TDDE66 – Seminar 4
Exam Preparation

Adrian.Pop@liu.se

2024

LINKÖPINGS UNIVERSITET

TDDB44 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

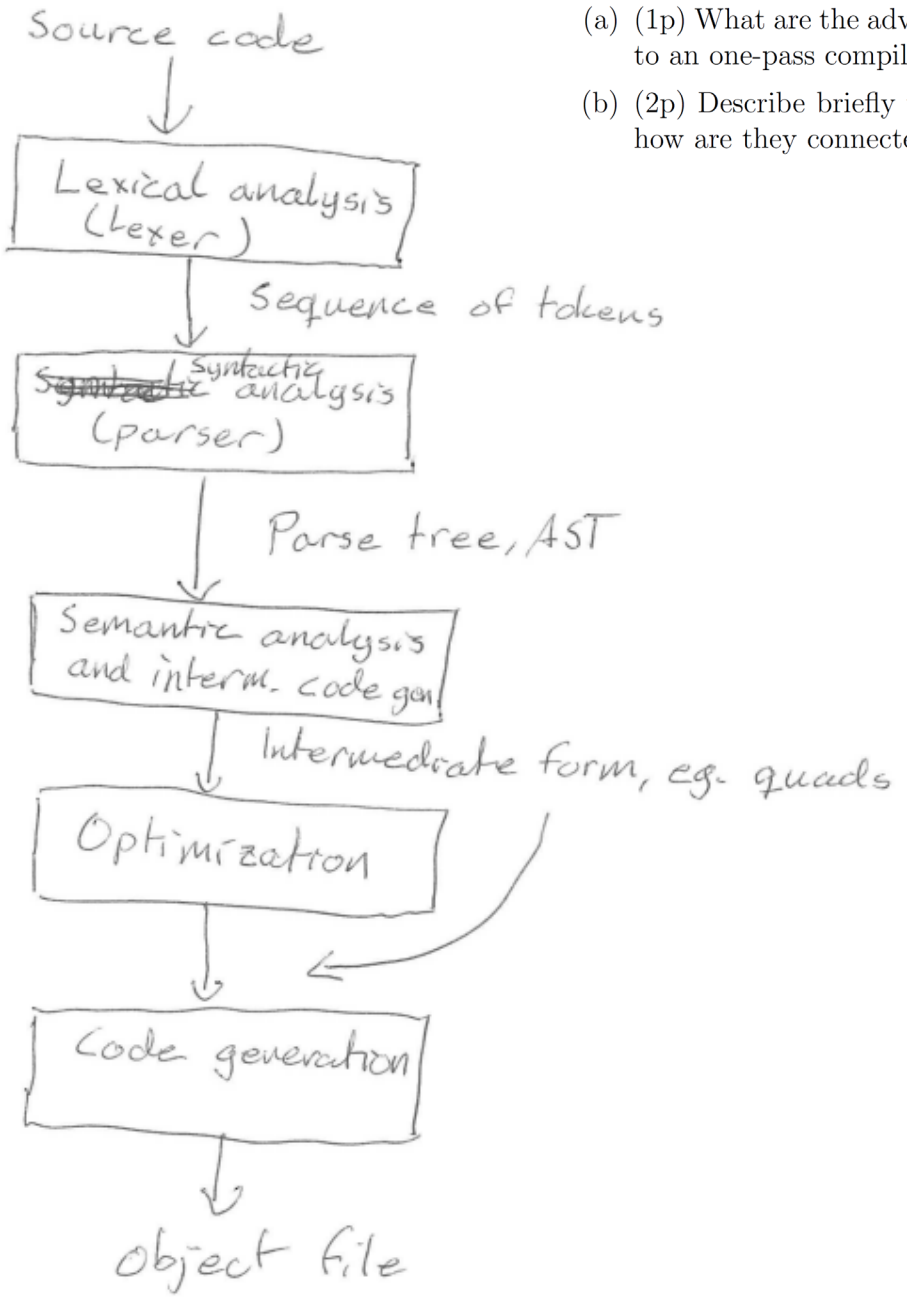
- Longer compile times and higher memory consumption

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).

