xPJP, vybraná vzorová zkoušková zadání z minulých let

Jde pouze o orientační přehled, v konkrétní zkoušce se mohou objevit zcela odlišné příklady

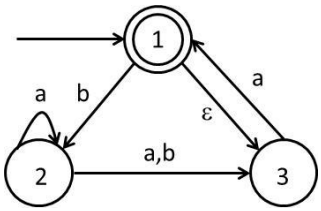
Veškerá zadání jsou bez záruky, tj. mohou obsahovat neúmyslné chyby či překlepy

---

Zařaďte jednotlivé funkce do odpovídajících standardních návrhových fází překladače:

	FUNKCE		**FÁZE**
A	Přiřazení proměnné ke konkrétnímu registru.	1	Lexikální analýza
B	Zjištění adres, příslušejících podmíněnému cyklu.	2	Syntaktická analýza
C	Úprava výrazu *A +4*3* na *A + 12.*	3	Sémantická analýza
D	Nalezení nedeklarované proměnné.	4	Optimalizace
E	Změna příkazu *A:=A+12* na *ADD #12, A.*	5	Příprava na generování kódu
F	Vytvoření syntaktického stromu.	6	Generování kódu

Napište nejméně tři důvody, kvůli kterým je následující konečný automat nedeterministický:



Odstraňte levou rekurzi z následující gramatiky. V jaké souvislosti tuto úpravu nejčastěji používáme?

A --> A X | Y
X --> b | c
Y --> d | f

Je gramatika na následujícím řádku jednoznačná? Své tvrzení zdůvodněte a ilustrujte příkladem!

E --> E + E | E - E | E * E | E / E | (E) | id

Mějme gramatiku G s pravidly (1) – (4) a související překladovou tabulku:

(1) S --> a A S

(2) S --> b A

(3) A --> c A

(4) A --> d

	a	**b**	**C**	**d**	**$**	**S**	**A**
0	S2	S3				**1**	
1					**OK**		
2			S5	S6			**4**
3				S6			**7**
4	S2	S3				**8**	
5			S5	S6			**9**
6		R4			R4		
7					R2		
8					R1		
9		R3			R3		

Na základě těchto podkladů trasujte větu *acdbd* ve třísloupcové notaci (vstup, zásobník, akce).

Navrhněte (i) sémantický strom a (ii) čtveřice pro příkaz:

bohaty = (majetek > milion) AND NOT pujcka

za předpokladu platnosti standardních pravidel pro asociativitu a prioritu operátorů.

Slovně popište a nejméně třemi příklady ilustrujte regulární výrazy, odpovídající následující LEX_ové specifikaci:

\"[^"]*(\"\"[^"]*)*\"

Určete množiny FIRST a FOLLOW pro následující gramatiku. Jaké z Vašeho zjištění plynou závěry?

S -> A B c

A -> a | *epsilon*

B -> b | *epsilon*

Mějme gramatiku G s pravidly (1) – (8) a související překladovou tabulku:

(1) E -> T E'
(2, 3) E' -> + T E' | *epsilon*
(4) T -> F T'
(5, 6) T' -> * F T' | *epsilon*
(7, 8) F -> (E) | id

	+	*	(	)	id	$
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

a/ O jakou gramatiku a typ překladu se jedná?
b/ Trasujte vstup *id*(id+id)* ve třísloupcové notaci (zásobník, vstup, výstup).

Popište funkci následující atributové gramatiky. Ve které fázi konstrukce překladače se tato pravidla uplatní?

E -> E1 + T *if ((E1.type == int) && (T.type == int)) E.type = int else E.type = real*
| T *E.type = T.type*
T -> num . num *T.type=real*
| num *T.type=int*

Za předpokladu platnosti standardních pravidel pro asociativitu a prioritu operátorů převeďte výraz
a*-(b+c)
převeďte do následujících forem mezikódu (vnitřní reprezentace, intermediální kód):

a) Abstraktní syntaktický strom
b) Postfixová notace
c) Tříadresový kód

Jakým způsobem organizuje (segmentuje) překladač paměť za běhu programu? Víte, co obvykle bývá příčinou *runtime* chybového hlášení *Stack overlow*?

