TDDB44 – Seminar 4
Exam Preparation

Adrian.Pop@liu.se

2022-12-14

LINKÖPINGS UNIVERSITET

Exam from 2022-08-25

Exercise 1

1. (3p) **Compiler Structure and Generators**

   (a) (1p) What are the advantages and disadvantages of a multi-pass compiler (compared to an one-pass compiler)?

   (b) (2p) Describe briefly what phases are found in a compiler. What is their purpose, how are they connected, what is their input and output?

Exercise 1

1. (3p) **Compiler Structure and Generators**

   (a) (1p) What are the advantages and disadvantages of a multi-pass compiler (compared to an one-pass compiler)?

   (b) (2p) Describe briefly what phases are found in a compiler. What is their purpose, how are they connected, what is their input and output?

Advantages

- wider scope, allows better optimization, better code generation
- Is needed for some type of languages (forward references)
- More modular design

Disadvantages

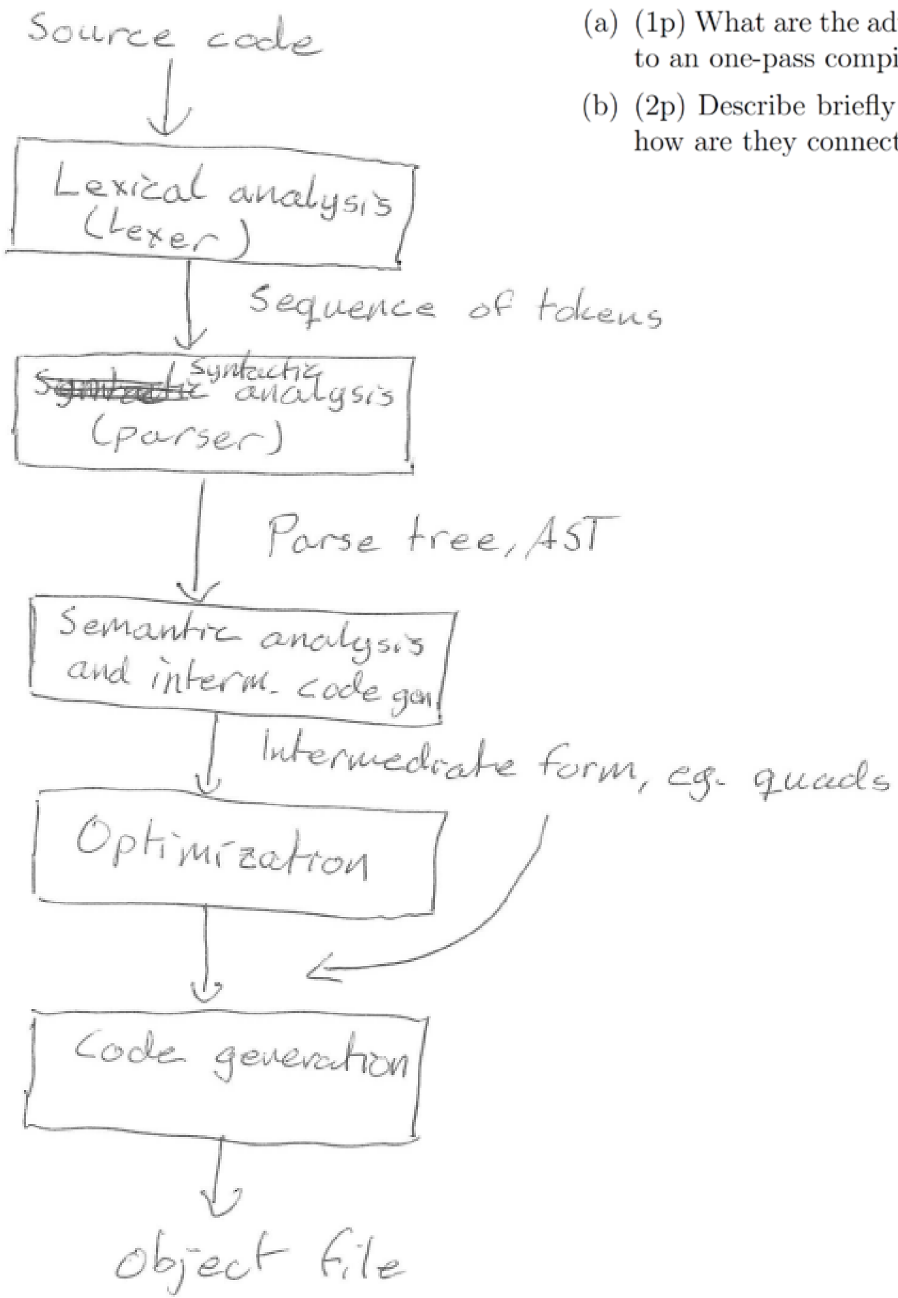
- Longer compile times and higher memory consumption

Exercise 1

1. (3p) **Compiler Structure and Generators**

   (a) (1p) What are the advantages and disadvantages of a multi-pass compiler (compared to an one-pass compiler)?