Exercise 2

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

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?

Exercise 2

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

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

Write a RDP parser for the modified grammar

$L ::= L\ \alpha \mid E\ F\ \beta \mid F\ E\ \beta$

$E ::= E\ \gamma \mid \delta$

$F ::= E\ \psi \mid \epsilon$

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

```
Parse () {
  scan(); L1(); L2();
  if (token != EOF) error;
}

L1() { E(); E”() }

E” {
  if (token == ‘g’) {
    scan();
    E();
    if (token != ‘b’) error();
  } else {
    F();
    if (token != ‘b’) error();
  }
}

L2() {
  if (token == ‘a’) {
    scan();
    L2();
  }
}
```

```
E() {
  if (token == ‘d’) {
    scan();
    E’();
  } else {
    error();
  }
}

E’() {
  if (token == ‘c’) {
    scan();
    E’();
  }
}

F() {
  if (token=‘d’) {
    E();
    if (token != ‘g’) error();
  }
}
```

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

Input

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

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

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

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

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

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

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

$+ \alpha * \beta\$$

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:

   `A` ::= α`A` | `A`β | α`B`β | γ
   `B` ::= β`B` | `B`α | β`A`α | δ

   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.

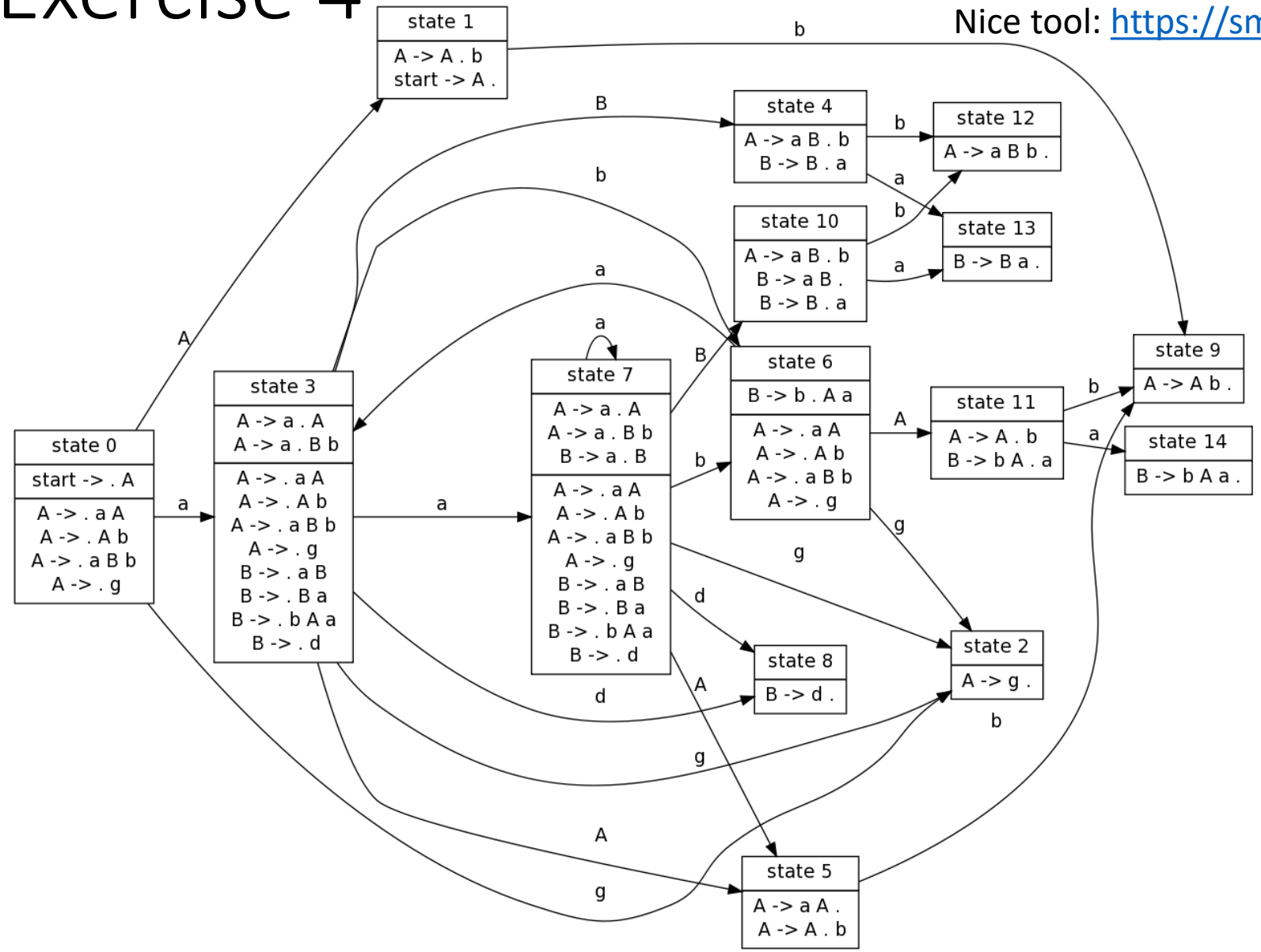
   If it is not: describe where/how the problem occurs.

