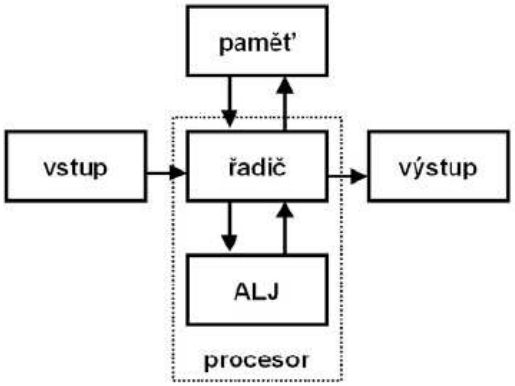
Otázky k teoretické části zkoušky z předmětu IAJCE 2006/2007

1. Nakreslete blokové schéma počítače Neumannova typu a popište jeho základní komponenty



Vstupní a výstupní zařízení – výměna informací mezi počítačem a okolím, vstupní zařízení = převod informací do číselné podoby, výstupní zařízení = opačný převod do podoby srozumitelné okolí.

Procesor (CPU) – řadič + ALU. Řadič – dává pokyny ke zpracování dat, produkuje výstupní data. ALU – podle pokynů řadiče provádí jednotlivé výpočty a zpracovává data.

Paměť – rozdělena na stejně velké buňky, které jsou průběžně očíslované (=adresa). Neumannova koncepce – jedna paměť – program + data. Výhody – Volitelně kolik program a kolik data, stroj může modifikovat sám sebe (program jehož výstupem je jiný program). Po sobě jdoucí instrukce programu se uloží do paměťových buněk jdoucích po sobě, přístup k následující instrukci se uskuteční z řídící jednotky zvýšením instrukční adresy o 1.

2. Popište CPU, vysvětlete pojmy: instrukce, instrukční soubor

Procesor (CPU) – řadič + ALU. Řadič – dává pokyny ke zpracování dat, produkuje výstupní data. ALU – podle pokynů řadiče provádí jednotlivé výpočty a zpracovává data. Dělení podle počtu bitů : 8b., 16b., 32b., 64b. – určuje výpočetní výkon.

CPU vykonává jednoduché, elementární operace = **instrukce** (+-*/, přesuny dat CPU ↔ paměť, paměť ↔ paměť, změna pořadí vykonávaných instrukcí (podprogramy, skoky)).

Množina všech instrukcí CPU = **instrukční soubor** (10-ky až 100-ky instrukcí).

Každý procesor = jiný instrukční soubor.

3. Vysvětlete pojmy bit, byte a popište tvorbu násobků těchto jednotek (zejména rozdíl k/K)?

Bit – základní jednotka informace, jedna číslice ve dvojkové soustavě, nejmenší část paměti počítače.

Byte – 8 bitů.

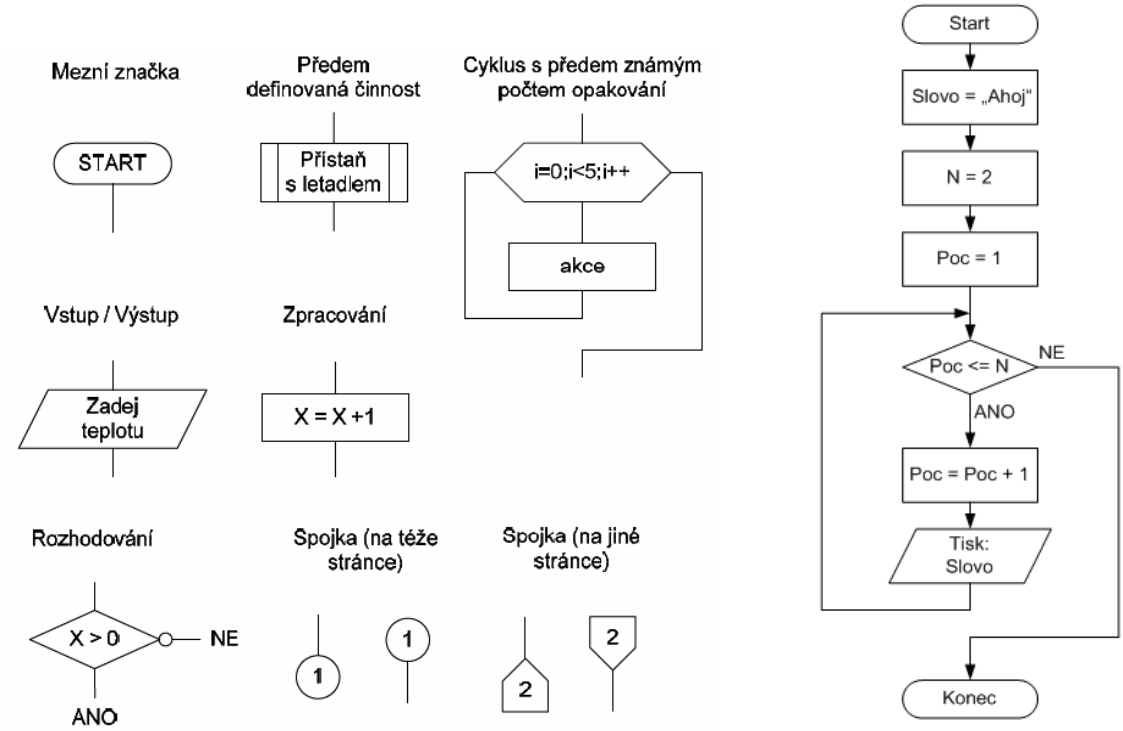
Násobky se základem 2 nikoli 10. K (kilo) = 2^{10} = 1024, k = 1000, M (mega) = 2^{20} = 1 048 576, G (giga) = 2^{30}, atd.