   (b) (2p) Describe briefly what phases are found in a compiler. What is their purpose, how are they connected, what is their input and output?



Exercise 2

2. (5p) **Top-Down Parsing**

   (a) (4.5p) Given a grammar with nonterminals L, E, F and the following productions:

   `L::=` L α| E F β| F E β

   `E::=` E γ| δ

   `F::=` E ψ| ϵ

   where `L` is the start symbol, α, β, γ, δ, ψ and ω are terminals. (ϵ is the empty string!) What is/are the problem(s) with this grammar if it is to be used for writing a recursive descent parser with a single token lookahead? Resolve the problem(s), and write a recursive descent parser for the modified grammar. (Pseudocode/program code without declarations is fine. Use the function scan() to read the next input token, and the function error() to report errors if needed.)

   (b) (0.5p) The theory for formal languages and automata says that a stack is required for being able to parse context-free languages. We have used such a stack, for instance, in the LL-item pushdown automaton in the lecture on top-down parsing. But where is the corresponding stack in a recursive descent parser?

Exercise 2

(b) (0.5p) The theory for formal languages and automata says that a stack is required for being able to parse context-free languages. We have used such a stack, for instance, in the LL-item pushdown automaton in the lecture on top-down parsing. But where is the corresponding stack in a recursive descent parser?

- In a recursive descent parser the stack is the call stack (dynamic link).

$$
\begin{aligned}
&\texttt{L::= L } \alpha \texttt{| E F } \beta \texttt{| F E } \beta \\
&\texttt{E::= E } \gamma \texttt{| } \delta \\
&\texttt{F::= E } \psi \texttt{| } \epsilon
\end{aligned}
$$

Exercise 2

What is/are the problems with the grammar if it is to be used for writing a recursive descent parser with a single token lookahead?

- Is it left recursive? Why?
- Why left recursive grammars are problematic for recursive descent parsers?
- What can we do?

$$
\begin{aligned}
&\mathtt{L ::= L}\ \alpha \mid \mathtt{E\ F}\ \beta \mid \mathtt{F\ E}\ \beta \\
&\mathtt{E ::= E}\ \gamma \mid \delta \\
&\mathtt{F ::= E}\ \psi \mid \epsilon
\end{aligned}
$$

Exercise 2

What is/are the problems with the grammar if it is to be used for writing a recursive descent parser with a single token lookahead?

- Is it left recursive? Why? – Starts with the same nonterminal on LHS.
- Why left recursive grammars are problematic for recursive descent parsers? – It would generate infinite loops!
- What can we do? – Refactor the grammar.

Exercise 2

$$L ::= L\ \alpha \mid E\ F\ \beta \mid F\ E\ \beta$$
$$E ::= E\ \gamma \mid \delta$$
$$F ::= E\ \psi \mid \epsilon$$

L ::= L a | E F b| F E b
E ::= E c | d
F ::= E g | e

- Refactor the grammar, eliminate left recursion
- Immediate left recursion
- E ::= E c | d
  =>
  E ::= d E'
  E' ::= c E' | e
- L ::= L a | E F b| F E b
  =>
  L ::= L a | L1
  L1 ::= E F b | F E b
  F E b => E g E b | e E b
  E F b | E g E b | E b => E E"
  E" ::= F b | g E b | b
  E" ::= F b | g E b // F can be e
  =>
  L ::= L1 L2
  L2 ::= a L2 | e
- L ::= L1 L2
  L1 ::= E E"
  E" ::= F b | g E b
  L2 ::= a L2 | e
  E ::= d E'
  E' ::= c E' | e
  F ::= E g | e

Exercise 2

$$L ::= L\ \alpha \mid E\ F\ \beta \mid F\ E\ \beta$$
$$E ::= E\ \gamma \mid \delta$$
$$F ::= E\ \psi \mid \epsilon$$

L ::= L1 L2
L1 ::= E E”
E” ::= F b | g E b
L2 ::= a L2 | e
E ::= d E’
E’ ::= c E’ | e
F ::= E g | e

- write a RDP parser for the modified grammar

```
Parse () { L(); }
L() { E(); E”() }
E” { if (scan() == “g”) { E(); if scan() != “b” error() }
     else { F(); if scan() != “b” error() }  }
L2() { scan() == a; L2() }
```

Exercise 3

3. (3p) **LR parsing**

Use the SLR(1) tables below to show how the string α `-` β `+` α `*` β is parsed. You should show, step by step, how stack, input data etc. are changed during the parsing. Start state is `00`, start symbol is `S`.

