1. Fie specificația lexicală de mai jos:

$$abc \quad (1)$$
$$(bac^+|abc^+)^*a \quad (2)$$
$$(ac^*b|c^*ab)^+b \quad (3)$$

   (a) Care este secvența de reguli lexicale (1–3) utilizate în analiza următorului șir?

$$abcababbbaccabccaaabc$$

   (b) Dați un exemplu de șir care **nu** poate fi analizat de această specificație.

2. Fie gramatica de mai jos:

$$S \to bcS \mid a \mid Sb$$

   Câte încercări de producții sunt necesare, pentru ca strategia *recursive descent* să recunoască șirul `bcbcabb`? Pentru a vă da seama mai ușor dacă derivarea și șirul se încheie concomitent, puteți adăuga producția $S' \to S\$$ la gramatică, și \$ la finalul șirului, pentru a putea potrivi cei doi *token*-i \$ la final.

Subiectele 3–4 vizează gramatica de mai jos:

$$S \to TabU \mid Uc \mid \varepsilon$$
$$T \to bS \mid aT \mid c$$
$$U \to dS \mid \varepsilon$$

3. Determinați mulțimile *First* și *Follow*, construiți tabelul de analiză $LL(1)$ și concludeți dacă gramatica este într-adevăr $LL(1)$.

4. (a) Construiți automatul de analiză LR doar până întâlniți o stare cu conflicte $LR(0)$, și precizați tipul acestora și itemii implicați.

   (b) Precizați dacă și de ce dispar conflictele depistate mai sus, asumând euristica $SLR(1)$.

Subiectele 5–8 vizează programul Cool de pe **verso**.

5. Care este conținutul contextelor de tipare pentru obiecte, O, în punctul `P1`, respectiv pentru metode, M?

6. (a) Aplicați regula potrivită pentru verificarea tipului expresiei `not accepted(self@LazyList.contents())`, din metoda `contents` a clasei `FilteredLazyList`. Utilizați manualul Cool.

   (b) Păstrând restul codului neschimbat, este posibilă modificarea tipului atributului `next` de la `SELF_TYPE` la `LazyList`? Dar modificarea tipului metodei `init` de la `SELF_TYPE` la `LazyList` (independent de prima întrebare)?

7. (a) Descrieți reprezentarea în memorie a obiectelor prototip pentru clasele `LazyList`, `FilteredLazyList` și `Generator`, incluzând valorile implicite ale atributelor și tabelele de metode.

   (b) Presupunând că toate informațiile necesare evaluării corpurilor metodelor se depun pe stivă, care este dimensiunea minimă a înregistrării de activare pentru metoda `accepted` din clasa `FilteredLazyList`?

8. Pentru expresia `case`, din metoda `contents` a clasei `LazyList`, este propusă următoarea secvență de cod MIPS, unde registrul `$s0` conține adresa lui `self`, iar convențiile de organizare a înregistrării de activare sunt cele din enunțul temei 3. Se consideră că `Object` are *tag*-ul 0, celelalte 4 clase predefinite, *tag*-uri între 1 și 4, iar celelalte clase, *tag*-urile din comentariile din program. Completați cele 5 spații libere cu valorile corecte și justificați!

```
1     lw      $a0 __($s0)
2     <verificare case on void>
3  case:
4     sw      $a0 -4($fp)
5     lw      $t1 0($a0)
6     blt     $t1 __ casebranch0
7     bgt     $t1 __ casebranch0
8     <cod aferent ramurii>
9     b       endcase
10 casebranch0:
11    blt     $t1 __ casebranch1
12    bgt     $t1 __ casebranch1
13    <cod aferent ramurii>
14    b       endcase
15 casebranch1:
16    lw      $a0 -4($fp)
17    jal     _case_abort
18 endcase:
```

9. Descrieți, pas cu pas, aplicarea strategiei *local value numbering* asupra *basic block*-ului de mai jos. Completați tabelul valorilor și indicați cum se modifică fiecare instrucțiune, unde este cazul.

```
1 a := 1                5 e := c + 2
2 b := x + a            6 f := e * a
3 c := a                7 g := f - 3
4 d := c + x
```

Subiectele 10–12 vizează codul intermediar de mai jos, unde doar `d` este *live* la final:

```
1  L0:                      9  L1:
2      a := 1                10     b := a + 2
3      if b > 0 goto L1      11     c := 4
4  L2:                      12 L3:
5      b := a + 1            13     d := b * c
6      c := b * 2            14     a := 2
7      b := 3                15     if d > 2 goto L1
8      jump L3
```

10. Fără alte optimizări, menționați variabilele *live* în **fiecare** punct al secvenței. Alocați numărul **minim** de registre necesare pentru evitarea *spilling*-ului.

