2.1 Alapfogalmak

1. Mirol mire fordít a fordítóprogram?
Fordítóprogramok valamely programozási nyelven írt programot másik nyelv szabályaira fordítanak le.

2. Mirol mire fordít az assembler?
assembler: assembly nyelvről gépi kódra fordít

3. Mi a különbség a fordítóprogram és az interpreter között?
Fordítás: A forráskódot a fordítóprogram tárgyprogrammá alakítja. A tárgyprogramokból a szerkesztés során futtatható állomány jön létre.
Fordítási idő: amikor a fordító dolgozik
Futási idő: amikor a program fut
Interpretálás: A forráskódot az interpreter értelmezi és azonnal végrehajtja.
a fordítási és futási idő nem különül el

4. Mi a virtuális gép?
Virtuális gép: olyan gép szoftveres megvalósítása, amelynek a bájtkód a gépi kódja.

5. Mi a különbség a fordítási és a futási ido között?
Forrásprogram->Forrás-kezelő (source handler)->Lexikális elemző (scanner)->Szintaktikus elemző (parser)->Szemantikus elemző (semantic analizer)->Belső reprezentáció

6. Mi a feladata az analízisnek (a fordítási folyamat elso fele) és milyen részfeladatokra bontható?
Forrásprogram->Forrás-kezelő (source handler)->Lexikális elemző (scanner)->Szintaktikus elemző (parser)->Szemantikus elemző (semantic analizer)->Belső reprezentáció

7. Mi a feladata a szintézisnek (a fordítási folyamat második fele) és milyen részfeladatokra bontható?
Belső reprezentáció->Kódgenerátor (code generator)->Optimalizáló (optimizer)->Kód-kezelő (code handler)->Tárgyprogram

8. A fordítóprogram mely részei adhatnak hibajelzést?
Lexikális elemző, Szintaktikus elemző, Szemantikus elemző

9. Mi a lexikális elemzo feladata, bemenete, kimenete?
bement: karaktersorozat kimenet: szimbólumsorozat, lexikális hibák feladata: lexikális egységek (szimbólumok) felismerése: azonosító, konstans, kulcsszó...

10. Mi a szintaktikus elemzo feladata, bemenete, kimenete?
bemenet: szimbólumsorozat kimenet: szintaxisfa, szintaktikus hibák feladata: a program szerkezetének felismerése a szerkezet ellenőrzése: megfelel-e a nyelv definíciójának?

11. Mi a szemantikus elemzo feladata, bemenete, kimenete?
bemenet: szintaktikusan elemzett program (szintaxisfa, szimbólumtábla, ...) kimenet: szemantikusan elemzett program, szemantikus hibák feladata: típusellenőrzés változók láthatóságának ellenőrzése eljárások hívása megfelel-e a szignatúrának? stb.

12. Mi a kódgenerátor feladata, bemenete, kimenete?
bemenet: szemantikusan elemzett program kimenet: tárgykód utasításai (pl. assembly, gépi kód) feladata: a forrásprogrammal ekvivalens tárgyprogram készítése

13. Mi a kódoptimalizáló feladata, bemenete, kimenete?
bemenet: tárgykód kimenet: optimalizált tárgykód feladata: valamilyen szempontok szerint jobb kód készítése pl. futási sebesség növelése, méret csökkentése pl. felesleges utasítások megszüntetése, ciklusok

kifejtése...

14. Mi a fordítás menetszáma?
A fordítás annyi menetes, ahányszor a programszöveget (vagy annak belső reprezentációit) végigolvassa a fordító a teljes fordítási folyamat során.

2.2. Lexikális elemzo
1. Milyen nyelvtanokkal dolgozik a lexikális elemzo?
lexikális elemzés: megadható reguláris nyelvtannal (3-as)

2. Hogy épülnek fel a reguláris kifejezések?
kifejezőereje azonos a reguláris nyelvtanokéval alapelemek: üres halmaz üres szöveget tartalmazó halmaz egy karaktert tartalmazó halmaz konstrukciós műveletek: konkatenáció unió: | lezárás: * további „kényelmi” műveletek: +, ?

3. Hogy épülnek fel a véges determinisztikus automaták?
elemei: ábécé állapotok halmaza átmenetfüggvény kezdőállapot végállapotok halmaza

4. Milyen elv szerint ismeri fel a lexikális elemzo a lexikális elemeket?
A lexikális elemző mindig a lehető leghosszabb karaktersorozatot ismeri fel.

5. Mi a szerepe a lexikális elemek sorrendjének?
A while input megfelel a változónév definíciójának és egy kulcsszó is egyben. Melyiket kellene felismerni?
A lexikális elemző megadásakor sorbarendezhetjük a szimbólumok definícióit. Ha egyszerre több szimbólum is felismerhető, a sorrendben korábbi lesz az

6. Mi a különbség a kulcsszavak és a standard szavak között?
Kulcsszó A kulcsszavaknak előre adott jelentésük van, és ez nem definiálható felül.
Standard szó A standard szavaknak előre adott jelentésük van, de ez felüldefiniálható.

7. Mi az elofeldolgozó fázis feladata?
feljegyzi a makródefiníciókat, elvégzi a makróhelyettesítéseket, meghívja a lexikális elemzőt a beillesztett fájlokra, kiértékeli a feltételeket és dönt a kódrészletek beillesztéséről vagy törléséről.