Grammar:

1. `S ::= A + A`
2. `A ::= B - A`
3. `| B`
4. `B ::= B * C`
5. `| C`
6. `C ::=` α
7. `|` β

Tables:

	Action						GOTO			
State	$	+	-	*	α	β	S	A	B	C
00	*	*	*	*	S09	S10	01	02	05	08
01	A	*	*	*	*	*	*	*	*	*
02	*	S03	*	*	*	*	*	*	*	*
03	*	*	*	*	S09	S10	*	04	05	08
04	R1	*	*	*	*	*	*	*	*	*
05	R3	R3	S06	S11	*	*	*	*	*	*
06	*	*	*	*	S09	S10	*	07	05	08
07	R2	R2	*	*	*	*	*	*	*	*
08	R5	R5	R5	R5	*	*	*	*	*	*
09	R6	R6	R6	R6	*	*	*	*	*	*
10	R7	R7	R7	R7	*	*	*	*	*	*
11	*	*	*	*	S09	S10	*	*	*	12
12	R4	R4	R4	R4	*	*	*	*	*	*

$\alpha - \beta + \alpha * \beta$

Exercise 3

Stack		Input
0	ACTION[0, α] = S9	$\alpha - \beta + \alpha * \beta\$$
0α9	ACTION[9, $-$] = R6	$-\beta + \alpha * \beta\$$
0C	GOTO[0, C] = 8	$-\beta + \alpha * \beta\$$
0C8	ACTION[8, $-$] = R5	$-\beta + \alpha * \beta\$$
0B	GOTO[0, B] = 5	$-\beta + \alpha * \beta\$$
0B5	ACTION[5, $-$] = S06	$-\beta + \alpha * \beta\$$
0B5-6	ACTION[6, β] = S10	$\beta + \alpha * \beta\$$
0B5-6β10	ACTION[10, $+$] = R7	$+ \alpha * \beta\$$
0B5-6C8	GOTO[6, C] = 8	

Exercise 5

5. (3p) **Symbol Table Management**

The C language allows static nesting of scopes for identifiers, determined by blocks enclosed in braces. Given the following C program fragment (some statements are omitted):

