1. **Charakterizujte pojem křížový překladač**
   Překlad na jiném procesoru než exekuce – zabudované systémy (např. pračky, myčky, automobily). Generuje kód pro jiný počítač než na kterém se provádí překlad.

2. **Objasněte pojem silikonový překladač**
   Pro návrh integrovaných obvodů. Proměnné nereprezentují místo v paměti, ale log. proměnnou obvodu. Výstupem je návrh obvodu. (Překlad zdrojového kódu do hradel, atp. - např. ABel)

3. **Co jsou to formátory textu? Uveďte příklad**
   Překladače pro sazbu textu (TEX→DVI). Upravování podoby textu podle požadavků uživatele.

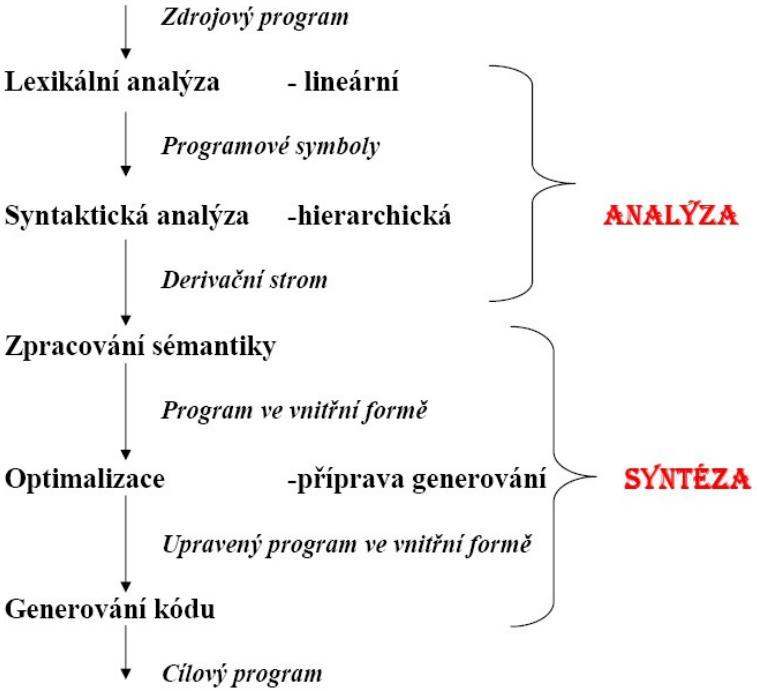
4. **Charakterizujte kaskádní překladač**
   Překládáme z jazyka A do jazyka C, nemáme překladač, ale máme překladač z A do B. Tak si uděláme překladač z B do C (nebo ho už také máme) a přeložíme (A → B → C). Obtížné ladění.

5. **Porovnejte výhody a nevýhody interpretačních a kompilačních překladačů**
   Interpret:
   - *vnitřní jazyk – lepší přechod na jiný stroj (např. byte kód u Javy)*
   - *snazší debugging*
   - *pomalejší kompilace – nutnost opakovaného překladu u opakujících se sekvencí (for)*

   Kompilátor:
   - *Okamžitý překlad do strojového jazyka*
   - *Rychlý běh programu – rychlá kompilace (překlad)*
   - *Složitější debugging – ladění, odstraňování chyb, krokování*

6. **Jaké jsou důvody pro použití víceprůchodového překladače?**
   - Překlad probíhá ve více fázích – dříve kvůli paměti – jen jedna část překladače je v paměti.
   - Snadněji jde rozdělit práci na překladači mezi více programátorů
   - Algoritmy pro optimalizaci jsou často složité, mají proto vlastní průchod, často i více průchodů.
   - zajímají nás meziprodukty – mezipřeklady:
   1. *Průchod → 1. meziprodukt → ... → N. průchod →* ***finální produkt***
   - *lze optimalizovat překládaný program – kvůli dopředným skokům*

7. **Zdůvodněte, proč se nepoužívají čistě interpretační překladače**
   Program prováděný čistě interpretačním překladačem je prováděn mnohem pomaleji něž odpovídající cílový program vytvořený kompilátorem. Nevhodné pro programy, kde je výpočet časově náročný. Proto Java tuto nevýhodu odstraňuje pomocí bytecode.