$$A ::= \alpha A \mid A\beta \mid \alpha B \beta \mid \gamma$$
$$B ::= \beta B \mid B\alpha \mid \beta A \alpha \mid \delta$$

A ::= aA | Ab | aBb | g
B ::= aB | Ba | bAa | d

Exercise 4

Nice tool: https://smlweb.cpsc.ucalgary.ca/start.html



Grammar

A → a A
| A b
| a B b
| g.
B → a B
| B a
| b A a
| d.

nonterminal	first set	follow set
A	a g	b a
B	a b d	b a

Exercise 4

Nice tool: https://smlweb.cpsc.ucalgary.ca/start.html

Grammar

```
A → a A
  | A b
  | a B b
  | g.
B → a B
  | B a
  | b A a
  | d.
```

LR(0) Table

	$	d	a	b	g	A	B
0			s3		s2	s1	
1	acc	acc	acc	acc/s9	acc		
2	r(A → g)	r(A → g)	r(A → g)	r(A → g)	r(A → g)		
3		s8	s7	s6	s2	s5	s4
4			s13	s12			
5	r(A → a A)	r(A → a A)	r(A → a A)	r(A → a A)/s9	r(A → a A)		
6			s3		s2	s11	
7		s8	s7	s6	s2	s5	s10
8	r(B → d)	r(B → d)	r(B → d)	r(B → d)	r(B → d)		
9	r(A → A b)	r(A → A b)	r(A → A b)	r(A → A b)	r(A → A b)		
10	r(B → a B)	r(B → a B)	r(B → a B)/s13	r(B → a B)/s12	r(B → a B)		
11			s14	s9			
12	r(A → a B b)	r(A → a B b)	r(A → a B b)	r(A → a B b)	r(A → a B b)		
13	r(B → B a)	r(B → B a)	r(B → B a)	r(B → B a)	r(B → B a)		
14	r(B → b A a)	r(B → b A a)	r(B → b A a)	r(B → b A a)	r(B → b A a)		

The grammar is not LR(0) because

- shift/reduce conflict in state 1.
- shift/reduce conflict in state 5.
- shift/reduce conflict in state 10.

