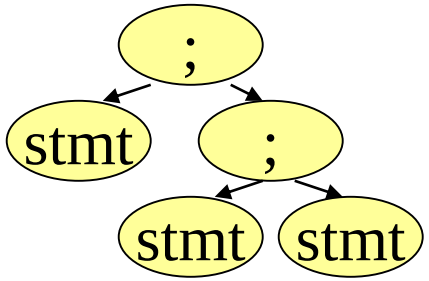
1. *Goal* → *Expr*
2. *Expr* → *Expr* + *Term*
3. | *Expr* – *Term*
4. | *Term*
5. *Term* → *Term* * *Factor*
6. | *Term* / *Factor*
7. | *Factor*
8. *Factor* → *number*
9. | *id*

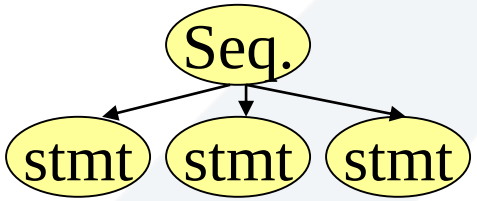
- لماذا نحتاج للتحويل لشجرة AST:
- لتقليل المعلومات الزائدة التي لن نحتاج لها في مرحلة توليد الشيفرة.
- تعتبر أقرب تمثيل ممكن للكود.
- كيف نحول شجرة الإعراب إلى شجرة مجردة AST؟
- نزيل اللاطرفيات Non-Terminals ونحتفظ فقط بالطرفيات Terminals وتتم هذه العملية وفق مراحل Traverse اجتياز للاطرفيات.

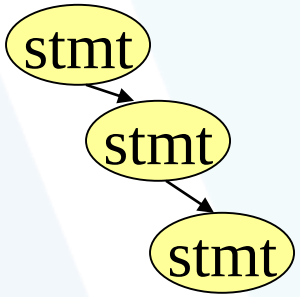




Example: *Stmt_sequence* → *stmt; Stmt_sequence* | *stmt*
At least 3 AST versions for *stmt; stmt; stmt*







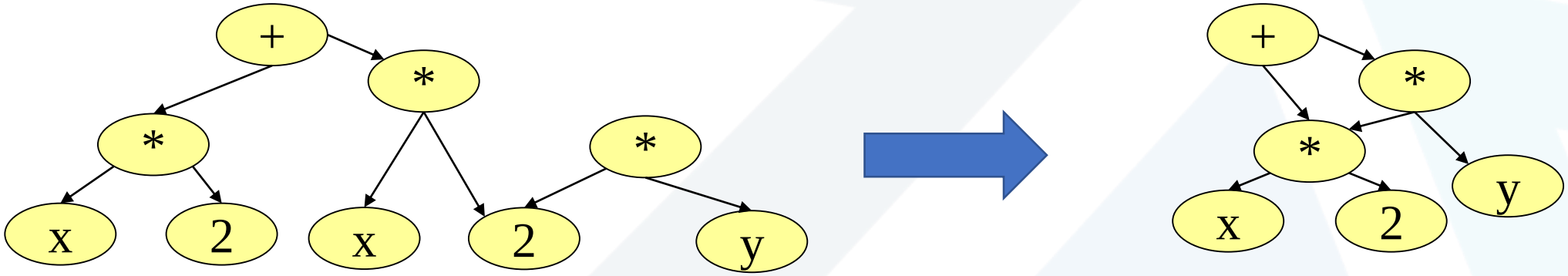
المشكلة هي إمكانية رسم أكثر من شجرة AST لنفس عبارة الدخل (لنفس البرنامج المصدري)
الحل برسم شجرة Directed Acyclic Graph

Directed Acyclic Graphs (DAGs)

توليد الشيفرة الوسيطة
Intermediate Code Generation

DAG هي شجرة AST لكن مع عقدة واحدة فقط لكل قيمة.