8. **Co to jsou generátory překladačů, uveďte příklad**
   Programy, které na základě symbolického popisu (např. pomocí gramatiky) vytvoří zdrojový kód překladače. Např. JavaCC, bison, yacc

9. **Nakreslete schéma překladače kompilačního typu**
   - ***lexikální analyzátor*** *– posloupnost symbolů programu*
   - ***syntaktický analyzátor*** *– derivační strom*
   - ***zpracování sémantiky*** *– program ve vnitřní formě*
   - ***optimalizace*** *– cyklů, redukce počtu registrů, …*
   - ***generátor cílového programu*** *– cílový program*

10. Jaká vlastnost gramatiky podmiňuje nekonečnost generovaného jazyka

Rekursivnost gramatiky. Např.: A -> aA

11. Popište gramatikou reálná čísla s desetinnou částí

$C \rightarrow cC_1$

$C_1 \rightarrow cC_1 \mid .C_2 \mid e$

$C_2 \rightarrow cC_2 \mid e$

12. Jaký je nejvyšší možný počet stavů deterministického KA, má-li ekvivalentní nedeterministický KA 5 stavů?

Počet stavů = $2^N - 1 \rightarrow 2^5 - 1 =$ ***31***

13. Popište tvar identifikátoru levou lineární gramatikou

Levá: $I \rightarrow Ip \mid Ic \mid p$

Pravá: $I \rightarrow pJ$

$J \rightarrow pJ \mid cJ \mid e$

c … číslo

p … písmeno

14. Zapište pravou lineární gramatiku čísla real v semilogaritmickém tvaru

$S \rightarrow +C \mid -C \mid C$

$C \rightarrow cC \mid c.D \mid c \mid ceE$

$D \rightarrow cD \mid ceE \mid c$

$E \rightarrow +E_1 \mid -E_1 \mid E_1$

$E_1 \rightarrow c \mid cE_1$

$N=\{S,C,D,E,E_1\}$

$T=\{c,e,+,-,.\}$

c … číslo

akceptuje záměrně i 000.000e000

samozřejmě lze upravit aby neakceptovala

15. Zapište co nejmenším počtem pravidel gramatiku popisující binární čísla se sudým počtem jedniček

Lichý počet jedniček:

$S_1 \rightarrow 0S_1 \mid 1S_2$

$S_2 \rightarrow 0S_2 \mid 1S_1 \mid e$

Sudý počet jedn

$S_1 \rightarrow 0S_1 \mid 1S_2 \mid$

$S_2 \rightarrow 1S_1 \mid 0S_2$

16. Uveďte obecný tvar překladových pravidel používaných v LEX

r_1 {akce 1}

r_2 {akce 2}

…

r_N {akce N}

r_i jsou regulární výrazy

{akce i} jsou programové fragmenty

17. Jaký řetězec rozpoznává LEX, je-li překladové pravidlo dáno výrazem \+?[0-9][0-9]*$

Všechny řetězce začínající + (ale nemusí), pokračují jednou či více číslicemi a končí koncem řádky.

VÝRAZ	POPIS	PŘÍKLAD
`const`	const	`const`
c	libovolný znak c, jež není operátorem	`a`
$\backslash c$	libovolný znak c	`\*`
"s"	řetězec s libovolných znaků	`"**"`
.	jakýkoliv znak kromě konce řádku	`a.*b`
^	začátek řádku	`^abc`
$	konec řádku	`abc$`
$[s]$	libovolný znak z množiny s	`[abc]`
$[\hat{}s]$	libovolný znak, který není v množině s	`[^abc]`
$r*$	0 nebo více r	`a*`
$r+$	1 nebo více r	`a+`
$r?$	0 nebo jeden r	`a?`
$r\{m,n\}$	m až n výskytů r	`a{1,5}`
r_1r_2	r_1 následovaný r_2	`ab`
$r_1\|r_2$	r_1 nebo r_2	`a|b`
(r)	r	`(a|b)`
r_1/r_2	r_1, pokud za ním následuje r_2	`abc/123`

Obr. 2.7: Regulární výrazy jazyka `lex`

18. Jak řeší lexikální analyzátory problém nalezení symbolu v případě, kdy je jeden symbol prefixem druhého?

Testuje další znaky na výskyt v jiném delším symbolu. Hledá nejdelší možný symbol.

19. Jaký řetězec rozpoznává LEX, je-li překladové pravidlo dáno výrazem *[1-9]*

Všechny řetězce začínající hvězdičkou a pokračující jednou nebo více číslicemi 1 až 9.