8. Mutass példát olyan hibára, amelyet a lexikális elemzo fel tud ismerni és olyanra is, amelyet nem!
kulcsszó elgépelés észrevehető: then helyet than
Szimbólum kihagyás már nem: a*1 helyett a1 már nem veszi észre mert az a1 lehet azonosító

2.3. Szinataktikus elemzés alapfogalmai
1. Mikor ciklusmentes egy nyelvtan?
ciklusmentesség: nincs A => +A levezetés ellenpélda: A -> B B -> A

2. Mikor redukált egy nyelvtan?
redukáltság: „nincsenek felesleges nemterminálisok” minden nemterminális szimbólum előfordul valamelyik mondatformában mindegyikből levezethető valamely terminális sorozat
ellenpélda: A -> aA

3. Mikor egyértelmu egy nyelvtan?
egyértelműség: minden mondathoz pontosan egy szintaxisfa tartozik

4. Mi a különbség a legbal és legjobb levezetés között?

Legbal: mindig a legbaloldalibb nemterminálist helyettesítjük: S => AB => aaB => aab
Legjobb: mindig a legjobboldalibb nemterminálist helyettesítjük S => AB => Ab v aab

5. Mi a különbség a felülrol lefelé és az alulról felfelé elemzés között?
Felülről lefelé: A startszimbólumból indulva, felülről lefelé építjük a szintaxisfát. A mondatforma baloldalán megjelenő terminálisokat illesztjük az elemzendő szövegre.
Alulról felfelé: Az elemzendő szöveg összetartozó részeit helyettesítjük nemterminális szimbólumokkal (redukció) és így alulról, a startszimbólum felé építjük a fát.

6. Mi az összefüggés az elemzési irányok és a legbal, illetve legjobb levezetésközött?
Felülről lefelé -> legbal Alulról felfelé -> legjobb

7. Milyen alapveto stratégiák használatosak a felülrol lefelé elemzésekben?
visszalépéses keresés (backtrack): ha nem illeszkednek a szövegre a mondatforma baloldalán megjelenő terminálisok, lépjünk vissza, és válasszunk másik szabályt
előreolvasás: olvassunk előre a szövegben valahány szimbólumot, és az alapján döntsünk az alkalmazandó szabályról =>LL elemzések

8. Milyen alapveto stratégiák használatosak az alulról felfelé elemzésekben?
visszalépéses keresés (backtrack): ha nem sikerül eljutni a startszimbólumig, lépjünk vissza, és válasszunk másik redukciót
precedenciák használata: az egyes szimbólumok között adjunk meg precedenciarelációkat és ennek segítségével határozzuk meg a megfelelő redukciót
előreolvasás: olvassunk előre a szövegben valahány szimbólumot, és az alapján döntünk a redukcióról =>LR elemzések

2.4. LL elemzések
1. Definiáld a FIRSTk(α) halmazt, és röviden magyarázd meg a definíciót!
FIRSTk (α): az α mondatformából levezethető terminális sorozatokk hosszúságú kezdőszeletei (ha a sorozat hossza kisebb mint k, akkor az egész sorozat eleme FIRSTk (α)-nak, akár epszilon ∈ FIRSTk (α) is előfordulhat)

Definíció: FIRSTk (α) FIRSTk (α) = {x | α ⇒* xβ ∧ |x| = k}∪ {x | α ⇒* x ∧ |x| < k}

2. Definiáld az LL(k) grammatikákat és röviden magyarázd meg a definíciót!
LL(k) grammatika: a levezetés tetszőleges pontján a szöveg következő k terminálisa meghatározza az alkalmazandó levezetési szabályt
Definíció: LL(k) grammatika Tetszőleges S ⇒* wAβ ⇒ wα1β ⇒* wx S ⇒* wAβ ⇒ wα2β ⇒* wy levezetéspárra FIRSTk (x) = FIRSTk (y) esetén α1 = α2.

3. Definiáld a FOLLOWk(α) halmazt és röviden magyarázd meg a definíciót!
FOLLOWk (α): a levezetésekben az α után előforduló k hosszúságú terminális sorozatok (ha a sorozat hossza kisebb mint k, akkor az egész sorozat eleme FOLLOWk (α)-nak, ha α után vége lehet a szövegnek, akkor # ∈ FOLLOWk (α))
Definíció: FOLLOWk (α) FOLLOWk (α) = {x | S ⇒* βαγ ∧ x ∈ FIRSTk (γ) \ {epszilon}} ∪ {# | S ⇒* βα}

4. Definiáld az egyszeru LL(1) grammatikát!
Definíció: Egyszerű LL(1) Olyan LL(1) grammatika, amelyben a szabályok jobboldala terminális szimbólummal kezdődik (ezért epszilon-mentes is). (Az összes szabály A -> aα alakú.)

5. Mi az egyszeru LL(1) grammatikáknak az a tulajdonsága, amire az elemzo épül?
Tétel Egy grammatika pontosan akkor egyszerű LL(1), ha csak A → aα alakú szabályai vanak és ha A → a1α1 és A → a2α2 két különböző szabály, akkor a1 6= a2.

6. Mit csinál az egyszeru LL(1) elemzo, ha a verem tetején az A
ha a verem tetején nemterminális szimbólum (A) van: ha van A -> aα szabály: A helyére aα és bejegyzés a

szintaxisfába különben: hiba

7. Definiáld az epszilonmentes LL(1) grammatikát!
Definíció: epszilon-mentes LL(1): Olyan LL(1) grammatika, amely epszilon-mentes. (Nincs A -> epszilon szabály.)

8. Mi az epszilonmentes LL(1) grammatikáknak az a tulajdonsága, amire az elemzo épül?
Tétel Egy grammatika pontosan akkor LL(1), ha epszilon-mentes és ha A -> α1 és A -> α2 két különböző szabály, akkor FIRST1(α1) ∩ FIRST1(α2) = üreshalmaz.

