Bakalářské zkoušky (příklady otázek z informatiky)

léto 2022

1 Zásobníkový automat (3 body)

1. Napište definici zásobníkového automatu.
2. Uvažujme abecedu $\Sigma = \{+, *, (,)\}$ a jazyky generované bezkontextovými gramatikami

   $G_A = (\{A, E\}, \Sigma, \{A \to E + E, E \to EE \mid (E) \mid \lambda\}, A)$,

   $G_M = (\{M, E\}, \Sigma, \{M \to E * E, E \to EE \mid (E) \mid \lambda\}, M)$.

   Sestrojte zásobníkový automat, který přijímá koncovým stavem $L(G_A) \cup L(G_M)$, tedy množinu slov generovaných alespoň jednou z gramatik G_A, G_M.

2 Binární vyhledávací stromy (3 body)

1. Definujte binární vyhledávací strom.
2. Implementujte operaci vložení prvku do stromu (bez vyvažování). Rozeberte časovou složitost v nejlepším a nejhorším případě.
3. Implementujte operaci nalezení následníka: pro daný klíč x najděte vrchol stromu s nejmenším klíčem větším než x. Rozeberte časovou složitost v nejlepším a nejhorším případě.

Implementaci popište nejlépe pseudokódem.

3 Databáze (3 body)

1. Vysvětlete pojem *funkční závislosti atributů relace* v relačním datovém modelu. Bude relace $R(\underline{K_1}, A, B, C)$, s klíčem (K_1), kde $F = \{K_1 \to AC,\ A \to B\}$ vhodně navržená? Pokud ano, vysvětlete proč. Pokud ne, vysvětlete proč, a navrhněte, jak by bylo vhodné návrh změnit.
2. Uvažujte následující relační databázové schéma:

   Company(CompanyID, CompanyName, Country)
   Product(ProductID, CompanyID, ProductName, ProductType)
   Product.CompanyID $\subseteq$ *Company*.CompanyID

   Podtržení označuje klíče relací.

   - Pomocí SQL dotazu najděte společnosti, které nevyrábějí žádný produkt typu 'Automobil'.
   - Jak by se změnil sémantický význam modelu (pokud vůbec), pokud by se klíč relace *Product* změnil z jednoduchého klíče ProductID na složený klíč ProductID,CompanyID?
3. Je zaručeno, že rozvrhy
   `S_1=(R_1(X),W_2(X),W_1(Y),W_2(Y),COMMIT_1,COMMIT_2)` a
   `S_2=(R_1(X),W_1(Y),COMMIT_1,W_2(X),W_2(Y),COMMIT_2)` povedou ke stejné změně databáze?

   Vysvětlete své rozhodnutí.

4 Emulátor procesoru (3 body)

Vaším úkolem je navrhnout vnitřní rozhraní pro softwarový emulátor jednoduchého fiktivního procesoru. Procesor má 16 32-bitových registrů, 32-bitový ukazatel instrukcí a je k němu připojena paměť představující souvislý blok 8-bitových bajtů s počáteční adresou 0 a o velikosti nejvýše 2GB. V této paměti je uložen kód i data, počáteční naplnění paměti a registrů neřešte.

Instrukce jsou kódovány tak, že první bajt (*operační kód*) určuje, o jakou instrukci se jedná, a za ním následující bajty určují parametry instrukce, přičemž počet bajtů a jejich význam je jednoznačně určen operačním kódem. Některé operační kódy mohou zůstat nedefinované - pokus o provedení takové instrukce končí zastavením emulátoru s patřičným chybovým hlášením. Podobně mají být ošetřeny i další možné chyby, jako pokus o provedení instrukce ležící (byť jen částečně) mimo paměť a pokus o čtení či zápis dat na pozici (částečně) mimo paměť.

Emulátor musí být snadno rozšiřitelný o nové instrukce (připojením nových zdrojových souborů implementujících nové instrukce a minimálním množstvím zásahů do existujícího kódu). Rozhraní společné části kódu by přitom mělo maximálně usnadňovat implementaci jednotlivých instrukcí.

Navržené rozhraní by mělo být použitelné i pro emulaci multiprocesorového systému, v němž jsou (jednovláknovým simulátorem, tedy bez nutnosti synchronizace) emulovány paralelně běžící procesory s oddělenými registry a sdílenou pamětí.