20. Popište formálně zásobníkový automat a význam jeho částí

Zásobníkový automat je sedmice R = (Q, T, G, δ, q_0, Z_0, F), kde

Q – konečná množina vnitřních stavů

T – konečná vstupní abeceda

G – konečná abeceda zásobníku

δ – přechodová funkce

q_0 – počáteční stav

Z_0 – počáteční symbol (dno) zásobníku

F – množina koncových stavů (je podmnožinou Q)

21. Popište přechodovou funkci zásobníkového automatu akceptujícího s prázdným zásobníkem, který je ekvivalentní gramatice G [S]: S → (S) | S ( ) | e

```
δ = {  (q, '(', ε)      → (q, '(')
       (q, ')', '(')    → (q, ε)
       (q, e, ε)        → (q, ε)     }
```

22. Jaký jazyk popisuje gramatika G [S]: S → (S) | S ( ) | e

Vnořené i nevnořené uzavřené kulaté závorky [počet otvíracích a zavíracích závorek bude stejný - např. (()(()))()] a prázdný řetězec.

23. Jaký jazyk popisuje gramatika G [S]: S →a S a | b S b | e

*Řetězec **a** a **b**, který je symetrický podle svého středu nebo prázdný řetězec.*

Jedná se o Palindrom – čte se stejně zepředu i zezadu.

24. Navrhněte gramatiku jazyka, jehož věty mají tvar w $w^{reverzní}$, kde w∈{0,1}*

```
S → 0S1 | 1S0 | e
```

25. Kdy označujeme větu jazyka jako víceznačnou

Věta generovaná gramatikou G je víceznačná, existují-li alespoň dva různé derivační stromy této věty. G pak rovněž nazýváme víceznačnou. (Věta = větná forma složená pouze z terminálních symbolů).

26. Popište princip způsobu zotavení ze syntaktické chyby v překladači PL/0

Vynechá část řetězce a část zásobníku. Přeskočí text až k následujícímu symbolu za procedurou. Slouží k tomu procedura TEST a stop symboly. Je to tzv. panické zotavení chyby.

27. Jaké vlastnosti musí splňovat jazyk analyzovatelný rekurzivním sestupem

Musí splňovat vlastnosti LL gramatik, nesmí obsahovat levou rekurzi. Na základě prvních několika symbolů se musí rozhodnout, jakou použije větev (pravidlo).

28. Vysvětlete funkci procedury Test(s1,s2: symset; n: integer); v překladači PL/0

Ověřuje, zda symbol patří do množiny následovníků S1. S2 – stop symboly, n – číslo chyby. Pokud symbol není následovníkem, pokračuje v načítání dalších symbolů až do místa se stop symbolem nebo možným následovníkem.

29. Zapište gramat. aritmetického výrazu s operátory + , *, a závorkami (,). Zapište levou derivaci věty i + i.

```
E → E     T - F - i      (položenej strom)
T → T E - +
F → (     E - T - F - i
```

30. Popište princip metody rekurzivního sestupu

Derivační strom je budován postupným voláváním procedur odpovídajících jednotlivým neterminálům gramatiky a jejich prováděním. Tělo procedury určuje pravé strany pravidel pro daný neterminální symbol. Pravé strany musí být rozlišitelné na základě symbolů vstupního řetězce aktuálních v okamžiku uplatnění příslušné pravé strany.

31. Charakterizujte syntetizované atributy

Atributy vyhodnocované průchodem derivačního stromu zdola nahoru nazýváme syntetizované.

32. Popište způsob vyhodnocování dědičných atributů

Atributy vyhodnocované průchodem derivačního stromu shora dolů nazýváme dědičné.

33. Popište zásady konstrukce postfixového výrazu z infixového

Vytvoříme derivační strom výrazu a poté jej vypíšeme metodou POSTORDER.

Nebo lze vytvořit pomocí zachování pořadí operandů, operátory následují za operandy.