9. Mit csinál az epszilonmentes LL(1) elemzo, ha a verem tetején az A nemterminális van és a bemenet következo szimbóluma az a terminális?
ha a verem tetején nemterminális szimbólum (A) van (és a szövegben a következik): ha van A → α szabály, amelyre a ∈ FIRST1(α): A helyére α és bejegyzés a szintaxisfába különben: hiba

10. Definiáld az LL(1) grammatikát!
Tetszőleges S ⇒*wAβ ⇒ wα1β ⇒* wx S ⇒* wAβ ⇒ wα2β ⇒* wy levezetéspárra FIRST1(x) = FIRST1(y) esetén α1 = α2.

11. Mi az LL(1) grammatikáknak az a tulajdonsága, amire az elemzo épül?
Tétel Egy grammatika pontosan akkor LL(1) grammatika, ha bármely A → α1 és A → α2 különböző szabályok esetén FIRST1(α1FOLLOW1(A)) ∩ FIRST1(α1FOLLOW1(A)) = üreshalmaz.
FIRST1(αFOLLOW1(A)) jelentése: α-hoz egyenként konkatenáljuk FOLLOW1(A) elemeit és az így kapott halmaz minden elemére alkalmazzuk a FIRST1 függvényt.

12. Mit csinál az LL(1) elemzo, ha a verem tetején az A nemterminális van és a bemenet következo szimbóluma az a terminális?
ha a verem tetején nemterminális szimbólum (A) van (és a szövegben a következik): ha van A → α szabály, amelyre a ∈ FIRST1(αFOLLOW1(A)): A helyére α és bejegyzés a szintaxisfába különben: hiba

13. Milyen komponensei vannak az LL(1) elemzoknek?

14. Hogyan épülnek fel a rekurzív leszállásos elemzoben a nemterminális szimbólumokhoz rendelt eljárások?
a rekurzív leszállás eljárásai elején megvizsgálhatjuk, hogy megfelelő-e az aktualis_szimbolum, és szükség esetén addig ugorjuk át a szimbólumokat a bemeneten, amíg megfelelő nem lesz

2.5. LR elemzések
1. Mit jelentenek a léptetés és redukálás muveletek?
Léptetés A bemenet következő szimbólumát a verem tetejére helyezzük.
Redukálás A verem tetején lévő szabály-jobboldalt helyettesítjük a megfelelő nemterminális szimbólummal.

2. Mi a kiegészített grammatika és miért van rá szükség?
Az elemzés végét arról fogjuk felismerni, hogy egy redukció eredménye a kezdőszimbólum lett. Ez csak akkor lehet, ha a kezdőszimbólum nem fordul elő a szabályok jobboldalán.

3. Mi a nyél szerepe az alulról felfelé elemzésekben?
Nyél: a mondatformában a legbaloldalibb egyszerű részmondat. Épp a nyelet kell megtalálni a redukcióhoz. Probléma: „Mi a nyél?” léptetni vagy redukálni kell? ha több lehetőség is van, melyik szabály szerint kell redukálni?

4. Mondd ki az LR(k) grammatika definícióját és magyarázd meg!
LR(k) grammatika: k szimbólum előreolvasásával eldönthető,hogy mi legyen az elemzés következő lépése.

Egy kiegészített grammatika LR(k) grammatika (k >= 0), ha
S =>* α Aw) α β w
S =>*
α β y = gamma delta x és FIRSTk (w) = FIRSTk (y) esetén
α =

5. Hogyan határozza meg az LR(0) elemzo véges determinisztikus automatája, hogy léptetni vagy redukálni kell?
az átmeneteit a verembe kerülő szimbólumok határozzák meg léptetéskor terminális redukáláskor nemterminális amikor elfogadó állapotba jut, akkor kell redukálni

6. Hogy néz ki egy LR(0)-elem és mi a jelentése?
Például: [S -> a.Ad] jelentse azt, hogy az a szimbólumot már elemeztük, az Ad rész még hátra van.
Definíció: LR(0) elem Ha A -> α a grammatika egy helyettesítési szabálya, akkor az α = α1 α2 tetszőleges felbontás esetén [A -> α1.α2] a grammatika egy LR(0) eleme.

7. Milyen muveletek segítségével állítjuk elo a kanonikus halmazokat és mi ezeknek a szerepe?
Definíció: LR(0) kanonikus halmazok 1) closure([S' -> .S]) a grammatika egy kanonikus halmaza. 2) Ha I a grammatika egy kanonikus elemhalmaza, X egy terminális vagy nemterminális szimbóluma, és read(I, X) nem üres, akkor read(I, X) is a grammatika egy kanonikus halmaza. 3) Az első két szabállyal az összes kanonikus halmaz előáll.

8. Mi köze van az LR(0) kanonikus halmazoknak az LR(0) elemzo véges determinisztikus automatájához?
Az automata felépítése: A closure([S' -> .S]) legyen a kezdőállapot. Ha I' = read(I, X), akkor az automatában legyen (I) ->(nyilon X) (I′) átmenet. A végállapotok azok a kanonikus halmazok, amelyekben olyan elemek vannak, ahol a pont a szabály végén van.