Zodpovězte následující otázky:

a) Jaký typ rozkladu realizuje syntaktický analyzátor, pracující shora dolů?
b) Máte-li syntaktický analyzátor shora dolů a na vrcholu zásobníku řetězec *αABC*, jaké pravidlo musí obsahovat generující gramatika, aby se jeho obsah v následujícím kroku změnil?
c) Doplňte chybějící výrazy: vstupem nástroje *yacc* je _____.a jeho výstupem je _____ .
d) Jak se nazývá „výpočetní stroj“, umožňující zpracovávat regulární výrazy?

Odstraňte levou rekurzi z následující gramatiky. Proč se takovou činností vůbec zabýváme?

X -> X c | a | b

Popište funkci následující atributové gramatiky. Ve které fázi konstrukce překladače se tato pravidla uplatňují?

E -> E1 + T *if ((E1.type == int) && (T.type == int)) E.type = int else { E.type = real;}; print +*
| T *E.type = T.type*

T -> num . num *T.type=real; print T.val*
| num *T.type=int; print T.val*

Za předpokladu platnosti standardních pravidel pro asociativitu a prioritu operátorů převeďte výraz:

A and (B or not (C and D))

do následujících forem mezikódu (= vnitřní reprezentace, intermediální kód):

a) Abstraktní syntaktický strom („sémantický strom“)
b) Tříadresový kód v libovolné formě reprezentace

Optimalizujte následující základní blok kódu. Výsledek vyjádřete buď textově nebo graficky. Co je to základní blok?

t1 = b + c
t2 = 7 + d
t3 = t1 * t2
t4 = b + c
t5 = 7 + d
t6 = t4 * t5
t7 = t3 + t6
d = t7

Napište regulární výraz, pracující s množinou argumentů {a,b} a definující pro ně následující operátory (následující výpis je s klesající prioritou):

- Postfixový **'*'** pro 0 nebo více opakování,
- Infixový **'.'** pro konkatenaci (spojování řetězců, AND),
- Infixový **'|'** pro alternaci (OR).

Regulární výraz musí dále umožňovat/detekovat párování závorek **'(' ')'** pro seskupování regulárních výrazů na základě uživatelské priority.

Mějme gramatiku s následujícími pravidly:

```
S -> E '#' E   | E '$' E  | E
E -> E '!' T   | T
T -> F '%' T   | F
F -> '(' S ')' | '0' | '1' ... | '9'
```

Charakterizujete aritu, asociativitu a prioritu všech jejích operátorů.

Pro gramatiku:

```
1.   S -> S ; S
2.   S -> id := E
3.   S -> print ( L )
4.   E -> id
5.   E -> num
6.   E -> E + E
7.   E -> (S , E)
8.   L -> E
9.   L -> L , E
```

sestrojte LL(1) překladovou tabulku. Jaké kroky musí v tomto případě Váš návrh zahrnovat?

Gramatiku

```
expr → expr + term
expr → expr - term
term → term / factor
term → factor
digit → <číslo>
```

doplňte akcemi, realizujícími převod vstupního infixového výrazu do postfixové formy. Pro výraz:

1+2/3

nakreslete odpovídající plně anotovaný syntaktický strom (struktura + akce + hodnoty).

Minimalizujte konečný automat, určený tabulkou:

	a	b
A	A	B
B	E	C
C	C	D
D	F	C
E	D	E
F	B	E

Ke gramatice:

```
S -> x | ( S R )
R -> , S  R |  epsilon
```

sestrojte LL(1) překladovou tabulku a na jejím základě prověřte syntaxi vstupního výrazu (x,x).