1. Definujte kompletní rozhraní (v podobě tříd a hlaviček jejich veřejných funkcí) mezi společnou částí emulátoru a implementací jednotlivé instrukce. To zahrnuje
   - Mechanismus pro registraci implementace instrukce pod daným operačním kódem v emulátoru
   - Hlavičku funkce provádějící danou instrukci a dalších případných funkcí z rozhraní instrukce volaných emulátorem
   - Hlavičky funkcí společné části emulátoru volaných z kódu implementujícího jednotlivé instrukce

   Popište, jakým způsobem budou vyřešeny chybové situace (popsané výše).
2. Napište kompletní implementaci těchto instrukcí:
   - `STORE32 ra,rb,c32`, která zapíše 32-bitový obsah registru `ra` jako 4 bajty do paměti na adresu danou součtem obsahu registru `rb` a konstanty `c32` (a do 3 následujících bajtů). Číslo registru `ra` je uloženo v dolních 4 bitech prvního bajtu za operačním kódem, číslo registru `rb` v jeho horních 4 bitech a konstanta `c32` je v dalších 4 bajtech, instrukce má tedy celkem 6 bajtů.
   - `JEQ ra,rb,c32`, která porovná obsah registrů `ra` a `rb` a pokud jsou shodné, provede skok na adresu určenou konstantou `c32`, v opačném případě běh programu pokračuje následující instrukcí. Parametry instrukce jsou kódovány stejně jako u `STORE32`. Vysvětlete, jak budou ve společné části emulátoru řešeny skokové instrukce.

   Uveďte také kód, kterým se tyto instrukce zařadí do emulátoru, a popište, odkud se tento kód volá.
3. Emulovaný procesor má běžet i v režimu, kdy jsou 32-bitová čísla ukládána v opačném formátu (*little/big-endian*) než používá prostředí, ve kterém emulátor běží. Jak tento problém vyřešíte?

K řešení úlohy si vyberte libovolný jazyk z množiny {Java, C#, C++}.

5 Překlad konstrukcí vyššího jazyka (3 body)

Mějme jednoduchý procesor s architekturou inspirovanou MIPS. Procesor má 32 32-bitových celočíselných obecných registrů označených **$R0-$R31** a jeden 32-bitový registr **PC**, který ukazuje na začátek následující instrukce. Hodnota registru **$R0** je vždy **0**.

Instrukce procesoru jsou uvedeny v následující tabulce. V popisu používáme tyto operandy:
– *label* - symbolické návěští do kódu
– *[mem]* - adresa paměti
– *reg* - jeden obecný registr

instrukce	popis	operace
LD reg=[mem]	načte celočíselnou proměnnou z adresy *mem* do registru *reg*	reg = *mem
ST [mem]=reg	uloží registr *reg* do celočíselné proměnné na adrese *mem* do	*mem = reg
ADD reg1=reg2,reg3	sečte registr *reg2* s registrem *reg3* a výsledek uloží do registru *reg1*	reg1 = reg2 + reg3
J label	nepodmíněný skok na adresu instrukce označenou návěštím *label*	PC = label
BEQ reg1,reg2,label	podmíněný skok, který se provede, pokud jsou si registry *reg1* a *reg2* rovny	if(reg1==reg2) PC = label
BNE reg1,reg2,label	podmíněný skok, který se provede, pokud si registry *reg1* a *reg2* nejsou rovny	if(reg1!=reg2) PC = label
SLT reg1=reg2,reg3	nastaví registr *reg1* na hodnotu 1, pokud je registr *reg2* menší než registr *reg3*, jinak se nastaví na 0	reg1 = (reg2 < reg3)

Mějme následující posloupnost instrukcí:

```
    LD    $r1=[q]
    LD    $r2=[n]
    LD    $r3=[inc]
    J     lab1
lab2:
    LD    $r4=[a]
    LD    $r5=[b]
    ADD   $r4=$r4,$r1
    SLT   $r5=$r5,$r4
    BEQ   $r5,$r0,lab3
    ST    [b]=$r4
lab3:
    ADD   $r1=$r1,$r3
lab1:
    SLT   $r4=$r1,$r2
    BNE   $r4,$r0,lab2
```