9. Hogyan határozzuk meg az LR(0) elemzo automatájában az átmeneteket?

10. Hogyan határozzuk meg az LR(0) elemzo automatájában a végállapotokat?

11. Mondd ki az LR(0)-elemzés nagy tételét!
Tétel: az LR(0) elemzés nagy tétele Egy

12. Milyen konfliktusok lehetnek az LR(0) elemzo táblázatban?
Ha egy Ik kanonikus halmaz alapján nem lehet egyértelműen eldönteni, hogy az adott állapotban milyen akciót kell végrehajtani.
léptetés/redukálás konfliktus: az egyik elem léptetést, egy másik redukálást ír elő
redukálás/redukálás konfliktus: az egyik elem az egyik szabály szerinti, a másik egy másik szabály szerinti redukciót ír elő

13. Milyen esetben ír elo redukciót az SLR(1) elemzés?
léptessünk, ha az automata tud lépni az előreolvasott szimbólummal
redukáljunk, ha az előreolvasott szimbólum benne van a szabályhoz tartozó nemterminális FOLLOW1 halmazában

14. Hogy néz ki egy LR(1)-elem és mi a jelentése?
Definíció: LR(1) elem
Ha A -> α a grammatika egy helyettesítési szabálya, akkor az α = α1α2 tetszőleges felbontás és a terminális szimbólum (vagy a = #) esetén [A → α1.α2, a] a grammatika egy LR(1)-eleme. A -> α1.α2 az LR(1) elem magja, a pedig az előreolvasási szimbóluma.
[V -> a. , =] jelentése: a V -> a szabály építését befejeztük és a szabályt az = szimbólum követheti.

15. Milyen esetben ír elo redukciót az LR(1) elemzés?
LR(1) elemzés: Az {[S -> (.S)S, #], [S -> .,)], [S → .(S)S,)]} állapotban: léptetés: (hatására , redukálás:) hatására

16. Miért van általában lényegesen több állapota az LR(1) elemzoknek, mint az LR(0) (illetve SLR(1)) elemzoknek?
Az előreolvasási szimbólumok miatt nő az állapotok száma!

17. Mikor nevezünk két LR(1) kanonikus halmazt összevonhatónak?
Egyes kanonikus halmaz párok csak az előreolvasási szimbólumokban különböznek. Ezeket a halmazokat egyesíthetőnek nevezzük.

18. Hogyan kapjuk meg az LALR(1) kanonikus halmazokat?
Ha az LR(1)-es kanonikus halmazokat kiszámoljuk és összevonjuk, akkor az összevont halmazokat LALR(1) kanonikus halmazoknak nevezzük.

19. Milyen fajta konfliktus keletkezhet a halmazok összevonása miatt az LALR(1) elemzo készítése során?
Redukálás-redukálás konfliktus: előfordulhat!
Léptetés-léptetés konfliktus nem fordulhat elő!
Léptetés-redukálás konfliktus sem fordulhat elő!

20. Mi az LALR(1) elemzo felépítésének két alapveto módja?
Az LALR(1) elemzőnek ugyanannyi állapota van, mint az SLR(1) (és LR(0)) emelzőknek. Mivel megjelennek benne az előreolvasási szimbólumok, több nyelvtan lesz elemezhető vele, mint SLR(1) módszerrel.
egyszerűbb módszer: az LR(0) kanonikus halmazokból indulunk (de azoknak is csak a lényeges elemeit fogjuk tárolni) az előreolvasási szimbólumokat utólag határozzuk meg

21. Milyen lépésekbol áll az LR elemzok hibaelfedo tevékenysége?
1) Hiba detektálása esetén meghívja a megfelelő hibarutint. 2) A verem tetejéről addig töröl, amíg olyan állapotba nem kerül, ahol lehet az error szimbólummal lépni. 3) A verembe lépteti az error szimbólumot. 4) Az bemeneten addig ugorja át a soron következő terminálisokat, amíg a hibaalternatíva építését folytatni nem tudja.

2.6. Az if - then - else probléma
1. Mi az if - then - else probléma?

```
if(b1)    if(b1)
  if(b2)    if(b2)
    x++;      x++;
  else    else
    y++;    y++;
```

2. Hogyan kell értelmezni a gyakorlatban az egymásba ágyazott elágazásokat, ha az az if - then - else probléma miatt nem egyértelmu?
„Az else ág az őt közvetlenül megelőző if utasításhoz tartozik.”

3. Hogyan oldják meg az if - then - else problémát az LR elemzok?
Az LALR(1) elemzésnél léptetés-redukálás konfliktushoz vezet az if utasítás előbb látott nyelvtana. Feloldása: léptetni kell! Így az else az őt közvetlenül megelőző if utasításhoz fog tartozni.

4. Mire kell figyelni programozási nyelvek tervezésekor, ha el akarjuk kerülni az if - then - else problémát?
az if utasítás végét jelző kulcsszó bevezetése.

2.7. Szimbólumtábla

1. Milyen információkat tárolunk a szimbólumtáblában a szimbólumokról (fajtájuktól függetlenül)?
szimbólum neve és szimbólum attribútumai: definíció adatai, típus, tárgyprogram-beli cím, definíció helye a forrásprogramban, szimbólumra hivatkozások a forrásprogramban

2. Milyen információkat tárolunk a szimbólumtáblában a változókról?
változó: típus, módosító kulcsszavak: const, static ..., címe a tárgyprogramban (függ a változó tárolási módjától)

3. Milyen információkat tárolunk a szimbólumtáblában a függvényekrol?
függvény, eljárás, operátor: paraméterek típusa, visszatérési típus, módosítók, címe a tárgyprogramban

4. Milyen információkat tárolunk a szimbólumtáblában a típusokról?
típus (típusleíró, típusdeszkriptor): egyszerű típusok: méret, rekord: mezők nevei és típusleírói, tömb: elem típusleírója, index típusleírója, méret, intervallum-típus: elem típusleírója, minimum, maximum, unio-típus: a lehetséges típusok leírói, méret

5. Milyen információkat tárolunk a szimbólumtáblában az osztályokról?
A rekordokhoz hasonló: típusleírót kell készíteni hozzá.
Láthatóság szabályozása: a mezőkhöz és tagfüggvényekhez fel kell jegyezni, hogy milyen a láthatóságuk (public, protected, private).
Az osztályok névteret is alkotnak: (statikus) adattagjai minősített névvel is elérhetők, ilyenkor az előbb látott technikát lehet használni.