Pro kód:

```
while (c) {
   x = y + 1;
   y = 2 * z;
   if (d)  x = y+z;
   z = 1;
}
z = x;
```

sestrojte graf řízení (control flow graph). K čemu se tento typ grafu v konstrukci překladačů používá?

V následující atributové gramatice:

```
Pravidla                                || Semantické akce
========================================++====================
0:  Number -> Sign List                 || (1) List.Scale := 0
                                        || (2) Number.Value :=
                                        ||      IF Sign.Neg THEN -List.Value
                                        ||      ELSE List.Value
1:  Sign -> +                           || (1) Sign.Neg := False
2:  Sign -> -                           || (1) Sign.Neg := True
3:  List -> BinaryDigit                 || (1) BinaryDigit.Scale = List.Scale
                                        || (2) List.Value := BinaryDigit.Value
4:  List(0) -> List(1) BinaryDigit      || (1) List(1).Scale =
                                        ||       List(0).Scale + 1
                                        || (2) BinaryDigit.Scale = List.Scale
                                        || (3) List(0).Value := List(1).Value
                                        ||       + BinaryDigit.Value
5:  BinaryDigit -> 0                    || (1) BinaryDigit.Value = 0
6:  BinaryDigit -> 1                    || (1) BinaryDigit.Value =
                                        ||       2^(BinaryDigit.Scale)
```

správně identifikujte typy všech atributů. Jaká je její funkce?

Navrhněte deterministický konečný automat, který přijímá jazyk, definovaný regulárním výrazem: `(ab*c)|(abc*)`.

Na základě výpočtu kompletní sady kontextových množin zdůvodněte, zda je následující gramatika LL(1):

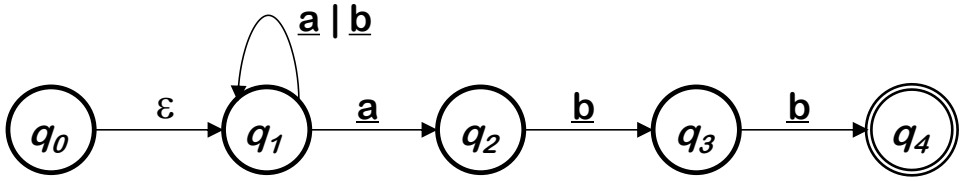
```
S --> ABa
A --> bA | ε
B --> aB | ε
```

V každém případě pro tuto gramatiku zkonstruujte LL překladovou tabulku a s její pomocí analyzujte vstup bbaa .

Navrhněte graf řízení (control flow graph) pro následující kódový fragment:

```
if (x < y)
      if (x < z)
            nejmensi = x;
      else
            nejmensi = z;
else
      if (y < z)
            nejmensi = y;
      else
            nejmensi = z;
return nejmensi;
```

Následující nedeterministický konečný automat převeďte na deterministický:



Napište regulární výraz, který je tímto automatem přijímán.

Na základě této LR(1) překladové tabulky:

	','	'('	')'	'['	']'	id	'$'	B	P	E
0						s2		1		
1							acc			
2	r3	s5	r3	s4	r3		r3		3	
3	r1		r1		r1		r1			
4						s2		7		6
5						s2		7		8
6					s9					
7	s10		r5		r5					
8			s11							
9	r2		r2		r2		r2			
10						s2		7		12
11	r4		r4		r4		r4			
12			r6		r6					

zodpovězte následující dotazy:

a) Jak se jmenuje startovací neterminál zdrojové gramatiky a proč?
b) Který její neterminál implementuje nejvyšší prioritu a proč?
c) Kolik má gramatika celkem pravidel a jak jste to zjistili?
d) Obsahuje překladová tabulka nějaké konflikty? Pokud ano, jaké? Pokud ne, svůj závěr zdůvodněte!
e) Napište nejméně čtyřznakovou úvodní sekvenci korektního programu, generovaného touto gramatikou.
f) Co se stane, když se ve stavu 2 objeví na vstupu id?
g) Co všechno se stane, když se ve stavu 7 objeví na vstupu libovolná uzavírací závorka?
h) Čím je zajímavý stav 3 a jak byste tuto zajímavost slově interpretovali?