POSTFIXOVÉ INSTRUKCE

`PLUS/MINUS/TIME/DIV` – vybere ze zásobníku dvě hodnoty, provede operaci a výsledek uloží do zásobníku

`TA adr`	– vloží do zásobníku adresu **adr**
`DR`	– dereferencuje vrchol zásobníku (nahradí adresu jejím obsahem)
`ST`	– uloží hodnotu z vrcholu zásobníku na adresu uloženou pod vrcholem zásobníku
`IFJ m`	– je-li vrchol zásobníku FALSE, provede skok na **m**-tou postfixovou instrukci
`NEG`	– unární mínus

Postfixová notace:

- (binární/unární) operátor je funkcí 2 nebo 1 operandů a zapisuje se za operandy.
- výstup funkce je vstupem další funkce
- operátory bezprostředně následují za operandy, pořadí operandů je zachováno

34. Zapište posloupnost postfixových instrukcí pro např: $a_{10} := - (x_{20} + y_{30})/(x_{20} - y_{30})$

```
(1) TA 10   (5) DR     (9)  DR      (13) DIV
(2) TA 20   (6) PLUS   (10) TA 30   (14) ST
(3) DR      (7) NEG    (11) DR      (15) STOP
(4) TA 30   (8) TA 20  (12) MINUS
```

NEG má přednost před DIV

35. Zapište výraz -2*(x+y)^3 pro případ 1) nejvyšší, 2) nejnižší precedence operátoru unárního minus

a) v prefixové, **b) v postfixové notaci**

```
1a) *, Un-, 2, ^, +, x, y, 3
2a) Un-, *, 2, ^, +, x, y, 3
1b) 2, Un-, x, y, +, 3, ^, *
2b) 2, x, y, +, 3, ^, *, Un-
```

nejvyšší precedence=operace provedena hned

nejnižší precedence=operace s nejnižší prioritou provedeny nakonec

36. Přeložte do posloupnosti postfix. instrukcí. if ($A_{10} < B_{20}$) then $C_{30} := (A_{10} + B_{20}) * (A_{10} - B_{20})$;

10,20,<,ifj, 30,10,20,+,10,20,-,*,=

```
(1) TA 10   (6) IFJ 20   (11) DR      (16) DR
(2) DR      (7) TA 30    (12) PLUS    (17) MINUS
(3) TA 20   (8) TA 10    (13) TA 10   (18) TIME
(4) DR      (9) DR       (14) DR      (19) ST
(5) <       (10) TA 20   (15) TA 20   (20) STOP
```

37. Přeložte do postfixových instrukcí příkaz: while x<y do x:= (x+y) / (x-y); je-li x na adrese 100 a y na adrese 101.

100,101,<,ifj,100,100,101,+,100,101,-,/,=,jmp

```
(1) TA 100   (6) IFJ 21    (11) DR       (16) DR
(2) DR       (7) TA 100    (12) PLUS     (17) MINUS
(3) TA 101   (8) TA 100    (13) TA 100   (18) DIV
(4) DR       (9) DR        (14) DR       (19) ST
(5) <        (10) TA 101   (15) TA 101   (20) JMP 1
                                         (21) STOP
```

38. Popište význam částí dynamické adresy (adresové dvojice)

*Adresová dvojice se skládá z **báze** (určuje hloubku zanoření aktuálního bloku ve vykonávaném programu) + **offsetu** (posun od začátku bloku/adres v daném bloku – aktivačním záznamu).*

39. **Formulujte podmínku, kterou musí splňovat program, aby statický řetězec výpočtového zásobníku stále splýval s dynamickým řetězcem.**
Podprogramy mohou volat pouze podprogramy, které jsou v dané úrovni deklarované (jsou jim podřazené), žádná rekurzivní volání. Statický a dynamický ukazatel jsou totožné, pokud je podprogram deklarován a volán ze stejné hladiny.

40. **Jaké informace o proměnných jsou uloženy v tabulce symbolů překladače jazyka pascalského typu**
    - *druh (návěští, konstanta, typ, proměnná, procedura, funkce)*
    - *hladina popisu (udává hladinu bloku, ve kterém byla popsána)*
    - *adresa*
    - *použití (Ano/Ne)*
    - *typ – údaj o standardním jednoduchém typu*
        - *údaj o typu definovaném uživatelem*
        - *údaj o strukturovaném typu*
    - *formální parametr*
    - *druhy formálních parametrů*
    - *způsob volání a hodnota*

41. **Jaká je časová složitost práce s rozptýleně organizovanou tabulkou symbolů v závislosti na počtu symbolů v programu?**
$O(n^2/k)$, kde n je počet symbolů v programu a k je počet seznamů v tabulce

42. **Jaká je závislost časové režie vyhledávání v netříděné uspořádané tabulce symbolů na počtu jmen v tabulce?**
Kvadratická. $O(n^2)$, kde n je počet jmen v tabulce.
O(m*n) n počet symbolů, m počet výskytů, čím bude program obsáhlejší tím bude obsahovat více výskytů.
Takže m*n => n*n => n^2.