6. Mi a szimbólumtábla két alapveto muvelete és mikor használja ezeket a fordítóprogram?
keresés: szimbólum használatakor
beszúrás: új szimbólum megjelenésekor tartalmaz egy keresést is: „Volt-e már deklarálva?”

7. Mi a változó hatóköre?
hatókör: „Ahol a deklaráció érvényben van.”

8. Mi a változó láthatósága?
láthatóság: „Ahol hivatkozni lehet rá a nevével.” része a hatókörnek az elfedés miatt lehet kisebb, mint a hatókör

9. Mi a változó élettartama?
élettartam: „Amíg memóriaterület van hozzárendelve.”

10. Hogyan kezeljük változó hatókörét és láthatóságát szimbólumtáblával?
A szimbólumokat egy verembe tesszük.
Keresés: a verem tetejéről indul az első találatnál megáll
Blokk végén a hozzá tartozó szimbólumokat töröljük.

11. Milyen szerkezeteku szimbólumtáblákat ismersz?
Verem szimbólumtábla: Számon kell tartani a blokk kezdetét a szimbólumtábla vermében!
Faszerkezetű szimbólumtábla: Minden blokkhoz egy fa tartozik. A szimbólumtábla sorai a fa csúcsaiban helyezkednek el.

12. Miben tér el a névterek és blokkok kezelése a szimbólumtáblában?
A névterek szimbólumait a veremből nem törölni kell, hanem feljegyezni egy másik tárterületre.

2.8. Szemantikus elemzés

1. Miért nem a szintaktikus elemzo végzi el a szemantikus elemzés feladatait?
Mert a szemantikus elemzés erősen függ a konkrét programozási nyelvtől!

2. Mi a különbség a statikus és a dinamikus típusozás között?
Statikus: a kifejezésekhez fordítási időben a szemantikus elemzés rendel típust az ellenőrzések fordítási időben történnek futás közben csak az értékeket kell tárolni futás közben „nem történhet baj” előny: biztonságosabb pl.: Ada, C++, Haskell ...
Dinamikus: a típusellenőrzés futási időben történik futás közben az értékek mellett típusinformációt is kell tárolni minden utasítás végrehajtása előtt ellenőrizni kell a típusokat típushiba esetén futási idejű hiba keletkezik előny: hajlékonyabb pl.: Lisp, Erlang ...

3. Mi a különbség a típusellenorzés és a típuslevezetés között?
Típusellenőrzés: minden típus a deklarációkban adott a kifejezések egyszerű szabályok alapján típusozhatók egyszerűbb fordítóprogram, gyorsabb fordítás kényelmetlenebb a programozónak
Típuslevezetés, típuskikövetkeztetés: a változók, függvények típusait (általában) nem kell megadni a típusokat fordítóprogram „találja ki” a definíciójuk, használatuk alapján bonyolultabb fordítóprogram, lassabb fordítás kényelmesebb a programozónak

4. Mi a fordítóprogram teendoje típuskonverzió esetén?
Típuskonverzió esetén: a típusellenőrzés során át kell írni a kifejezés típusát ha szükséges, akkor a tárgykódba generálni kell a konverziót elvégző utasításokat az int és a double típusok reprezentációja különbözik!

5. Mik az akciószimbólumok?
A grammatika szabályaiban jelöljük, hogy milyen szemantikus elemzési tevékenységekre van szükség. Ezeket hívjuk akciószimbólumoknak. A nyelvtanban @(ró-szerű vacak) jellel kezdődnek. Az akciószimbólumok által jelölt szemantikus tevékenységeket szemantikus rutinoknak nevezzük.

6. Mik az attribútumok?
A szimbólumokhoz attribútumokat rendelünk. Ezek jelzik, hogy a szimbólumhoz milyen szemantikus értékek (attribútumértékek) kapcsolódnak.

7. Hogyan kapnak értéket az attribútumok?
Jelölés: A.x, y Az A szimbólumhoz az x és y attribútumokat rendeljük.

8. Mi a szintetizált attribútum?
Szintetizált attribútum: A helyettesítési szabály bal oldalán áll abban a szabályban, amelyikhez az őt kiszámoló szemantikus rutin tartozik.
például: szabály: Kifejezés0.t ! Kifejezés1.t + Kifejezés2.t szemantikus rutin: Kifejezés0.t := int Az információt a szintaxisfában alulról felfelé közvetíti.

9. Mi a kitüntetett szintetizált attribútum?
Kitüntetett szintetizált attribútum: Olyan attribútumok, amelyek terminális szimbólumokhoz tartoznak és kiszámításukhoz nem használunk fel más attribútumokat.
például: szabály: Kifejezés.t ! konstans.t Az információt általában a lexikális elemző szolgáltatja.

10. Mi az örökölt attribútum?
Örökölt attribútum: A helyettesítési szabály jobb oldalán áll abban a szabályban, amelyikhez az őt kiszámoló szemantikus rutin tartozik.
például: szabály: Változólista.t ! változó.t Folytatás.t szemantikus rutin: változó.t := Változólista.t Az információt a szintaxisfában felülről lefelé közvetíti.