1. Vyberte (zakroužkujte) z následujících úryvků v jazyce C všechny, které odpovídají výše uvedené posloupnosti instrukcí (všechny neznámé identifikátory považujte za 32-bitovou celočíselnou proměnnou, která je někde deklarovaná). Pro zvolené řešení připište k instrukcím čísla řádků odpovídajících příkazů v jazyce C.

```
1    for (i=q; i<n; i+=inc)
2      if (a+i <= b)
3        b = a+i;
```

```
1    if (a+i > b)
2      for (i=q; i<n; i+=inc)
3        b = a+i;
```

```
1    for (i=q; i<n; i+=inc)
2      if (a+i > b)
3        b = a+i;
```

```
1    if (a+i < b)
2      for (i=q; i<n; i+=inc)
3        b = a+i;
```

```
1    i = q;
2    while (i<n) {
3      if (a+i > b)
4        b = a+i;
5      i += inc;
6    }
```

Žádný z uvedených úseků neodpovídá. Napište vlastní kód a očíslujte řádky kódu.

2. Napište (stačí jeden řádek), jak se změní původní kód v jazyce C, pokud instrukci `BEQ` nahradíme instrukcí `BNE` se stejnými operandy.

3. Předpokládejme, že je podmínka v příkazu `if` rozšířena o část `&& i < b`, tj. podmínka bude mít tvar `if (`*původní podmínka* `&& i < b)`. Upravte původní sekvenci instrukcí tak, aby realizovala tuto rozšířenou podmínku.

6 Sítě (3 body)

Uvažme soustavu 8 síťových uzlů označených `A`–`H`. Zde je výpis IPv4 routovacích tabulek všech zařízení:

Metriky nejsou uvedeny, předpokládejte, že jsou všude stejné. Stejně tak můžeme přepokládat, že každé síťové rozhraní má právě jednu IPv4 adresu. Loopback rozhraní jsme pro jednoduchost vynechali.

Node `A`

Destination	Gateway	Interface
default	10.0.1.3	10.0.1.5

Node `B`

Destination	Gateway	Interface
default	10.0.3.1	10.0.3.16

Node `C`

Destination	Gateway	Interface
default	10.0.4.1	10.0.4.2

Node `D`

Destination	Gateway	Interface
10.0.4.0/24	on-link	10.0.4.1
default	10.0.3.1	10.0.3.42

Node `E`

Destination	Gateway	Interface
default	10.0.1.4	10.0.1.9

Node `F`

Destination	Gateway	Interface
10.0.3.0/24	on-link	10.0.3.1
10.0.4.0/24	10.0.3.42	10.0.3.1
default	10.0.2.1	10.0.2.2

Node `G`

Destination	Gateway	Interface
10.0.1.0/24	10.0.1.2	10.0.1.1
10.0.0.0/16	10.0.2.2	10.0.2.1
default	192.168.0.1	192.168.0.42

Node `H`

Destination	Gateway	Interface
10.0.1.9/32	on-link	10.0.1.4
10.0.1.0/24	on-link	10.0.1.3
default	10.0.1.1	10.0.1.2

1. Načrtněte grafové schéma topologie sítě a vyznačte na něm IP adresy na jednotlivých rozhraních (rozhraní nejsou v tabulkách pojmenována, ale IP adresy nám je jednoznačně identifikují).

2. Dále uvažme IPv4 datagram odesílaný z uzlu `A`:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver = 4|IHL = 5|Type of Service|       Total Length = 1044     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Identification = 0xc001   |Flags|    Fragment Offset = 0  |     Flags:
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+     - do not fragment = 0
|TimeToLive = 3 |Proto = 17(UDP)|         Header Checksum       |     - more fragments = 0
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Source Address = 10.0.1.5                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Address = 10.0.4.2                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Neuvedené hodnoty považujte za správně vyplněné, ale v našem příkladu nebudou hrát žádnou roli. Stručně popište průchod datagramu soustavou sítí, zejména pak:

- přes které uzly datagram postupně projde,
- co s daným datagramem každý uzel provede (s krátkým zdůvodněním na základě hodnot z hlavičky datagramu a routovacích tabulek),
- jaké položky z hlavičky datagramu, případně routovacích tabulek, budou při zpracování datagramu změněny.