SLR(1) Table

	$	d	a	b	g	A	B
0			s3		s2	s1	
1	acc			s9			
2	r(A → g)		r(A → g)	r(A → g)			
3		s8	s7	s6	s2	s5	s4
4			s13	s12			
5	r(A → a A)		r(A → a A)	r(A → a A)/s9			
6			s3		s2	s11	
7		s8	s7	s6	s2	s5	s10
8			r(B → d)	r(B → d)			
9	r(A → A b)		r(A → A b)	r(A → A b)			
10			r(B → a B)/s13	r(B → a B)/s12			
11			s14	s9			
12	r(A → a B b)		r(A → a B b)	r(A → a B b)			
13			r(B → B a)	r(B → B a)			
14			r(B → b A a)	r(B → b A a)			

The grammar is not SLR(1) because

- shift/reduce conflict in state 5.
- shift/reduce conflict in state 10.

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;
    }
  }
}
```

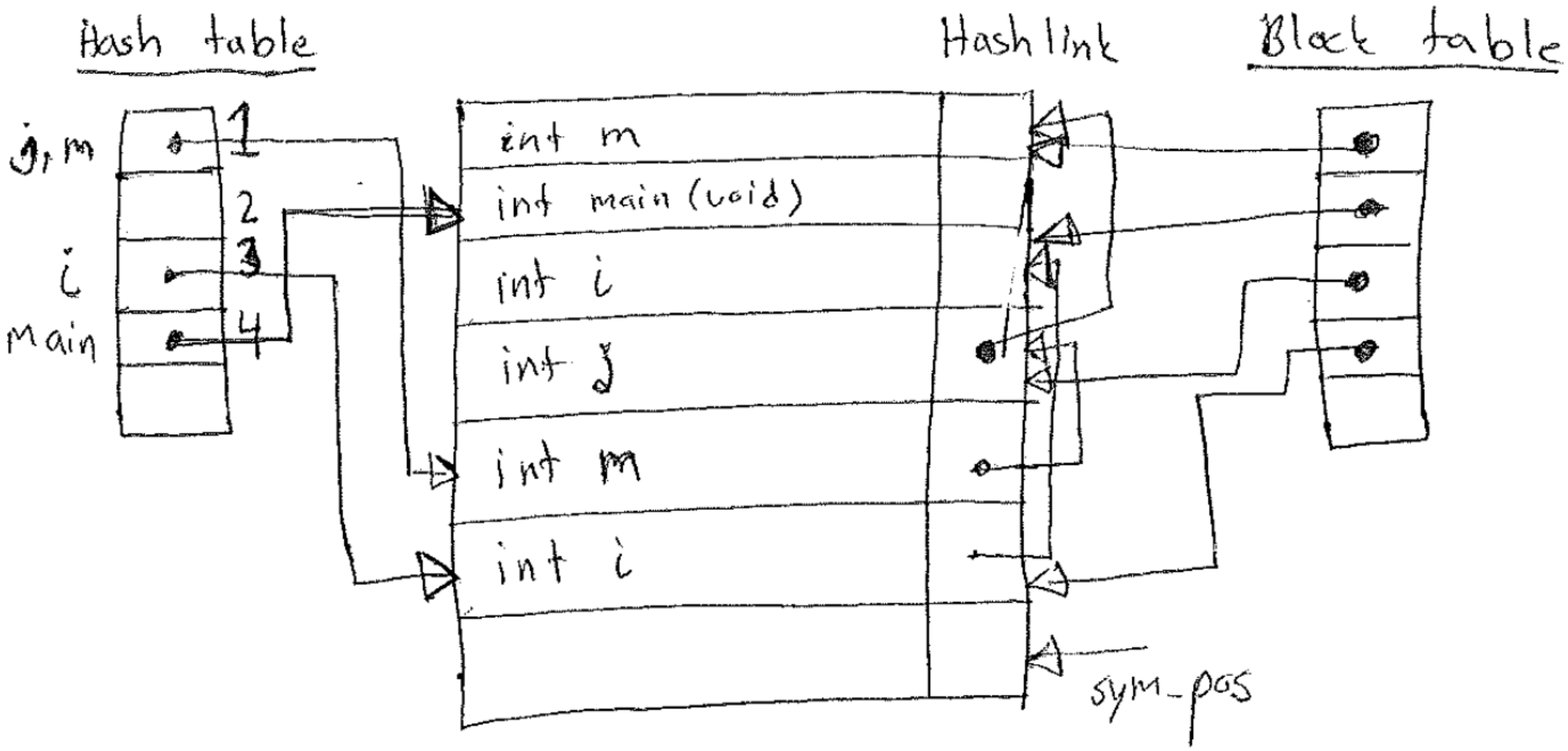
(a) (2p) For the program point containing the assignment `i = m * 2`, show how the program variables are stored in the symbol table if the symbol table is to be realized as a hash table with chaining and block scope control. Assume that your hash function yields value 3 for `i`, value 1 for `j` and `m`,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

See also:
https://www.ida.liu.se/~TDDE66/laboratories/instructions/lab2.html
https://www.ida.liu.se/~TDDE66/lessons/TDDE66_Exam_17_12_2009_Solutions1.pdf



Exercise 6

6. (5p) **Syntax-Directed Translation**

The Modelica programming language has an `if` statement defined according to the following grammar ({ } is repetition and [ ] is optional content):