Example: The DAG for x*2+x*2*y



من ميزات DAG أنها تلغي الزيادات في بنية الشجرة وتساعد على الحصول على شجرة مختصرة تماماً
مشكلتها صعبة التحويل إلى كود نظراً للاختصارات الشديدة فيها.

Three-address code
الشيفرة ثلاثية العناوين

توليد الشيفرة الوسيطة
Intermediate Code Generation

يشير مصطلح الشيفرة ثلاثية العناوين إلى شيفرة مؤلفة من 3 حدود على الأكثر وهي التعليمة والحد المصدر والحد الهدف
محاسنها: ذات صيغة موجزة، توضح القيم الوسيطية، وملائمة جداً للغة الآلة.
مساوئها: حتى وقت قريب كانت هناك مشكلة في المساحة اللازمة للحجز خلال زمن Compile Time بسبب هذه الشيفرة.
تمتلك هذه الشيفرة الشكل العام الآتي:

A := B op C

حيث B,C هي الحدود المصدرية للتعليمة أما A فهو الحد الهدف.
هناك شيفرة ثنائية الحدود وشيفرة أحادية الحدود حيث الشيفرة الثنائية أقرب للغة الآلة أما الأحادية فهي الأقرب.

مثال:

Three address code

```
t1=load x
t2=load y
t3=2*t2
t4=t1-t3
```

Two address code

```
load  r1,y
loadi r2,2
mult r2,r2,r1
load r3,x
sub r3,r3,r2
```

تستهلك حجم أعلى من one-address
تسمح باستخدام تعليمات بحد واحد أو اثنين وهي أقل حجماً من three-code

One address code

```
push  2
push  y
multiply
push  x
subtract
```

تستهلك حجم أقل
مستخدمة في لتوليد Bytecodes لبرامج الجافا

Symbol Table
جدول الرموز

تمثيلات وسيطية أخرى

يعتبر جدول الرموز من التمثيلات الوسيطية رغم أنه يتم إنشاؤه قبل البدء بهذه المرحلة وقبل توليد الشيفرة.

يعتبر من معماريات المترجم التي تستخدم في كلا الطرفين Front-End و Back-End

يربط جدول الرموز بين جزء صغير من الكود مع كل قاعدة نحو (نتاج) بحيث عندما يتم تنفيذ هذا النتاج أو القاعدة يتم تنفيذ هذا الكود.

مثلا: الكود الذي يتنفذ عند تفعيل القاعدة (النتاج) Factor → id هو الكود الخاص بالتحقق من أن المتحول id قد تم التصريح عنه مسبقاً (ويمكن التحقق من ذلك من خلال التأكد من أن الرمز موجود ضمن جدول الرموز).

مثال آخر: بالنسبة للقاعدة (النتاج) $Term \rightarrow Term*Factor$ يتم التحقق أنَّ الحدين اليساري واليميني من نفس نوع البيانات.

يتم الاحتفاظ بجداول الرموز عبر جميع مراحل الترجمة (لأنه يساعد أيضاً في مرحلة Debugging)

تمثيلات وسيطية أخرى

Variable names أسماء المتغيرات
defined constants الثوابت المعرفة
procedure and function names أسماء التوابع والإجرائيات
literal constants and strings الثوابت والسلاسل المحرفية
source text labels تسميات النص المصدري (البرنامج المصدري)
compiler-generated temporaries أي بيانات مؤقتة تم إنشاؤها من قبل المترجم وستحذف عند انتهاء الترجمة.

يحتاج المترجم خلال مراحل الترجمة من جدول الرموز الأمور الآتية:

أسماء المتحولات – أنواع بياناتها – التصريحات الخاصة بالتوابع – معلومات تخزينية.