7 Aproximační algoritmy (otázka studijního zaměření – 3 body)

Uvažte následující problém: Vstupem je neorientovaný graf $G = (V, E)$, k dvojic vrcholů $s_1, t_1, ..., s_k, t_k$ a celočíselný parametr $L \in \mathbb{N}$. Výstupem je podmnožina indexů $I \subset \{1, ..., k\}$ a množina cest $\{P_i \mid i \in I\}$ taková, že pro každé $i \in I$, cesta P_i vede mezi s_i a t_i a má délku (v počtu hran) nejvýšše L, a cesty P_i jsou hranově disjunktní. Cílem je maximalizovat velikost I, tedy počet nalezených cest.

Navrhněte co nejlepší aproximační algoritmus pro tento problém. Dokažte co nejlepší odhad (horní i dolní) na aproximační faktor Vašeho algoritmu.

8 Max-cut (otázka studijního zaměření – 3 body)

Nechť $G = (V, E)$ je neorientovaný graf s m hranami.

1. Navrhněte pravděpodobnostní algoritmus, který najde řez grafu G o velikosti (ve střední hodnotě) alespoň $m/2$.
2. Dokažte, že G obsahuje bipartitní podgraf (ne nutně indukovaný) s alespoň $m/2$ hranami.

9 Arrangement přímek (otázka studijního zaměření – 3 body)

1. Definujte arrangement přímek v rovině.
2. Popište (jakýkoliv) způsob reprezentace arrangementu přímek v rovině v počítači.
3. Navrhněte (jakýkoliv) polynomiální algoritmus, který zkonstruuje arrangement přímek v rovině ze zadané množiny přímek, a odhadněte co nejlépe jeho (asymptotickou) časovou složitost.

   (Při popisu algoritmu je důležité popsat, z jakých dílčích kroků (popř. subalgoritmů) algoritmus sestává, ale není potřeba zacházet do podrobností ohledně implementace takových dílčích kroků.)

10 Relační databáze (otázka studijního zaměření – 3 body)

Uvažujme následující tabulky v relační databázi: *Person* a *Task*. Tabulka *Person* má následující strukturu :

- **ID** (int): primary key
- **Name** (varchar): jméno osoby
- **Age** (int): věk osoby

V tabulce Person jsou následující záznamy:

ID	Name	Age
1	Tom	25
2	Jane	30
3	Adam	29
4	Max	36

Tabulka *Task* má následující strukturu:

- **ID** (int): primary key
- **Description** (varchar): popis tasku
- **Deadline** (date): deadline pro splnění tasku
- **Done** (bool): informace, zda byl task splněn
- **PersonID** (int): foreign key vedoucí na ID v tabulce Person

V tabulce Tasks jsou následující záznamy:

ID	Description	Deadline	Done	PersonID
1	watch movie	2022-05-30	false	1
2	cook dinner	2022-05-31	false	1
3	plan holiday	2022-06-30	false	1
4	submit paper	2022-04-20	true	2
5	watch movie	2022-05-30	false	2
6	write thesis	2022-07-30	true	4

1. Napište SQL dotaz, který bude mít na výstupu stejnou strukturu jako tabulka *Person*. Dotaz vrátí tabulku, kde bude osoba uvedena právě jednou, pokud platí, že má přiřazený nesplněný úkol po uplynutí deadlinu (k datu zkoušky).
2. Napište, jaký výstup budou mít následující dotazy. V případě chyby v SQL dotazu vyznačte místo s chybou a chybu vysvětlete.
   - ```
     SELECT Name FROM Person WHERE ID Not In
     (SELECT PersonID From Task)
     ```
   - ```
     SELECT Name, Description FROM Person JOIN Task ON
     (ID = PersonID) WHERE Age < 30
     ```
3. V průběhu práce s databází bylo zjištěno, že relace mezi tabulkou Person a Task neodpovídá skutečnosti. Konkrétně k práci na jednomu tasku může být přiřazeno vícero (neomezeně) osob, z nichž jedna může být za provedení tasku zodpovědná. Upravte schéma relační databáze tak, aby odpovídalo popsaným skutečnostem a zároveň splňovalo první normální formu (1NF).