43. **Popište způsob vytváření a práce s frekvenčně uspořádanou tabulkou symbolů**
Když se najde položka, posune se k začátku, často používané jsou více nahoře a dříve se na ně narazí, tabulka se prohledává od začátku.

44. **K čemu slouží mapovací funkce pole a z jakých se skládá částí**
Slouží k určení umístění prvku vzhledem k počátku pole. Skládá se z adresy začátku pole + hodnota mapovací fce, také záleží na velikosti jednotlivých prvků pole (jejich typu – char, int, double, ...).
    - *konstantní část – adresa začátku pole*
    - *deskriptor pole*
    - *indexy prvků*
    - *jednotlivé rozměry pole (n-rozměrnost)*
    - *velikost jednoho prvku (char, integer, double, float, ...)*

45. **Jakými vlastnostmi jazyka je podmíněno statické přidělování paměti**
Jazyk nesmí povolovat rekurzivní volání podprogramů, proměnné musí být globální a pouze statické.
46. **Popište odlišnost zpřístupnění nelokálních proměnných Pascalu a C**
V C nelze definovat vnořené procedury, všechny proměnné jsou buď lokální, nebo globální.
V Pascalu lze definovat libovolně vnořené procedury, v nejnižší proceduře jsou přístupné proměnné všech nadřazených procedur.

47. **Uveďte, jaké údaje ukládá překladač v aktivačním záznamu**
Lokální proměnné, parametry, návratová adresa, funkční hodnotu (je-li podprogram funkcí), statický a dynamický ukazatel, pomocné proměnné pro mezivýsledky, další potřebné informace k uspořádání aktivačních záznamů.
48. **Vysvětlete, jakým mechanismem překladač zajišťuje respektování lokality identifikátorů v blokově strukturovaném jazyce.**
Při vytváření bloku uloží případné symboly (proměnné) do tabulky a při ukončování bloku je vyjme.
Tabulka symbolů je uspořádána do podoby zásobníku (pro jazyky s blokovou strukturou). Rozsahová jednotka je blok, modul, funkce, balík, …

49. **Uveďte datové struktury , které jsou použitelné k přidělování paměti pro a) rekurzivně volané procedury a funkce, b) dynamické proměnné, c) dynamické typy, d) paralelně proveditelné programové jednotky**
***a)** zásobník (stack), **b)** halda (heap), **c)** halda, zásobník , **d)** obecný zásobník*

50. **Popište způsob a důvod použití displeje.**
Je to vektor ukazující na aktivační záznamy, pro zrychlení přístupu k nelokálním objektům při velké hloubce zanoření podprogramů.
51. **Jaká omezení budou důsledkem přístupu do výpočtového zásobníku pomocí displeje, který je realizován jako tříprvkový vektor adres**
Hloubka zanoření volání podprogramů je omezena na počet prvků displeje – max. hloubka zanoření je 3.

52. **Jaké informace jsou předávány při volání podprogramu, je-li formálním parametrem procedura a) v případě statického přidělování paměti, b) v případě dynamického přidělování paměti**
a) předá se jako parametr adresa začátku procedury
b) ještě k tomu přidáme ukazatel na statické prostředí ve kterém je definována procedura, hladina+offset

53. Uveďte příklad víceznačné gramatiky. Víceznačnost dokažte.

E → E+E | E*E | i

*Pro větu i * i + i existují 2 d. stromy. Gramatika je víceznačná, má-li věta jazyka více derivačních stromů.*

54. Může pro víceznačnou gramatiku existovat ekvivalentní gramatika jednoznačná?

Ano může, ale nemusí. Jednoznačnost gramatik je algoritmicky nerozhodnutelná (důvodů může být nekonečně mnoho). Existují gramatiky, jejichž nejednoznačnost nelze odstranit (inherentně nejednoznačné).

55. Operátor umocnění je ve Fortranu pravoasociativní. Zapište pravidla pro aritmetický výraz respektující tuto vlastnost.

U → U ^ c, kde c je číslice a ^ je operátor umocnění,(umocnění → umocnění^c)

56. Charakterizujte vztah mezi jazyky s LL(0) gramatikou a regulárními jazyky

LL(0) je podmnožinou regulárních jazyků. Regulární jazyky vytváření derivační strom shora dolů.