11. (a) Determinați **dominatorii** fiecărui bloc din CFG.

   (b) Desenați **arborele de dominanță**.

   (c) Determinați **frontiera de dominanță** a fiecărui bloc.

12. (a) Utilizând informațiile de **dominanță** și ***liveness***, pentru ce variabile și în ce blocuri trebuie introduse instrucțiuni ϕ, pentru a obține **forma SSA minimală** a CFG-ului?

   (b) Redesenați CFG-ul cu instrucțiunile ϕ introduse și toate variabilele **redenumite** corespunzător.

   (c) Aplicați repetat optimizări pe CFG-ul în forma SSA, inclusiv prin instrucțiuni ϕ, și eliminați codul mort, până când nu se mai schimbă nimic.

```
class LazyList (* tag 6 *) {
    contents : Object;
    next : SELF_TYPE;

    init(generator : Generator) : SELF_TYPE {{
        contents <- generator;
        self;
    }};

    contents() : Object {
        case contents of
            element   : Object    => contents;
            generator : Generator => {
                next     <- new SELF_TYPE.init(generator);
                contents <- generator.generate();
            };
        esac
    };

    next() : SELF_TYPE {
        if isvoid next then {
            abort();
            self;
        }
        else
            next
        fi
    };
};

class FilteredLazyList (* tag 7 *) inherits LazyList {
    accepted(o : Object) : Bool {
        case o of
            s : String => false;
            i : Int    => 2 * (i / 2) = i;
        esac
    };

    contents() : Object {
        let oldContents : Object <- contents in {
            (* P1 *)
            while not accepted(self@LazyList.contents()) loop
                contents <- oldContents
            pool;
            contents;
        }
    };
};

class Generator (* tag 8 *) {
    n : Int;

    generate() : Object {
        let oldN : Int <- n in {
            n <- n + 1;
            oldN;
        }
    };
};

class Main (* tag 5 *) {
    main() : Object { 0 };
};
```

State Andreea-Rebecca
341C3

Total 89.95
1 - 10p
2 - 9p
3 - 9.5p
4 - 10p
5 - 9.5p
6 - 7.2p
7 - 9p
8 - 0p
9 - 10p
10 - 9.7p
11 - 6p
12 - 0p

Examen CPL

1) (1) abc
(2) $(bac^+ | abc^+)^*a$
(3) $(ac^*b | c^*ab)^+b$

a) a b c a b a b b | b a c c a b c c a | a | a | a b c |
3 2 2 1

Reguli: (1) maximal munch
(2) dacă 2 reg. cuprind același subșir, se utilizează prima reg definită

b) Șirul „c" nu poate fi analizat

2) $S \to bcS \mid a \mid Sb$

bcbcabb

$S \to bcS \to bcbcS \to bcbcbcS$ ×
$\to bcbca$ ×
$\to bcbcSb \to bcbcbcSb$ ×
$\to bcbcaab$ ×
$\to bcbcSbb \to bcbcbcSbb$ ×
$\to bcbcabb$ ✓

⇒ După 9 încercări s-a ajuns la șirul dorit

3) $S \to TabU \mid Uc \mid \varepsilon$

$T \to bS \mid aT \mid c$

$U \to dS \mid \varepsilon$

	FIRST
S	ε, a, b, c, d
T	b, a, c
U	d, ε

	FOLLOW
S	$\$, c, a$
T	a
U	$c, \$, a$

$First(S) \supseteq \{\varepsilon\}$

$First(T) \supseteq \{b, a, c\}$

$First(U) \supseteq \{d, \varepsilon\}$

$First(S) \supseteq First(TabU) = First(T) = \{a, b, c\}$

$First(S) \supseteq First(Uc) = First(U) \setminus \{\varepsilon\} \cup First(c) = \{d, c\}$

$\$ \in Follow(S)$

$Follow(T) \supseteq First(a) = \{a\}$

$Follow(U) \supseteq Follow(S)$ (1)

$Follow(U) \supseteq First(c) = \{c\}$

$Follow(S) \supseteq Follow(U)$ (2)

$Follow(S) \supseteq Follow(T) = \{a\}$

Din (1) și (2) $\Rightarrow Follow(U) = Follow(S) = \{c, \$, a\}$

	a	b	c	d	$
S	ε / TabU	TabU	ε / Uc	Uc	ε
T	aT	bS	c		
U	ε		ε	dS	ε

→ Nu este LL(1)

4) a)

$S' \to \cdot S$

$S \to \cdot TabU$

$S \to \cdot Uc$

$S \to \cdot$ (R)