Při řešení můžete použít SQL dialekt běžně známých databází (Oracle, MySQL, PostgreSQL, MSSQL, . . .).

11 PHP + JavaScript (otázka studijního zaměření – 3 body)

Detail záznamu z tabulky *Person* si může přihlášený uživatel zobrazit pomocí následujícího PHP skriptu.

```
<?php
if(!isset($_COOKIE['isLoggedIn'])) {
  exit(401);
}
$mysqli = mysqli_connect("localhost", "user", "1234", "database");
$result = $mysqli->query("SELECT * FROM Person WHERE ID=".$_GET["userID"]);
$person = $result->fetch_assoc();
?>
<dl>
 <dt>Name</dt>
 <dd><?= $person["Name"] ?></dd>
 <dt>Age</dt>
 <dd><?= $person["Age"] ?></dd>
</dl>
<?php
$mysqli->close();
```

1. Najděte a popište bezpečností rizika výše uvedeného skriptu.
2. Napište PHP skript, který pro ID specifikované jako "userID" query argument, vrátí JSON s informacemi o daném uživateli z tabulky Person. Pro tento případ nemusíte řešit přihlášení uživatele. Pokud v řešení použijete knihovní funkce, krátce popište jejich význam (zejména pokud si nejste jisti přesným názvem).
3. Server vrací jako odpověď na HTTP GET požadavek na adresu "./person/{id}" JSON s informacemi z tabulky Person. Řetězec {id} je třeba nahradit za identifikátor uživatele. Například pro id=1, adresa "./person/1", vrátí server následující JSON:

   ```
   {"ID": 1, "Name": "Tom", "Age": 25}
   ```

   Napište JavaScript funkci, která má jediný argument "userID", který obsahuje ID uživatele. Po zavolání funkce provede HTTP GET požadavek, čímž získá JSON pro daného uživatele. Tento JSON pak využije aby vygenerovala následující fragment HTML kódu:

   ```
   <dl><dt>Name</dt><dd>{name}</dd></dl>
   ```

   Kde řetězec {name} je nahrazen hodnotou "Name" ze získaného JSONu. Daný fragment pak funkce vloží do DOM stromu, jako obsah elementu `<div>` s atributem `id="userDetail"`. Pokud v řešení použijete funkce poskytnuté webovým prohlížečem, krátce popište jejich význam (zejména pokud si nejste jisti přesným názvem).

12 CSS (otázka studijního zaměření – 3 body)

Uvažujte následující fragment HTML kódu:

```
<ul class="container">
  <li title="First">Item 1
      <ul class="level2">
        <li>Item 1.1</li>
        <li>Item 1.2</li>
      </ul>
  </li>
  <li id="item2">Item 2
      <ul>
        <li>Item 2.1</li>
        <li>Item 2.2
              <ul>
                <li class="emph">Item 2.2.1</li>
                <li class="emph">Item 2.2.2</li>
              </ul>
```

```
        </li>
      </ul>
  </li>
  <li><a href="#">Item 3</a></li>
  <li><a href="https://cuni.cz">Item 4</a></li>
  <li title="Last">Item 5</li>
</ul>
```

K HTML stránce obsahující zmíněný fragment jsou přilinkované následující CSS definice:

```
ul li {color: red;}
.emph li {color: green;}
li a[href^="http"] {color:brown}
#container li li {color: violet;}
ul li>li {color:blue; font-weight:bold;}
.level2 li {color:grey;}
li[title] {font-weight:bold;}
```

1. Vysvětlete princip určení specificity CSS selektorů (na čem specificita záleží, jak konkrétně se počítá). Který/které z výše uvedených CSS selektorů má nejvyšší specificitu a který/které má nejnižší specificitu?
2. Odpovězte na následující otázky a své odpovědi stručně zdůvodněte (např. uveďte, který selektor se použije).
   - Jaká je barva textu "Item 2.2.1"?
   - Jaká je barva textu "Item 3"?
   - Jaký text je zobrazen **tučně**?
   - Existují ve výše popsaných CSS definicích selektory, které se nenamapují na žádný HTML tag z ukázkového fragmentu? Pokud ano, jaké?
3. Navrhněte CSS pravidlo, které po najetí myší na odkaz “Item 3” zobrazí za odkazem text “Tooltip:...”. Používáte-li CSS selektory / vlastnosti u kterých si nejste jisti názvem, vysvětlete do komentáře jejich význam.