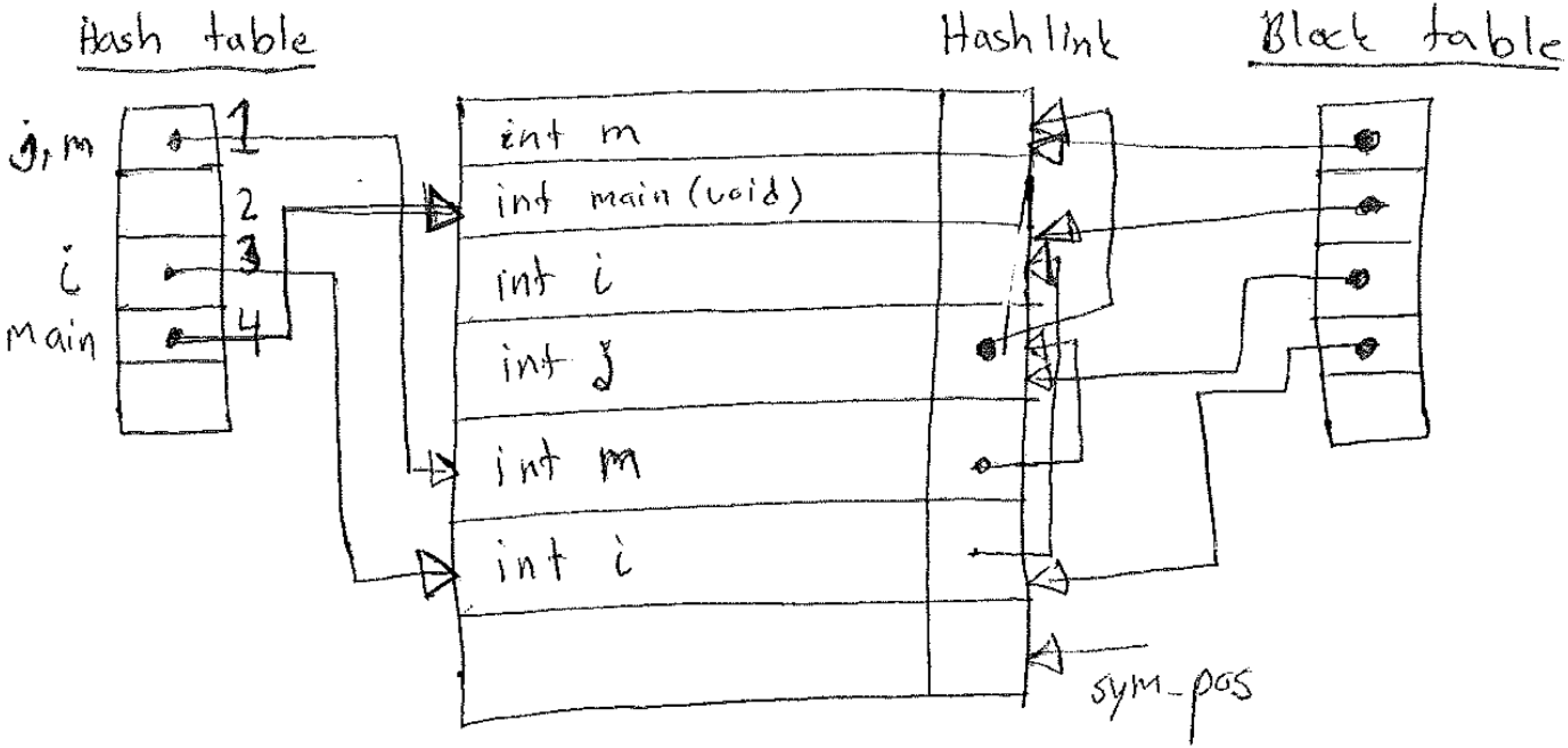
```
int m;
int main( void ) {
  int i;
  if (i==0) {
    int j, m;
    for (j=0; j<100; j++) {
      int i;
      i = m * 2;
    }
  }
}
```

(a) (2p) For the program point containing the assignment `i = m * 2`, show how the program variables are stored in the symbol table if the symbol table is to be realized as a hash table with chaining and block scope control. Assume that your hash function yields value 3 for `i`, value 1 for `j` and `m`, and value 4 for `main`.

(b) (0.5p) Show and explain how the right entry of the symbol table will be accessed when looking up identifier `m` in the assignment `i = m * 2`.

(c) (0.5p) After code for a block is generated, one needs to get rid of the information for all variables defined in the block. Given a hash table with chaining and block scope control as above, show how to "forget" all variables defined in the current block, without searching through the entire table.

Exercise 5



Hash table
Hashlink
Block table
j, m
i
main
1
2
3
4
int m
int main (void)
int i
int j
int m
int i
sym_pos

Exercise 7

7. (3p) **Error Handling**

   Explain, define, and give examples of using the following concepts regarding error handling:

   (a) (1p) Valid prefix property,

   (b) (1p) Phrase level recovery,

   (c) (1p) Global correction.

Exercise 7

7. (3p) **Error Handling**

Explain, define, and give examples of using the following concepts regarding error handling:

(a) (1p) Valid prefix property,

(b) (1p) Phrase level recovery,

(c) (1p) Global correction.

a) LL and LR parsers have the valid prefix property. They will report an error as soon as the parser prefix is not a valid prefix.
Example: “a = b then” will report and error when parsing “then”

b) The parser may perform local corrections if an error is discovered

c) The parser finds the syntax tree of the correct string with a minimum edit distance to the given erroneous string

Exercise 8

8. (3p) **Memory management**

   (a) (1p) What does an activation record contain?

   (b) (1p) What happens on the stack at function call and at function return?

   (c) (1p) What are static and dynamic links? How are they used?

a) AR – all data needed to call a function and execute it
b) An AR is created, arguments are passed, the function is called, on return SP is decreased, the previous AR is active, the return result is read
c) Static link point to AR of enclosing scope, dynamic link is the call stack, points to the previous function on the stack (the caller)

Exercise 9

9. (3p) **Intermediate Representation**

Given the following code segment in a Pascal-like language:

```
if x=y
  then x:=x-10
  else while y>10 do
    if y<x
      then y:=y+1
      else y:=func(x)
```

Translate the code segment into an abtract syntax tree, quadruples, and postfix code.