Rozhodněte, zda je následující gramatika vhodná pro překlad shora dolů:

S --> ABd | aBc
A --> ε
B --> b | c

Pokud ne, pokuste se ji upravit na vhodný tvar a navrhněte kostru překladače, pracujícího rekurzivním sestupem (RD). Pokud vhodná je, přejděte rovnou k návrhu RD překladače.

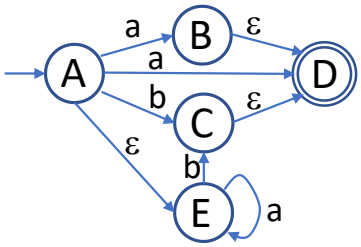
Gramatiku:

E --> E+E
E --> num E.val = num.yylval
E --> (E) E.val = E1.val

doplňte sémantickými pravidly v libovolném tvaru a do odpovídajícího atributy plně anotovaného syntaktického grafu zakreslete řešení vstupního výrazu:
2 + (7+3)

Pomocí rozšířených T-diagramů (thumbstones) znázorněte procesy překladu a spuštění aplikace APP v HW prostřední procesoru Intel (x86) za předpokladu, že zde máte k dispozici nativní překladač jazyka, ve kterém je aplikace napsána.

a) Je konečný automat na obrázku deterministický? Své tvrzení zdůvodněte.
b) Pokud je, minimalizujte ho. Pokud není, převeďte ho na deterministický.
c) Najděte regulární výraz, ekvivalentní k tomuto automatu.



Mějme gramatiku:

T --> R | aTc *(1, 2)*
R --> ε | bR *(3, 4)*

a jí odpovídající překladovou tabulku:

	a	b	c	$	*T*	*R*
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

Na základě těchto informací zkontrolujte syntaxi vstupní věty `aabbbcc`

Přiřazovací příkaz:

```
x := a+b*a
```

vyjádřete:

a) Abstraktním syntaktickým stromem
b) Tříadresovým kódem
c) Zásobníkovým kódem

Pro každý typ mezikódu uveďte jeho výhodu ve srovnání s ostatními.

Upravte následující gramatiku tak, aby byla vhodná pro překlad shora dolů:

```
C --> if E then S else S | if E then S
```

Následující gramatickou specifikaci pro *yacc* rozšiřte o operátor unární mínus a zachovejte přitom všechna obvyklá prioritní a asociativní pravidla:

```
%term NUM
%left '+' '-'
%left '*' '/'
%%
e : e '+' e | e '-' e
  | e '*' e | e '/' e
  | '(' e ') ' | NUM
```

Na základě znalosti běžné víceznačné gramatiky pro aritmetické výrazy, používané například generátorem *yacc*, nakreslete úplný syntaktický strom (parse tree) a jeho redukovanou verzi, používanou jako jedna z forem mezikódu (také abstraktní syntaktický strom nebo sémantický strom) pro vstup:

```
(a + b) * c
```

Optimalizujte následující základní bloky:

a)	b)	c)

`x = 5 + 2;`	`j = j - 1;`
	`k = 4 * j;`
		`while ( a < b ) {`
		`  c = 10 * d;`
		`  a = a + c;`
		`}`

Slovně charakterizujte funkci následující gramatiky. Jakého je typu, jaké má operátory a jak je to s jejich prioritou a asociativitou? Napište alespoň dva strukturně odlišné řetězce, které je schopna generovat a využijte v nich všech zavedených operátorů.

Uměli byste napsat odpovídající regulární výraz? Pokud ano, učiňte tak.

```
S -> R $
R -> R '|' T | T
T -> T '.' F | F
F -> F '*' | B
B -> '0' | '1' | '(' R ')'
```