11. Mivel egészítjük ki a nyelvtan szabályait attribútum fordítási grammatikák esetében?
egy fordítási grammatika szimbólumait attribútumokkal, a szabályokat feltételekkel, valamint rendeljünk szemantikus rutinokat az akciószimbólumokhoz

12. Mi a direkt attribútumfüggoség?

Definíció: Direkt attribútumfüggőségek Ha az Y .b attribútumot kiszámoló szemantikus rutin használja az X.a attribútumot, akkor (X.a, Y .b) egy direkt attribútumfüggőség. Ezek a függőségek a függőségi gráfban ábrázolhatók.

13. Mi az S - ATG? Milyen elemzésekhez illeszkedik?
S-ATG és az alulról felfelé elemzés
redukció esetén a szabály jobb oldalának attribútumértékei már ismertek (hiszen nincsenek örökölt attribútumok)
a szabályhoz rendelt szemantikus rutin feladata a baloldalon álló szimbólum szintetizált attribútumainak meghatározása

14. Mi az L - ATG? Milyen elemzésekhez illeszkedik?
L-ATG és a rekurzív leszállás
a A ! X1 . . . Xn szabályhoz tartozó eljárás formális paraméterei legyenek az A örökölt attribútumai visszatérési értéke az A szintetizált attribútumai az eljárások végrehajtása épp az L-ATG attribútumkiértékelési sorrendjét adja

2.9. Assembly
1. Mi az assembly?
programozási nyelvek egy csoportja
gépközeli: az adott processzor utasításai használhatóak általában nincsenek programkonstrukciók, típusok, osztályok stb.
a futtatható programban pontosan azok az utasítások lesznek, amit a programba írunk lehet optimalizálni lehet olyan trükköket használni, amit a magasabb szintű nyelvek nem engednek meg

2. Mi az assembler?
az assembly programok fordítóprogramja az assembler

3. Milyen fobb regisztereket ismersz (általános célú, veremkezeléshez, adminisztratívcélra)?
eax: „accumulator” - elsősorban aritmetikai számításokhoz
ebx: „base” - ebben szokás tömbök, rekordok kezdőcíméttárolni
ecx: „counter” - számlálókat szokás tárolni benne (pl. forjellegű ciklusokhoz)
edx: „data” - egyéb adatok tárolása; aritmetikai számításokhoz segédregiszter
esi: „source index” - sztringmásolásnál a forrás címe
edi: „destination index” - sztringmásolásnál a cél címe
veremkezeléshez:
esp: „stack pointer” - a program vermének a tetejét tartja nyilván
ebp: „base pointer” - az aktuális alprogramhoz tartozó verem-részt tartja nyilván
olvasni és írni:
eip: „instruction pointer” - a következő végrehajtandó utasítás címe
eflags: jelzőbitek - pl. összehasonlítások eredményeinek tárolása

4. Mi köze van egymáshoz az eax, ax, al, ah regisztereknek?
eax 32 bitből áll; két része van: a „felső” 16 bitnek nincs külön neve az „alsó” 16 bit: ax; ennek részei: a „felső” bájt: ah az „alsó” bájt: al

5. Milyen aritmetikai utasításokat ismersz assemblyben?
inc eax az eax értékét megnöveli eggyel (lehet memóriahivatkozás is az operandus)
dec word [C] a C címtől kezdődő 2 bájt értékét csökkenti eggyel (lehet regiszter is az operandus)
add eax,dword [D] eax-hez adja a D címtől kezdődő 4 bájt értékét
sub edx,ecx levonja edx-ből ecx-et
Az add és a sub utasításokra ugyanaz a szabály, mint a mov-ra:
mul ebx Megszorozza eax értékét ebx értékével. Az eredmény: edx-be kerülnek a nagy helyiértékű bájtok eax-be az alacsony helyiértékűek

div ebx Elosztja a 2564·edx+eax értéket ebx értékével. Az eredmény: eax-be kerül a hányados edx-be a maradék

6. Mutasd be a logikai értékek egy lehetséges ábrázolását és a muveleteik megvalósítását assemblyben!
A magasszintű programozási nyelvek logikai (bool) típusának egy
lehetséges megvalósítása: 1 bájton tároljuk, hamis érték: 0, igaz érték: 1, logikai és: and utasítás, logikai vagy: or utasítás, tagadás: not és and 1,

7. Milyen feltételes ugró utasításokat ismersz?
je: „equal” - ugorj, ha egyenlő
jne: „not equal” - ugorj, ha nem egyenlő
jb: „below” - ugorj, ha kisebb ≡ jnae: „not above or equal” - nem nagyobb egyenlő
ja: „above” - ugorj, ha nagyobb ≡ jnbe: „not below or equal” - nem kisebb egyenlő
jnb: „not below” - nem kisebb ≡ jae: „above or equal” - nagyobb egyenlő
jna: „not above” - nem nagyobb ≡ jbe: „below or equal” - kisebb egyenlő

8. Hogyan kapják meg a feltételes ugró utasítások a cmp utasítás eredményét?
a cmp utasítás a zero flag -et állítja be és az ugró utasítások ezt a flag-ot nézik.