13 Teorie množin (otázka studijního zaměření – 3 body)

1. Definujte ordinální čísla.
2. Definujte kardinální čísla.
3. Existuje množina všech ordinálních čísel? Svou odpověď zdůvodněte.

14 Ramseyova teorie (otázka studijního zaměření – 3 body)

1. Formulujte Ramseyovu větu pro grafy.
2. Aplikací Ramseyovy věty pro grafy dokažte následující tvrzeni:
   Pro každe přirozené číslo $k > 1$ existuje přirozené číslo $M(k)$ takové, že pro každé obarvení množiny $\{1, 2, 3, \ldots, M(k)\}$ dvěma barvami existuje monochromatická podmnožina tvaru $\{x_1, x_2, x_3, \ldots, x_k, x_1 + x_2 + x_3 + \ldots + x_k\}$.

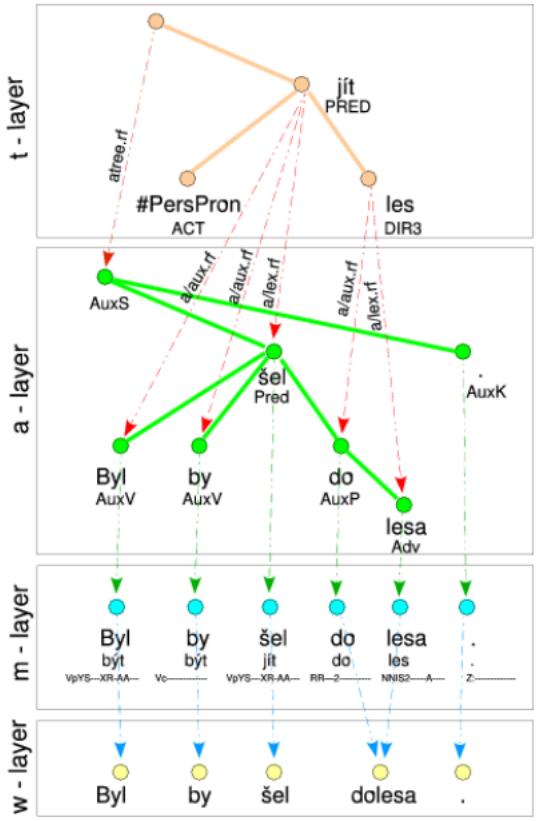
15 Hranová barevnost (otázka studijního zaměření – 3 body)

1. Definujte hranovou barevnost grafu.
2. Formulujte Vizingovu větu.
3. Pro každé $n \in \mathbb{N}$ určete hranovou barevnost grafu K_n.

16 Základní formalismy pro popis přirozeného jazyka (otázka studijního zaměření – 3 body)

1. Vysvětlete pojmy *korpus* a *anotovaný korpus*.

2. Následující obrázek znázorňuje vztahy mezi rovinami *Pražského závislostního korpusu*. Jednotlivé roviny popište.



17 Morfologická, syntaktická a sémantická analýza přirozeného jazyka (otázka studijního zaměření – 3 body)

1. Vysvětlete pojem ontologie ve zpracování sémantiky přirozeného jazyka.
2. Popište sémantickou síť Wordnet.
3. Vysvětlete pojem anafora a uveďte základní kategorie anafory v textu.

18 Jazykové modelování (otázka studijního zaměření – 3 body)

1. Zapište Bayesovský vzorec a vysvětlete význam pravděpodobností, které se v něm vyskytují.
2. Máme tři mince: 2 pravé a 1 falešnou. Pro pravé mince platí, že panna má stejnou pravděpodobnost $1/2$ jako orel. Pravděpodobnost, že padne panna na falešné minci, je $3/4$. Z těchto tří mincí náhodně vybereme jednu a dvakrát s ní hodíme. Výslednou hodnotu náhodné veličiny reprezentujeme jako vektor $\langle x_1, x_2 \rangle$, kde $x_i \in \{P, O\}$. V našem pokusu pozorujeme výsledek $\langle P, P \rangle$. Jaká je pravděpodobnost, že jsme házeli falešnou mincí?