4. Definujte algoritmus. Napište a charakterizujte základní vlastnosti algoritmů.

Definice = přesný návod či postup, kterým lze vyřešit daný typ úlohy. V programování – teoretický princip řešení problému (oproti přesnému zápisu v konkrétním programovacím jazyce)

Vlastnosti – **Konečnost** – algoritmus musí skončit v konečném počtu kroků.
Determinismus – další krok algoritmu – jednoznačně definován.
Vstupy – data, která jsou mu předány před započetím jeho provádění, nebo v průběhu jeho činnosti.
Výstupy – odpovědi na problém, který algoritmus řeší.
Efektivita – každá operace algoritmu provedena v konečném čase.
Obecnost – algoritmus řeší obecnou třídu problémů.

5. Vývojové diagramy – značky, příklad.



6. Proveďte základní rozdělení programovacích jazyků.

Dle míry abstrakce od HW počítače: strojově orientované, vyšší programovací jazyky.

Dle využitelných programátorských technik: strukturované – Pascal, C; OOP – C++, Java, C#

Dle účelu: pro „spustitelné aplikace“, webové aplikace, skriptovací, atd.

7. Popište vztah vyššího programovacího jazyka, assembleru a strojového kódu.

Strojový kód – programuje PC na úrovni jeho instrukcí

Assembler – prog. jazyk velice blízký strojovému kódu, jazyk symbolických adres, tvoří jej pouze zástupné symboly, které přímo odpovídají strojovému kódu.

Vyšší prog. jazyk – srozumitelnější, logická struktura programů, nezávislé na strojových principech počítače, př. Pascal, Basic, Fortran, C, C++, C#, Java, PHP, ASP, …

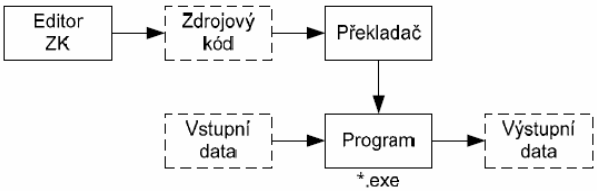
8. Definujte pojmy syntaxe a sémantika programovacího jazyka. Uveďte příklad (např. jeden z příkazů).

Syntaxe – souhrn pravidel udávajících příslušné tvary dílčích konstrukcí a celého programu, nutné dodržovat

Př.
`if plat < 5000` ← špatně
`if (plat < 5000)` ← správně.

Sémantika – význam jednotlivých konstrukcí, slovní popis nebo vývojové diagramy `if (a > 3)` → jestliže je `a > 3`, pak …

9. Popište mechanizmus překladu a ladění programu z vyššího programovacího jazyka (kompilační implementace), vysvětlete a přeložte pojmy: Compile, Compiler, Debug, Debuger.



Z editoru zdrojového kódu se zdrojový kód přeloží v překladači – výsledkem je spustitelný soubor *.exe. Ten sám zpracovává vstupní a výstupní data.

Ladění programu se odehrává za pomoci ladícího programu Debuggeru, ten na pozadí programu prochází jednotlivé řádky zdrojového kódu. Umožňuje zobrazení jednotlivých proměnných, apod.

Compile – přeložit – proces vytvoření programu ze zdrojového kódu.

Compiler – překladač – program, který umožňuje vytvoření programu.

Debug – ladit – proces odstranění chyb z programu

Debugger – ladící program, který umožňuje ladění.

10. Vysvětlete a popište pojmy: proměnná, identifikátor, klíčové slovo, konstanta, literál.

Proměnná – místo v paměti (nebo také tzv. datový objekt), označené jménem; je v ní uložená hodnota, která se může v průběhu práce programu měnit.

Identifikátor – jakékoli uživatelské jméno v našem programu, pojmenovat můžeme (spíše musíme) proměnnou, konstantu, metodu, …

Klíčové slovo – zvláštní identifikátory, které mají speciální význam pro překladač jazyka.

Konstanta – konstantní proměnné, musí se deklarovat přímo v definici, dál se nesmí měnit.

Literál – jakýkoli neproměnný objekt datové povahy v programu.

11. Popište deklaraci proměnné a deklaraci s inicializací.

Deklarace = příkaz, který zavede v programu novou proměnnou, a který vyhradí v paměti místo pro proměnnou → alokace paměti.