57. Popište tvar LR(0) položky a význam jejích jednotlivých částí

Obsahuje vrchol zásobníku, jednotlivá přepisovací pravidla obsahující načtený řetězec. Tečka určuje aktuální pozici zpracování řetězce.

vrchol zásobníku **venku**

 # : S → . B
B → . a B b
B → . A
A → . b A
A → . c

Tečka symbolizuje rozhranní *zásobník . dosud nezpracovaná část vstupu*

58. Uveďte formální definici LL(1) gramatiky

Bezkontextová gramatika G = (N, T, P, S) se nazývá LL(1) gramatika, když pro každé A z N platí podmínka: Jestliže A → α, A → β jsou různá pravidla v P, pak:
FIRST(α FOLLOW(A)) ∩ FIRST(β FOLLOW(A)) = množina prázdná.

59. Zdůvodněte, proč je každá LL(1) gramatika silná

Pokud pro danou gramatiku G vystačíme při rozhodování o výběru pravidla pro expanzi s informací o dopředu prohlíženém řetězci délky nejvýše k, pak se gramatika G nazývá silná LL(k) gramatika.
$FIRST_1(\alpha\ FOLLOW_1(A)) \cap FIRST_1(\beta\ FOLLOW_1(A)) =$ množina prázdná.
=> při analýze není potřeba používat ahead (náhled) vystačíme si pouze s právě načtených symbolem (jedním).

60. Uveďte nutnou a postačující podmínku pro to, aby gramatika byla silná LL(k)

Gramatika LL(k) je silná, jestliže pro všechna A z N platí: jsou-li $A \rightarrow \alpha$, $A \rightarrow \beta$ různá pravidla v P (P=pole pravidel), pak: $FIRST_K(\alpha \cdot FOLLOW_K(A)) \cap FIRST_K(\beta \cdot FOLLOW_K(A)) = \emptyset$

61. Uveďte příklady algoritmicky nerozhodnutelných problémů z teorie formálních jazyků

Nejednoznačnost bezkontextových gramatik. Nelze rozhodnout, zda dojde k zacyklení programu na základě zdrojového kódu. Zda daná gramatika je LR(k) pro libovolné k. Zda pro daný jazyk existuje LR(k) gramatika.

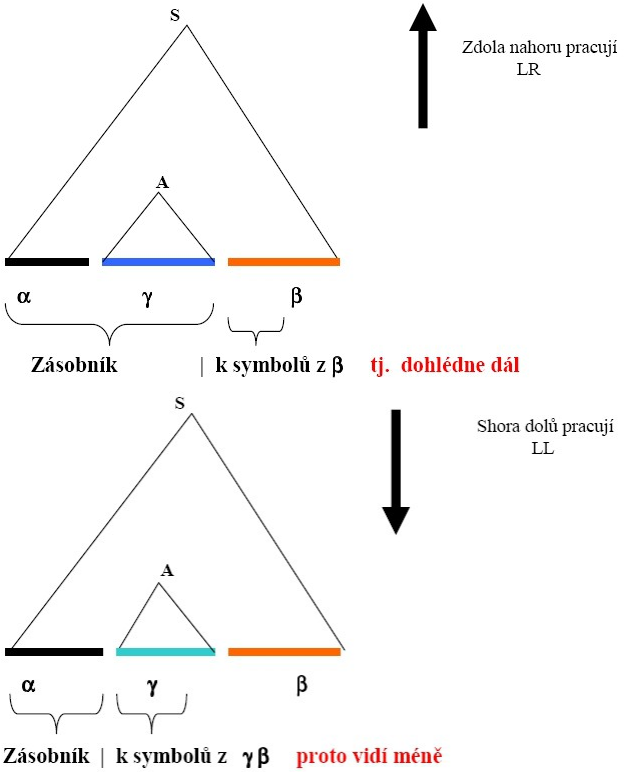
62. Existuje pro libovolnou gramatiku typu 2 algoritmus pro převod na a) ekvivalentní nelevorekurzivní gramatiku, b) LL(k) gram., c) LR(k) gramatiku, d) pro výpočet množin LR(0) položek?