19 Anti-aliasing (otázka studijního zaměření – 3 body)

1. Který z parametrů rastrového zobrazovacího zařízení je anti-aliasingem vylepšen? Jak se to pozná na výsledném obrázku?
2. Jak se může anti-aliasing implementovat v klasickém rastrovém vykreslování (rasterizace, např. při kreslení vektorové grafiky)?
3. Jak se anti-aliasing implementuje v prostředí paprskového zobrazovače (např. ray-tracing)?

20 Zobrazování 3D scény na grafické kartě (otázka studijního zaměření – 3 body)

1. Jaká primitiva umí GPU přímo zobrazit (v klasickém zobrazovacím řetězci pro 3D grafiku)? Napište, která primitiva se podle Vás používají nejčastěji.
2. Jak se posílají 3D data z aplikace do GPU? Uveďte jen přehledně několik příkladů, několik různých principů. Zamyslete se i nad globálními daty pro vykreslování: transformační matice, parametry pro stínování - jak ty se předávají do GPU?
3. Navrhněte rámcově efektivní systém organizace 3D scény a jejího odesílání do GPU. Předpokládejte, že scéna se skládá z pevné části (např. terén = povrch Země) a několika málo pohybujících se objektů, které se nedeformují, jen mění polohu a orientaci. Jde nám o racionální návrh koncepce, ne o implementační detaily.

21 Rotace v prostoru (otázka studijního zaměření – 3 body)

Jde nám o metody, jak matematicky reprezentovat rotaci pevného tělesa v 3D prostoru. Středem rotace bude pro jednoduchost počátek souřadnic. Těleso se kolem počátku bude jenom otáčet, to znamená, že nebude měnit svůj tvar ani velikost ("transformace tuhého tělesa" nebo anglicky "rigid body transform").

1. Navrhněte první metodu, kterou byste zvolili pro případ, že bude třeba často rotaci interpolovat – tj. animovat dané těleso (později se budeme snažit vyjádřit plynulý a přirozený animační pohyb).
2. Navrhněte ještě jinou metodu, jak rotaci vyjádřit. Může mít proti té první některé výhody (rychlost, jednoduchost, snadnost GPU implementace), ale i nevýhody. Výhody a nevýhody uveďte a alespoň stručně zdůvodněte.
3. Jak byste postupovali při animaci rotace v čase? Vyberte si první nebo druhou metodu a dostatečně přesně popište matematický postup animace (i u této podotázky se zabývejte jen rotacemi).

22 Spouštění procesů a relokace (otázka studijního zaměření – 3 body)

1. Zvolte si libovolnou vhodnou instrukci procesoru a vysvětlete na ní rozdíl mezi absolutní a relativní adresací (není třeba použít žádný konkrétní procesor, můžete si vymyslet libovolnou vlastní instrukci, pokud bude přiměřeně realistická).
2. Pro zvolený příklad z prvního bodu ukažte, jaké kroky musí provést operační systém, pokud program s danou instrukcí přesouvá (relokuje). Uvažujte odděleně případ s absolutní a relativní adresací.
3. Zde je krátký program v jazyce C, který sečte (s přetečením) pole 1024 čísel:

```
int data [1024];
int sum = 0;

void summarize () {
    for (int i = 0 ; i < 1024 ; i ++) {
        sum += data [i];
    }
}
```

Zde je jeden z možných způsobů, kterým překladač jazyka C může uvedený program přeložit do asembleru procesoru MIPS – první sloupec udává šestnáctkovou adresu, druhý sloupec šestnáctkový kód instrukce, zbytek řádku je přepis kódu instrukce do mnemoniky asembleru:

```
800010a0:       3c068000        lui     a2,0x8000
800010a4:       3c028000        lui     v0,0x8000
800010a8:       8cc31500        lw      v1,5376(a2)
800010ac:       24421510        addiu   v0,v0,5392
800010b0:       24451000        addiu   a1,v0,4096
800010b4:       8c440000        lw      a0,0(v0)
800010b8:       24420004        addiu   v0,v0,4
800010bc:       00641821        addu    v1,v1,a0
800010c0:       1445fffc        bne     v0,a1,800010b4
800010c4:       00000000        nop
800010c8:       acc31500        sw      v1,5376(a2)
800010cc:       03e00008        jr      ra
```

```
800010d0:       00000000        nop
```

Podle výpisu rozhodněte, zda je uvedený program pozičně závislý nebo pozičně nezávislý. Zdůvodněte své rozhodnutí a ve výpisu vyznačte, o které informace se v rozhodování opíráte.