```
<if-statement>  :
if <expression> then { <statement> ; }
{ elseif <expression> then
  { <statement> ; }
}
[ else
  { <statement> ; }
]
end if
```

Example:

```
if i<10 then
  ...;
elseif i<20 then
  ...;
elseif i<30 then
  ...;
else
  ...;
end if;
```

Write a syntax-directed translation scheme, with attributes and semantic rules, for translating the if-statements to quadruples.

The translation scheme should be used during bottom-up parsing. You are **not** allowed to define and use symbolic labels. Using global variables (except quads and index to the next quadruple) in the solution gives a deduction of (1p). You may need to rewrite the grammar. Explain all the attributes, functions, and instructions that you introduce. State all your assumptions. (Since it is a syntax-directed translation scheme, not an attribute grammar, generation of a quadruple puts it in an array of quadruples and attribute values are "small" values such as single quadruple addresses.)

Exercise 6

S → **if** E **then** S_1 **else** S_2

- See Lecture 8
  https://www.ida.liu.se/~TDDE66/lectures/PDF-OH2024/08-Semantics-AttribGrammar.pdf

Generate Quadruples for if-then-else (2)

- Factorised grammar:
  1. <ifstmt> ::= <truepart> S_2
  2. <truepart> ::= <ifclause> S_1 **else**
  3. <ifclause> ::= **if** E **then**

Attributes:

addr = address to the symbol table entry for result of E

quad = quadruple number

Generate quadruples for if-then-else (3)

3. <ifclause> ::= **if** E **then**

```
{ <ifclause>.quad = currentquad + 1;
  // save address p of jump over S1 for later in <ifclause>.quad
  GEN ( JEQZ,  E.addr,  0,     0  );
  // jump to S2.  Target q+1 not known yet.
}
```

2. <truepart> ::= <ifclause> S_1 **else**

```
{ <truepart>.quad = currentquad + 1;
  // save address q of jump over S2 for later
  GEN ( JUMP,  0,       0,     0 );
  // jump over S2.  Target r not known yet.
  QUADRUPLE[ <ifclause>.quad ][ 2 ] = currentquad + 1;
  // backpatch JEQZ target to q+1
}
```

3. <ifstmt> ::= <truepart> S_2

Generate Quadruples for if-then-else (4)

3. <ifclause> ::= **if** E **then**

   …

2. <truepart> ::= <ifclause> S_1 **else**

```
{ <truepart>.quad = currentquad + 1;
  // save address q of jump over S2 for later
  GEN ( JUMP,  0,       0,     0 );
  // jump over S2.  Target r not known yet.
  QUADRUPLE[ <ifclause>.quad ][ 2 ] = currentquad + 1;
  // backpatch JEQZ target to q+1
}
```

1. <ifstmt> ::= <truepart> S_2