Exercise 10

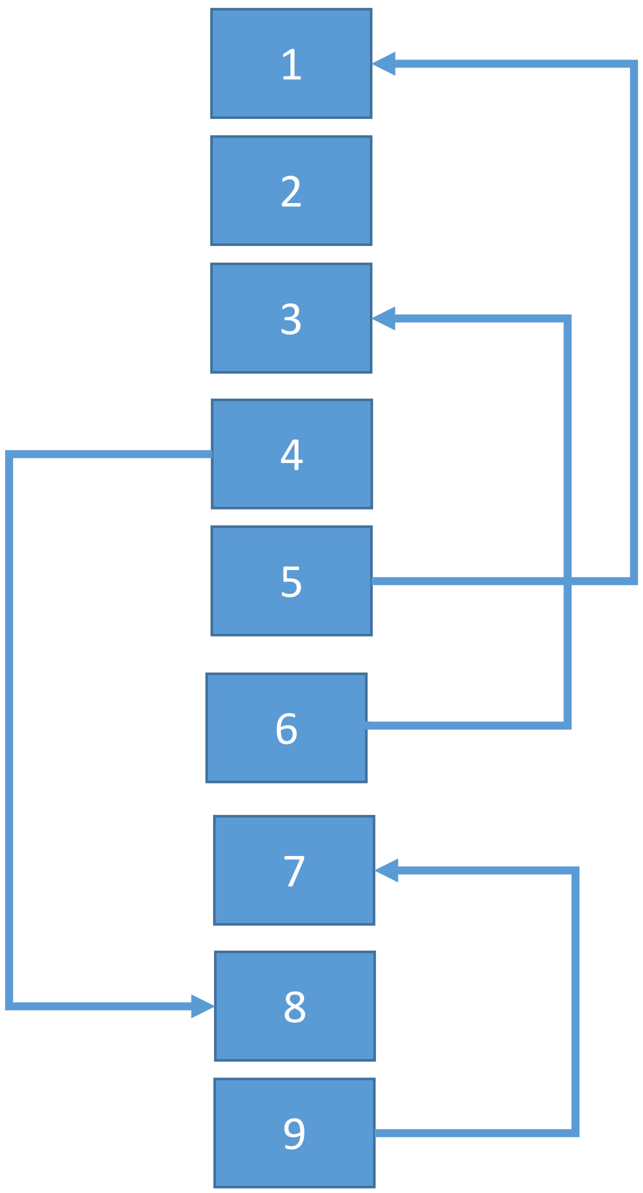
10. (3p) **Intermediate Code Generation**

Divide the following code inte basic blocks, draw a control flow graph, and show as well as motivate the existing loop(s):

```
L1: x:=x+1
L2: x:=x+1
L3: x:=x+1
    if x=1 then goto L5
    if x=2 then goto L1
    if x=3 then goto L3
L4: x:=x+1
L5: x:=x+1
    if x=4 then goto L4
```

Exercise 10

1. L1: x:=x+1
2. L2: x:=x+1
3. L3: x:=x+1
4. if x=1 then goto L5
5. if x=2 then goto L1
6. if x=3 then goto L3
7. L4: x:=x+1
8. L5: x:=x+1
9. if x=4 then goto L4



BBlocks

1. 1 & 2
2. 3
3. 4
4. 5
5. 6
6. 7
7. 8 & 9

Loops?

1-6? SEP? SC?

7-9? SEP? SC?

Exercise 4

4. (3p) **LR parser construction**

Given the following grammar `G` for strings over the alphabet $\{\alpha, \beta, \gamma, \delta\}$ with nonterminals `A` and `B`, where `A` is the start symbol:

$$\mathtt{A} ::= \alpha\mathtt{A} \mid \mathtt{A}\beta \mid \alpha\mathtt{B}\beta \mid \gamma$$
$$\mathtt{B} ::= \beta\mathtt{B} \mid \mathtt{B}\alpha \mid \beta\mathtt{A}\alpha \mid \delta$$

Is the grammar `G` in SLR(1) or even LR(0)? Justify your answer using the LR item sets. If it is: construct the characteristic LR-items NFA, the corresponding GOTO graph, the ACTION table and the GOTO table.

If it is not: describe where/how the problem occurs.