9. Milyen veremkezelo utasításokat ismersz assemblyben, és hogyan muködnek ezek?
push eax: eax értéke bekerül a verembe (4 bájt)
pop ebx: a verem tetején lévő (4 bájt) értéke bemásolódik ebx-be, a veremmutató visszaáll

10. Melyik utasításokkal lehet alprogramot hívni és alprogramból visszatérni assemblyben?
push eax ; Betesszük a verembe a paramétert.
call kiir ; Meghívjuk az eljárást. ; Az eljárás a veremből hanálhatja ; az átadott paramétert.
add esp,4 ; Visszaállítjuk a veremmutatót ; a push előtti állapotba.

2.10. Kódgenerálás
1. Hogyan generálunk kódot egyszeru típusok értékadásához?
alakja: assignment-> variable assignment_operator expression
Értékadást megvalósító kód 1) a kifejezést az eax regiszterbe kiértékelő kód 2) mov [Változó],eax

2. Hogyan generálunk kódot egy ágú elágazáshoz?
alakja: statement -> if condition then program end
Egy ágú elágazás kódja: 1) a feltételt az al regiszterbe kiértékelő kód 2) cmp al,1 3) jne near Vége 4) a then-ág programjának kódja 5) Vége:

3. Hogyan generálunk kódot több ágú elágazáshoz?
statement ->
if condition1 then program1
elseif condition2 then program2
. . .
elseif conditionn then programn
else programn+1 end

Több ágú elágazás kódja: 1) az 1. feltétel kiértékelése az al regiszterbe 2) cmp al,1 3) jne near Feltétel_2 4) az 1. ág programjának kódja 5) jmp Vége 6) . . . 7) Feltétel_n: az n-edik feltétel kiértékelése az al regiszterbe 8) cmp al,1 9) jne near Else 10) az n-edik ág programjának kódja 11) jmp Vége 12) Else: az else ág programjának kódja 13) Vége:

4. Hogyan generálunk kódot elöltesztelo ciklushoz?
alakja: statement -> while condition program end
Elöl tesztelő ciklus kódja: 1) Eleje: a ciklusfeltétel kiértékelése az al regiszterbe 2) cmp al,1 3) jne near Vége

4) a ciklusmag programjának kódja 5) jmp Eleje 6) Vége:

5. Hogyan generálunk kódot hátultesztelo ciklushoz?
alakja: statement ! loop program while condition
Hátul tesztelő ciklus kódja: 1) Eleje: a ciklusmag programjának kódja 2) a ciklusfeltétel kiértékelése az al regiszterbe 3) cmp al,1 4) je near Eleje

6. Hogyan generálunk kódot for ciklushoz?
alakja: statement ! for variable from value1 to value2program end
For ciklus kódja: 1) a „from" érték kiszámítása a [Változó] memóriahelyre 2) Eleje: a „to" érték kiszámítása az eax regiszterbe 3) cmp [Változó],eax 4) ja near Vége 5) a ciklusmag kódja 6) inc [Változó] 7) jmp Eleje 8) Vége:

7. Hogyan generáljuk kezdoérték nélküli statikus változó definíciójának assembly kódját?
kezdőérték nélkül: int x;
Kezdőérték nélküli változódefiníció fordítása: section .bss ; a korábban definiált változók... Lab12: resd 1 ; 1 x 4 bájtnyi terület

8. Hogyan generáljuk kezdoértékkel rendelkezo statikus változó definíciójának assembly kódját?
kezdőértékkel: int x=5;
Kezdőértékkel adott változódefiníció fordítása: section .data ; a korábban definiált változók... Lab12: dd 5 ; 4 bájton tárolva az 5-ös érték

9. Hogyan generáljuk aritmetikai kifejezés kiértékelésének assembly kódját? (konstans, változó, beépített függvény)
Konstans kiértékelése mov eax,25
Változó kiértékelése: mov eax,[X] ; ahol X a változó címkéje

10. Mutasd meg a különbséget a mindkét részkifejezést kiértékelo és a rövidzáras logikai operátorok assembly kódja között!
kifejezés1 és kifejezés2 kiértékelése
; a 2. kifejezés kiértékelése az al regiszterbe
push ax ; nem lehet 1 bájtot a verembe tenni!
; az 1. kifejezés kiértékelése az al regiszterbe
pop bx ; bx-nek a bl részében van,
; ami nekünk fontos
and al,bl

kifejezés1 és kifejezés2 kiértékelése (rövidre zár)
; az 1. kifejezés kiértékelése az al regiszterbe
cmp al,0
je Vége
push ax
; a 2. kifejezés kiértékelése az al regiszterbe
mov bl,al
pop ax
and al,bl
Vége:

11. Hogyan generáljuk a goto utasítás assembly kódját?
Lab: inc [X]
...
jmp Lab

12. Miért nehéz a break utasítás kódgenerálását megoldani S-ATG használata esetén?

Alulról felfelé elemzéskor... a ciklusmag kódját kell először generálni (a break kódját is) a ciklus kódját később
Probléma: a Vége címkét a ciklus feldolgozásakor generáljuk, pedig szükség van rá a break kódjában is! Azaz ez a címke egy örökölt attribútum...

13. Mit csinál a call és a ret utasítás?
call Címke az eip regiszter tartalmát a verembe teszi ez a call utáni utasítás címe visszatérési címnek nevezzük átadja a vezérlést a Címke címkéhez mint egy ugró utasítás
ret kiveszi a verem legfelső négy bájtját és az eip regiszterbe teszi mint egy pop utasítás az program a veremben talált címnél folytatódik

14. Hogyan adjuk át assemblyben az alprogramok paramétereit és hol lesz a lefutás után a visszatérési érték (C stílus esetén)?
a paramétereket a verembe kell tenni a call utasítás előtt C stílusú paraméterátadás: fordított sorrendben tesszük a verembe az utolsó kerül legalulra az első a verem tetejére az ejárásból való visszatérés után a hívó állítja vissza a vermet visszatérési érték: az eax regiszterbe kerül

15. Hogyan épül fel az aktivációs rekord?
lokális változók, előző akt. rek. bázispointere, visszatérési cím, 1. paraméter, 2. paraméter ...