Zde je jako pomůcka rozepsaná sémantika jednotlivých instrukcí výpisu. Registr `pc` ukazuje na následující vykonávanou instrukci, registr `ra` obsahuje návratovou adresu, ostatní uvedené registry nemají zvláštní použití:

```
lui     a2,0x8000       ;register a2 = 0x80000000
lui     v0,0x8000       ;register v0 = 0x80000000
lw      v1,5376(a2)     ;register v1 = read from memory address [a2 + 0x1500]
addiu   v0,v0,5392      ;register v0 = v0 + 0x1510
addiu   a1,v0,4096      ;register a1 = v0 + 0x1000
lw      a0,0(v0)        ;register a0 = read from memory address [v0]
addiu   v0,v0,4         ;register v0 = v0 + 4
addu    v1,v1,a0        ;register v1 = v1 + a0
bne     v0,a1,800010b4  ;if v0 != a1 jump to address (pc + 0xFFFFFFFC*4)
nop                     ;do nothing
sw      v1,5376(a2)     ;write register v1 to memory address [a2 + 0x1500]
jr      ra              ;jump to address (ra)
nop                     ;do nothing
```

23 Rozhraní pro synchronizaci (otázka studijního zaměření – 3 body)

1. Napište rozhraní podmínkové proměnné (condition variable) jako objektu pro synchronizaci. Popište sémantiku metod.
2. Napište implementaci funkcí `produce` a `consume`, které si pomocí sdíleného pole omezené maximální délky předávají (reference na) objekty typu `item`. Funkce musí fungovat i při volání více vlákny, volání `produce` musí (pasivně) čekat v případě, že sdílené pole je plné, volání `consume` musí (pasivně) čekat v případě, že sdílené pole je prázdné. K synchronizaci použijte podle potřeby zámky a podmínkové proměnné (condition variables).

Dodržení přesné syntaxe konkrétního programovacího jazyka není nutné, odpovědi mohou použít pseudokód.

24 Zasílání zpráv (otázka studijního zaměření – 3 body)

Následující program je upravený minimální příklad odeslání a příjmu zprávy z dokumentace middleware JGroups:

```
JChannel channel = new JChannel ();
channel.setReceiver (new ReceiverAdapter () {
    public void receive (Message msg) {
        System.out.println (msg.getObject ());
    }
});
channel.connect ("MyCluster");
channel.send (new ObjectMessageSerializable (null, "Hello␣!"));
channel.close ();
```

1. Middleware v uvedeném příkladu doručuje zprávu pomocí callback funkce. Middleware typicky volá takovou callback funkci z nějakého vlastního vlákna. Navrhněte, jak přesně by implementace middleware měla využívat vlákna, aby program mohl zpracovávat více zpráv současně. Mechanismus popište v dostatečném detailu (tedy kdy jaké vlákno vzniká a zaniká a kdy a na co čeká). Můžete použít pseudokód i prostý text.
2. Middleware v uvedeném příkladu rozešle zprávu všem připojeným členům skupiny `MyCluster`. Navrhněte, jakým způsobem by middleware mohl takovou komunikaci implementovat v prostředí běžné IP sítě. Počítejte s tím, že komunikace má být spolehlivá (odolná proti ztrátě zpráv), a že seznam členů skupiny se může měnit (členové se mohou kdykoliv připojovat a odpojovat).

   Mechanismus popište v dostatečném detailu (tedy kdy a jaké zprávy a na jaké adresy se budou posílat). Můžete popsat reálnou komunikaci z middleware JGroups, nebo navrhnout vlastní rozumné složitosti (například je v pořádku, pokud bude mechanismus vyžadovat částečně ruční konfiguraci například význačných uzlů nebo význačných portů, nebo bude například fungovat jen v lokální síti).