$T \to \cdot bS$ (S)

$T \to \cdot aT$ (S)

$T \to \cdot c$ (S)

$U \to \cdot dS$ (S)

$U \to \cdot$ (R)

$T \to \cdot bS$, $S \to \cdot$ } conflict SR (1)

$T \to \cdot aT$, $S \to \cdot$ } conflict SR (2)

$T \to \cdot c$, $S \to \cdot$ } conflict SR (3)

$U \to \cdot dS$, $S \to \cdot$ } conflict SR (4)

$T \to \cdot bS$, $U \to \cdot$ } conflict SR (5)

$T \to \cdot aT$, $U \to \cdot$ } conflict SR (6)

$T \to \cdot c$, $U \to \cdot$ } conflict SR (7)

$U \to \cdot dS$, $U \to \cdot$ } conflict SR (8)

$S \to \cdot$, $U \to \cdot$ } conflict RR (9)

b) Pp euristica SLR(1) pt conflictul:

(1) se verif. $b \in Follow(S)$ și nu aparține => nu este conflict

(2) se verif $a \in Follow(S)$ și aparține => este conflict

(3) se verif $c \in Follow(S)$ și aparține => este conflict

(4) se verif $d \in Follow(S)$ și nu aparține => nu este conflict

(5) se verif $b \in Follow(U)$ și nu aparține => nu este conflict

(6) se verif $a \in Follow(U)$ și aparține => este conflict

(7) se verif $c \in Follow(U)$ și aparține => este conflict

(8) se verif $d \in Follow(U)$ și nu aparține => nu este conflict

(9) $Follow(S) \cap Follow(U) \neq \emptyset \Rightarrow \exists t$ a.î. $t \in Follow(S)$ și $t \in Follow(U)$

=> nu știu ce să reduc, deci se păstrează conflictul (3)

g)