a) Ano – Známe algoritmus pro odstranění levé rekurze, b) Ne, c) Ano, d) Ne

63. Zdůvodněte proč LR(k) gramatiky popisují obsáhlejší třídu jazyků než LL(k)

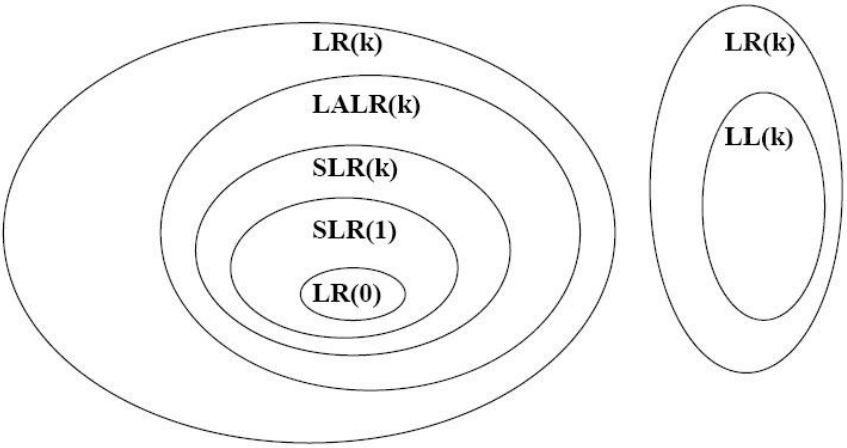
Protože jsou obecnější – sledují najednou větší část řetězce než LL(k), takže umožňují např. rozpoznat řetězec, který začíná libovolným počtem stejných znaků.

Proč je LR širší třídou než LL?
Je dáno tím, kolik má informace (jak daleko vidí do vstupního textu).



64. Porovnejte mohutnosti množin LR(0), LALR(k), LR(k) položek

Platí:



65. Jaké podmínky musí splňovat množiny LR(0) položek, aby gramatika byla SLR(1)?

X → a . - provedu, pokud následující symbol je prvkem ***follow(x)***

X → a . B - provedu, pokud následující symbol patří do ***first(B)***

Pokud je tam kolize v jedné množině, tečka na konci a v jiném pravidle tečka někde jinde (uprostřed, na začátku) pak není SLR(1).

66. K čemu slouží úprava gramatiky zvaná "pohlcení terminálu"? Uveďte příklad.

K odstranění first-follow či first-first kolize. Př.: Převeďte G(N, T, P, A) na LL(1):

```
A → BaC
B → e | abC
C → c | cBC
```

Pro odstranění použijeme pohlcení follow symbolu:

```
1)                  2)
A     → [Ba]C       A       → [Ba]C
[Ba]  → a | abCa    [Ba]    → a[Ba]'
B     → e | abC     [Ba]'   → e | bCa
C     → c | cBC     B       → e | abC
                    C       → c
```

Vytvořili jsme nový neterminální symbol `[Ba]`, který vznikl sloučením nepohodlného follow symbolu a pravidel pro B. Tím ovšem vznikla first-first kolize, kterou je třeba odstranit.

67. Jakou metodu syntaktické analýzy používá YACC

Zdola nahoru.

68. Jakým způsobem řeší YACC konflikty redukce-redukce

LALR algoritmus s konflikty řeší YACC takto: ***konflikt redukce-redukce řeší výběrem pravidla dle pořadí jejich uvedení****. Konflikt redukce-přesun řeší upřednostněním přesunu.*

69. Co je obsahem deskriptoru třídy u OO jazyků

Ukazatel na deskriptor rodiče, Seznam datových položek – ofset, viditelnost (private, public), Seznam metod – jejich vstupní bod, údaj o static, viditelnost.

U dynamických jazyků obsahuje descriptor třídy ještě ukazatel na tabulku virtuálních metod (VMT).

70. Popište mechanismus zpracování statických metod

a) c . f() překlad volání f závisí na typu objektové proměnné c
b) překladač vyhledá třídu C objektu c
c) v C hledá metodu f, když jí nenajde, hledá jí v rodiči C…
d) pokud program není chybný, v nějakém předkovi jí najde
e) volání se přeloží do skoku na její vstupní bod.

Souhrnně: U statických metod se hledá vstupní bod v deskriptoru třídy, při neúspěchu se hledá v deskriptoru předka...