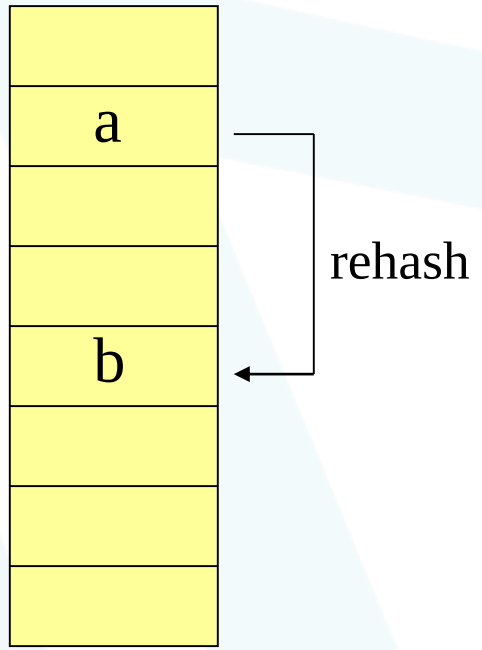
طرق تنظيم جدول الرموز

تمثيلات وسيطية أخرى

Linear Rehashing (open addressing)

يتألف من جدول كبير جداً لحفظ القيم بشكل متسلسل تنظم فيه البيانات بشكل تسلسلي وتحدث فيه الكثير من التصادمات خلال الحجز

درجة تعقيد هذا النوع O(N) بسبب وجود اللوائح المتصلة.



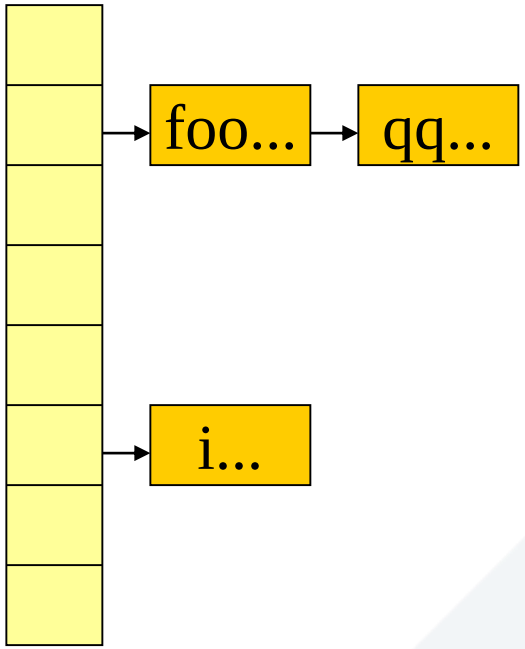
If $h(a)=h(b)$ rehash (say, add 3).

Bucket hashing (open hashing)

يتألف من مصفوفة من m مؤشر.

تنظم فيه البيانات على شكل لوائح منفصلة-متصلة.

درجة تعقيد هذا النوع O(1) بسبب وجود اللوائح المتصلة (المقصود الزمن اللازم للعثور على رمز)



Binary Tree

يتألف من لائحة متصلة بدرجة تعقيد $O(\log_2(N))$

Instruction Selection

تحسين الشيفرة أو اختيار التعليمات
Code Optimization

- Example: w=w*2*x*y*z

```
 1. load r1, @w
 2. mov  r2, 2
 6. mult r1,r1,r2
 7. load r2, @x
12. mult r1,r1,r2
13. load r2, @y
18. mult r1,r1,r2
19. load r2, @z
24. mult r1,r1,r2
26. store r1
```

2 registers, 31 cycles

```
 1. load r1, @w ; load: 5 cycles
 2. load r2, @x
 3. load r3, @y
 4. load r4, @z
 5. mov  r5, 2  ; mov: 1 cycle
 6. mult r1,r1,r5     ; mult: 2 cycles
 8. mult r1,r1,r2
10. mult r1,r1,r3
12. mult r1,r1,r4
14. store r1     ; store: 5 cycles
```

5 registers, 19 cycles

- Instruction scheduling (the problem): given a code fragment and the latencies for each operation, reorder the operations to minimise execution time (produce correct code; avoid spilling registers)