16. Mi a bázismutató és melyik regisztert szoktuk erre a célra felhasználni?
bázis pointer az ebp-ből indulva egy láncolt listára vannak felfűzve az aktivációs rekordok ez teszi lehetővé az eggyel korábbi aktivációs rekordhoz való visszatérést az alprogram végén

17. Hol tároljuk alprogramok lokális változóit?
a lokális változók a verem tetejére kerülnek

18. Mi a különbség az érték és a hivatkozás szerinti paraméterátadás assembly kódja között?
érték szerint: a paraméterértékeket másoljuk a verembe ha az alprogram módosítja, az nem hat az átadott változóra
hivatkozás szerint: az átadandó változóra mutató pointert kell a verembe tenni az alprogramban a lokális változó kiértékelése is módosul: mov eax,[ebp+p] mov eax,[eax]

19. Milyen csoportokba oszthatók a változók tárolásuk szerint és a memória mely részeiben tároljuk az egyes csoportokba tartozó változókat?
statikus memóriakezelés: a .data vagy .bss szakaszban globális vagy statikusnak deklarált változók előre ismerni kell a változók méretét, darabszámát
dinamikus memóriakezelés: blokk-szerkezethez kötődő, lokális változók: verem tetszőleges élettartamú változók: heap memória

2.11. Kódoptimalizálás

1. Mi a különbség a lokális és a globális optimalizálás között?
Lokális optimalizálás: egy alapblokkon belüli átalakítások
Globális optimalizálás: a teljes program szerkezetét meg kell vizsgálni

2. Mit jelent a gépfüggo optimalizálás?
gépfüggő optimalizálás: az adott architektúra sajátosságait használja ki

3. Mit nevezünk alapblokknak és melyik optimalizálás során van szerepe?
Definíció: alapblokk
Egy programban egymást követő utasítások sorozatát alapblokknak nevezzük, ha az első utasítás kivételével egyik utasítására sem lehet távolról átadni a vezérlést (assembly programokban: ahová a jmp, call, ret utasítások „ugranak”; magas szintű nyelvekben: eljárások, ciklusok eleje, elágazások ágainak első utasítása, goto utasítások célpontjai) az utolsó utasítás kivételével nincs benne vezérlés-átadó utasítás (assembly

programban: jmp, call, ret magas szintű nyelvekben: elágazás vége, ciklus vége, eljárás vége, goto) az utasítás-sorozat nem bővíthető a fenti két szabály megsértése nélkül

4. Mutass példát konstansok összevonására!
Eredeti kód: a := 1 + b + 3 + 4;
Optimalizált kód: a := 8 + b;

5. Mutass példát konstans továbbterjesztésére!
Eredeti kód
```
a := 6;
b := a / 2;
c := b + 5;
```
Optimalizált kód
```
a := 6;
b := 3;
c := 8;
```

6. Mutass példát azonos kifejezések többszöri kiszámításának elkerülésére!
Eredeti kód
```
x := 20 - (a * b);
y := (a * b) ^ 2;
```
Optimalizált kód
```
t := a * b;
x := 20 - t;
y := t ^ 2;
```

7. Mi az ablakoptimalizálási technika lényege?
Ez egy módszer a lokális optimalizálás egyes fajtáihoz.
Ablak: egyszerre csak egy néhány utasításnyi részt vizsgálunk a kódból a vizsgált részt előre megadott mintákkal hasonlítjuk össze ha illeszkedik, akkor a mintához megadott szabály szerint átalakítjuk ezt az „ablakot” végigcsúsztatjuk a programon
Az átalakítások megadása: {minta->helyettesítés} szabályhalmazzal a mintában lehet paramétereket is használni

8. Mi a különbség a kódkiemelés és a kódsullyesztés között?
Kódkiemelés
Eredeti kód
```
if( x < 10 ){
  a = 0; //
  b++;
}else{
  b--;
  a = 0; //
}
```
Optimalizált kód
```
a = 0; //
if( x < 10 ){
  b++;
}else{
  b--;
}
```

Kódsüllyesztés
Eredeti kód
```
if( x < 10 ){
```

```
  x = 0;//
  b++;
}else{
  b--;
  x = 0;//
}
```
Optimalizált kód
```
if( x < 10 ){
  b++;
}else{
  b--;
}
x = 0;//
```

9. Hogyan változtatja a program sebességét és méretét a ciklusok kifejtése?

10. Mi a frekvenciaredukálás lényege?
Frekvenciaredukálás: költséges utasítások „átköltöztetése” ritkábban végrehajtódó alapblokkba
példa: ciklusinvariánsnak nevezzük azokat a kifejezéseket, amelyeknek a ciklus minden lefutásakor azonos az értékük a ciklusinvariánsok (esetenként) kiemelhetők a ciklusból

11. Mutass példát eros redukcióra!
a ciklusban lévő költséges művelet (legtöbbször szorzás) kiváltása kevésbé költségessel
Eredeti kód
```
for( int i=a; i<b; i+=c ){
  cout < < 3*i;
}
```
Optimalizált kód
```
int t1 = 3*a;
int t2 = 3*c;
for( int i=a; i<b; i+=c ){
  cout < < t1;
  t1 += t2;
}
```