$a := 1 \mid a = 1$

$b := x + a \mid b = x + 1$

$c := a \mid c = 1$

$d := c + x \mid d = 1 + x = b$

$e := c + 2 \mid e = 3$

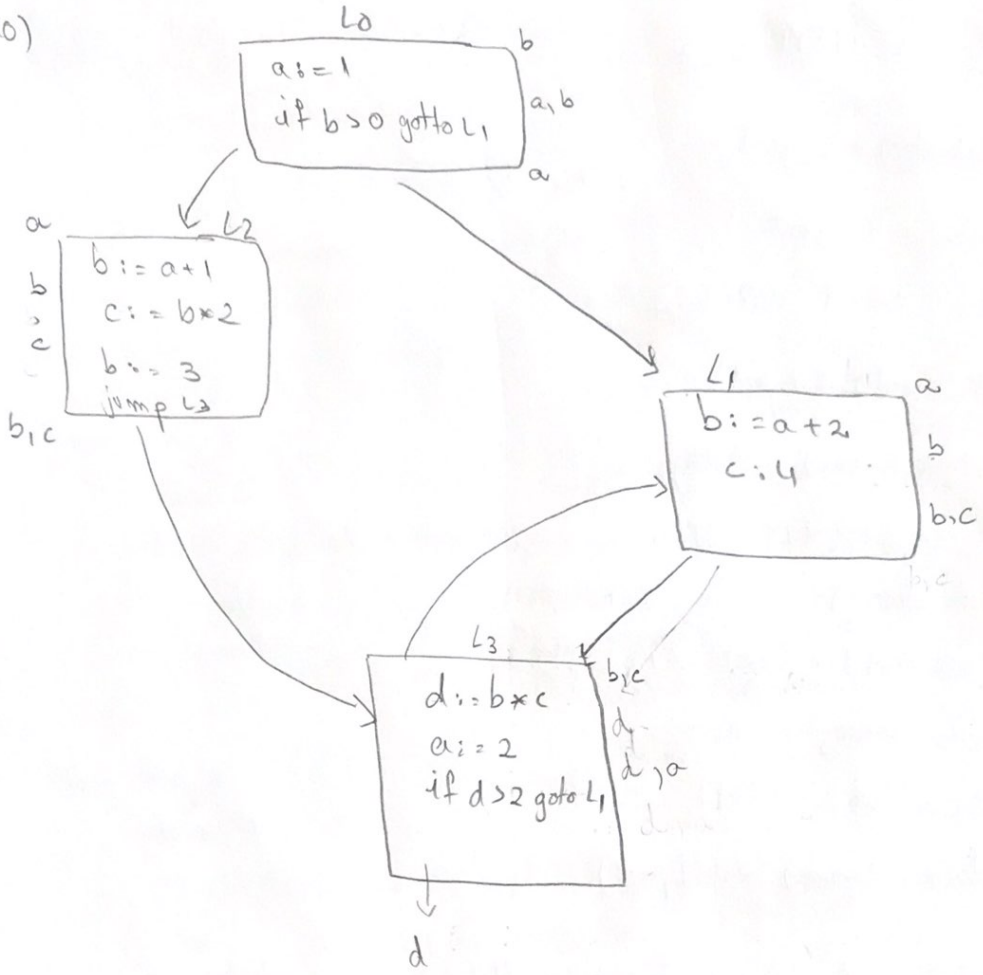
$f := e * a \mid f = 3$

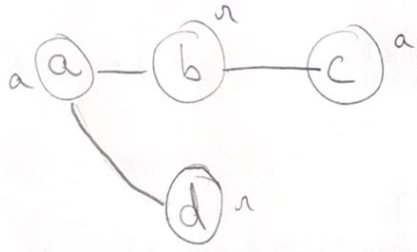
$g := f - 3 \mid g = 0$

$\#0 + \#1 \xrightarrow{\text{sort}} \#1 + \#0$

#	value	canonic
0	1	1
0	a	1
1	x	x
2	#1 + #0	b
0	c	1
2	d	b
3	2	2
4	#0 + #3	3
4	# 3	3
4	e	3
4	#4 * #0	3
4	# f	3
5	#4 - #4	0
5	0	0
5	g	0

10)





k = 2 => avem nevoie de minim 2 registri

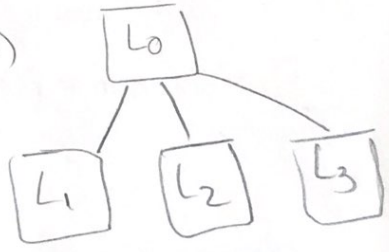
11) a) $L_0 : L_0$

$L_1 : L_1, L_0$

$L_2 : L_2, L_0$

$L_3 : L_3, L_0$

b)



5) O(self) = SELF_TYPE FLZ

FLZ = FilteredLazyList
LZ = LazyList

Notatii

O(oldContents) = Object

O(contents) = Object

O(next) = SELF_TYPE FLZ

M(Object), abort) = Object)

M(Object, type_name) = (String)

M(Object, copy) = (SELF_TYPE)

M(IO, out_string) = (String, SELF_TYPE)

M(IO, out_int) = (int, SELF_TYPE)

M(IO, in_string) = (String)

M(IO, in_int) = (int)

M(Main, main) = (Object)

M(FLZ, contents) = (Object) = M(LZ, contents)

M(FLZ, accepted) = (Object, Bool)

M(LZ, next) = (SELF_TYPE)

M(Generator, generate) = (Object)

M(LZ, init) = (Generator, SELF_TYPE)

7) a) LZ -> LazyList
FLZ -> FilteredLazyList
G -> Generator

LZ: tag 0
dim 5
LZ.dispTab → Object -- x3
contents → LZ.init
next → LZ.contents

FLZ: tag 1
dim 5
FLZ.dispTab → Object -- x3
contents → LZ.init
next → FLZ.contents
FLZ.accepted

G: tag 2
dim 4
G.dispTab → Object -- x3
n → G.generate

b) ① registrii : $fp, $so, $ra -> 3

② param : 1

③ locații temporare : 2

A = max(NT(1), 1+NT(2)) = max(0,1) = 1
max(NT(2), NT(A)+1) = max(0,2) = 2

=> dim. minimă este ①+②+③ = 6 cuvinte

6) a) not accepted(self@LZ.contents()) are type Bool

LZ = LazyList
FLZ = FilteredLazyList

accepte(self@LZ.contents()) = self.accepted(self@LZ.contents())

$O, M, FLZ \vdash self : SELF\_TYPE_{FLZ}$

$O, M, FLZ \vdash self@LZ.contents() : Object$

$M(FLZ, accepted) = (Object, Bool)$

$T_{u+1} = Bool$

[Dispatch]

$O, M, FLZ \vdash self.accepted(self@LZ.contents()) : Bool$

$O, M, FLZ \vdash self : SELF\_TYPE_{FLZ}$

$SELF\_TYPE_{FLZ} \leq LZ$ ✓

$M(LZ, contents()) = (Object)$

$T_{u+1} = Object$

[Static Dispatch]

$O, M, FLZ \vdash self@LZ.contents : Object$

$O, M, FLZ \vdash self.accepted(self@LZ.contents()) : Bool$

[Not]

$O, M, FLZ \vdash not\ accepted(self@LZ.contents()) : Bool$