71. Popište mechanismus zpracování dynamických (virtuálních) metod

Metody mohou být překryty ve třídě potomka.
V době překladu nelze určit zda se jedná o metodu předka či potomka.
Řešení:
Deskriptor třídy musí obsahovat vektor s metodami pro každé ze jmen nestatických metod (VMT).
Když třída P dědí z R, pak VMT pro P začíná se vstupními body všech metod platných pro R a pokračuje s novými metodami zavedenými v P. Pokud třída P překryje metodu R_f, bude na místě R_f ve VMT pro P adresa P_f.
Při exekuci p.f () musí přeložený kód provést:

1. *Zjistit deskriptor třídy objektu na který ukazuje pp,*
2. *Z něj zjistit adresu vstupního bodu metody f*
3. *Skok do podprogramu na adresu tohoto vstupního bodu*

72. Kdy překladač vytváří CIR (Class Instance Record) a jaké informace v něm ukládá?

CIR ukládá instanční proměnné a odkaz na seznam dynamicky vázaných metod.
Vytváří se:

- *instanciací – CIR vytvořen výpočtem na haldě*
- *deklarací – CIR vytvořen výpočtem, jeho zásobníková adresa a velikost určeny při překladu*

73. Co je obsahem VMT (tabulka virtuálních metod) a kdy se VMT vytváří?

Tabulka obsahuje vstupní body virtuálních metod třídy. Vytváří se při vytváření instance (volání konstruktoruú třídy. Viz 71

74. Popište odlišnost obsahu aktivačního záznamu v případě jazyka s nemožností/možností vnořování podprogramů

Jazyk bez možnosti vnořování podprogramů nepotřebuje mít uložen statický ukazatel.

75. Popište obecně základní cyklus interpretu

```
do {
      RI ← PROGRAM [ PC ];
      PC ← PC + 1;
      switch (RI.INSTRUCTION_CODE) {
            case INSTRUCTION_CODE_1:
            STATEMENTS_OF_INSTRUCTION_CODE_1;
            case INSTRUCTION_CODE_2:
            STATEMENTS_OF_INSTRUCTION_CODE_2;
            …
            case INSTRUCTION_CODE_n:
            STATEMENTS_OF_INSTRUCTION_CODE_n;
      }
} while (RI.INSTRUCTION_CODE ≠ STOP);
```

first(X) obsahuje vsechny terminalni symboly, kteryma X muze zacinat
follow(X) obsahuje vsechny terminalni symboly, ktery se vyskytnou za X

S pomocí algoritmu generování z přednášek rozepište zadaný příklad pro generování z čtveřic.
S pomocí algoritmu generování z přednášek rozepište zadaný příklad pro generování z trojic.
Příklady převodu na LL gramatiku, konstrukci rozkladové tabulky a rozklad zadané věty.
Příklady konstrukce LR(0) a SLR(1) tabulek a rozklad zadané věty.

69) Zadán vyraz `A:=(x+y)*(z-w)`

a) zapsat ho pomoci trojic - Víceadresové instrukce (čtveřice / trojice)
b) upravit tabulku z přednášky pro symbol :=
c) generovat instrukce podle tabulky z přednášky

69 a)
```
1.(+,x,y)
2.(-,z,w)
3.(*,(1),(2))
4.(:=,A,(3))
```

69 b) Překlad přiřazovacího příkazu do čtveřic

Gramatika: **S → a := E** ;
E → T { + T }
T → F { * F)
F → (E) | a

čtveřice tvaru:
```
( +, adr1, adr2, adr3 )
( *, adr1, adr2, adr3 )
(:=, adr1, adr2, 0 )
```

69 c) C := A * B; if A < B then C := A +B ;
30 10 20 10 20 30 10 20

se přeloží na:
```
(1) '* ' , 10, 20, 100
(2) ':=' , 100, 30, 0
(3) '< ' , 10, 20, 101
(4) 'JIF', 101, 7, 0
(5) '+ ' , 10, 20, 102
(6) ':=' , 102, 30, 0
```

70) Dana gramatika G[S]

```
S -> AS|e
A -> aA|b
```

Dokazte, ze G generuje regulární jazyk a tento jazyk zapište regulárním výrazem.

(a*b)*

71) Dana překladová atributovaná gramatika G[S] (viz. přednášky):
Pravidla:

```
1. S -> i {TA} := E {ST} {TA}.adr := i.adr
2. E -> TE\'
3. E\'-> +T {PLUS} E\'
4. E\'-> e
5. T -> FT\'
6. T -> *F {MULT} T\'
7. T\'-> e
8. F -> (E)
9. F -> i {TA} {DR}          TA.adr := i.adr
```

Za pomoci rozkladové tabulky znázorněte stavy zásobníku při překladu vety
$\mathbf{i_{11} := i_{12} + i_{11}}$ (ty čísla 11 a 12 u těch i mají být indexy = adresy proměnné)

72) Napište gramatiku s lichým počtem jedniček

```
S→1A
A→1S | e
```