```
{  QUADRUPLE[ <truepart>.quad ][ 1 ] = currentquad + 1;
   // backpatch JUMP target to (r-1)+1
}
```

Similarly: while statement, repeat statement …

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 (will insert missing “if” before the expression for: `expression then … else …`

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?

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:

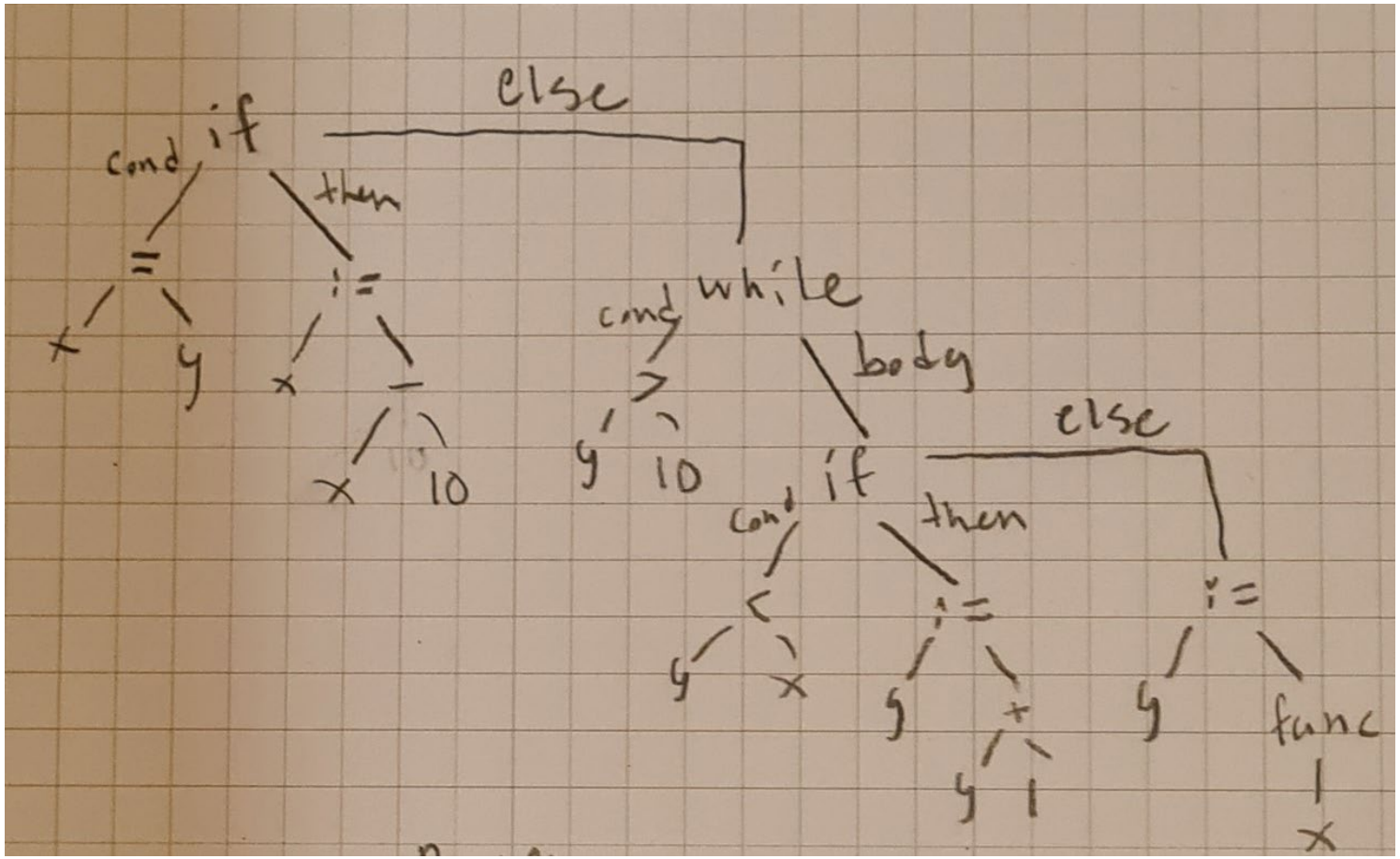
```
if x=y
  then x:=x-10
  else while y>10 do
    if y<x
      then y:=y+1
      else y:=func(x)
```

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

Exercise 9

Postfix:

```
x y = ELSE jfalse
x x 10 - :=
jmp IF_END
ELSE:
WHILE:
    y 10 < WHILE_END jfalse
    y x < ELSE2 jfalse
    y y 1 + :=
    jmp IF_END2
ELSE2:
    y x func :=
IF_END2:
WHILE_END:
IF_END:
```



Exercise 10

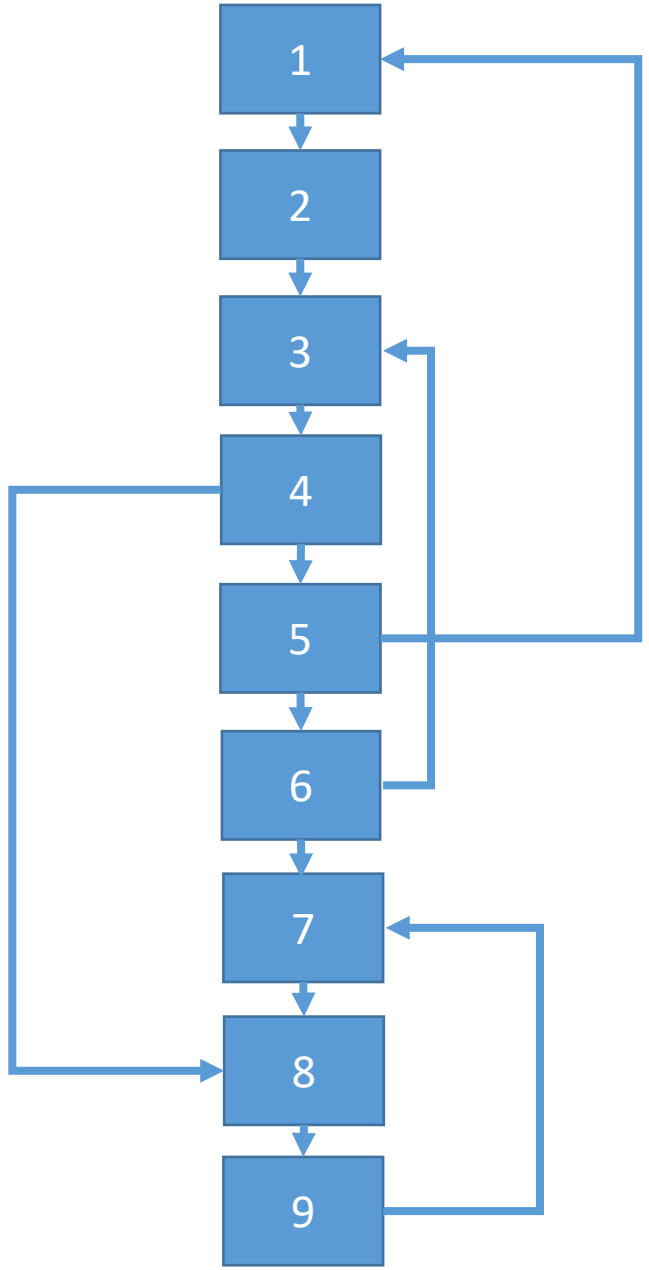
10. (3p) **Intermediate Code Generation**

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

Loops?

1-6? SEP? SC?

7-9? SEP? SC?

Check also previous exams with solutions

- https://www.ida.liu.se/~TDDE66/exam/

End

- Questions?