Deklarace s inicializací – po deklaraci proměnné je proměnné přiřazena hodnota: `0`, `0.0`, `false`, `''`. Při deklaraci lze proměnné přiřadit určitou hodnotu (inicializovat proměnnou). Např. `int pomoc = 10;`

12. Definujte pojem datový typ a proveďte jejich základní rozdělení (jednoduché-strukturované, celá čísla, reálná čísla apod.)

Každá proměnná musí být určitého datového typu. Datový typ specifikuje – množinu hodnot, jakých může nabývat, množinu operací, které lze z datovým typem provést.

Jednoduché:

- celá čísla – bez znaménka – `byte` (8b), `ushort` (16b), `uint` (32b), `ulong` (64b).
  – se znaménkem – `sbyte` (8b), `short` (16b), `int` (32b), `long` (64b).
- reálná čísla – `float` (32b), `double` (64b), `decimal` (128).
- další – `bool` – logické hodnoty, `char` – jeden znak, `string` – řetězec znaků.

Strukturované:

- pole – skládá se z většího množství položek, všechny prvky pole stejný datový typ → pole homogenní datový struktura, indexy pole – celá nezáporná čísla → indexy 0 až N-1. Jedná se o referenční proměnnou, pracuje se s odkazem, nebo přímo s jednotlivými prvky pole.
- třída – heterogenní datový struktura, také referenční proměnná.

13. Popište mechanizmus přiřazení

Naplnění proměnné hodnotou. Hodnota, kterou chceme do proměnné přiřadit, se uloží na určité místo (adresu) v paměti. Místo v paměti určuje operační systém.

14. Definujte výraz. Z čeho se skládá? Co může být operandem ve výrazu? Uveďte příklady.

Výraz je vše na pravé straně přiřazení. Výpočet hodnoty určitého datového typu, výsledek je **hodnota**.

Skládá se z: operandů (proměnných, konstant, volání metod), operátorů (+-*/ atd.), závorek.

Příklady:
```
novyPlat = 2*(staryPlat + sin(20*PI));
novyPlat = 2*(staryPlat + sin(20*PI));
novyPlat = 2*(staryPlat + sin(20*PI));
novyPlat = 2*(staryPlat + sin(20*PI));
```

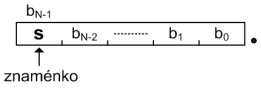
15. Proveďte základní rozdělení operátorů, vysvětlete pojmy priorita a asociativita.

Binární – sčítání +, odčítání -, násobení *, celočíselné dělení /, reálné dělení /, dělení modulo %.
Unární – + (nemá význam), - (otáčí znaménko).
Inkrementační a dekrementační – ++ zvětší o 1, -- zmenší o 1.
Logické – AND &&, OR ||, NOT !
Relační - > větší, < menší, >= větší nebo rovno, <= menší nebo rovno, == rovno, nerovno !=

Priorita – který operátor ovlivní operandy jako první

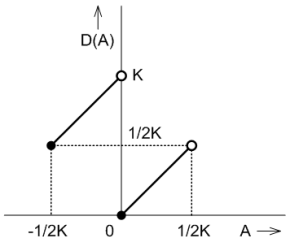
Asociativita – směr vyhodnocování operátorů (zleva, zprava), priorita vyhodnocení operátorů → výraz obsahuje operátory stejné priority

16. Jak jsou uložena záporná čísla v počítači? Ukažte na příkladu (vymyslete si příklad čísla ve 2D – umět vysvětlit!!!)



Nejvyšší bit → znaménko, 1 = záporné číslo, 0 = kladné číslo.

Uložen jako dvojkový kód (dvojkový doplněk)

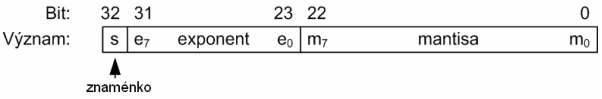


Obraz čísla A v dvojkovém kódu D(A):
D(A) = A pro A>=0
D(A)=Z+A pro A<0

Nula je nula.
Nejmenší číslo je -2^{N-1}
Největší číslo je $2^{N-1}-1$

K – kapacita čísla - $K = 2^N$

17. Jak jsou v počítači uložena reálná čísla?



Uloženo jako: $A = (-1)^S .2^{\exp onent-127}.1, mantisa$

mantisa určuje přesnost, exponent rozsah.

18. Vysvětlete pojem přetypování. Ukažte na příkladu oba typy (impl. vs. expl.)

Přetypování = změna datového typu

Implicitní přetypování – prováděna automaticky, = konverze z nižších typů na vyšší – tam, kde nemůže dojít ke ztrátě informace. Např. `byte` → `int` → `double`.

Explicitní přetypování – konverze z vyšších typů na nižší, dojde ke ztrátě informace!!!, neprobíhá automaticky → musí se do programu zapsat. Syntaxe: `(novy typ)vyraz`. Např. `(int)13.5;`

19. Popište syntaxi a sémantiku příkazu `if`, `if-else`

Syntaxe `if`:

```
if (BV)
    prikaz;
```

Sémantika : jestliže je výsledkem `BV` `true`, pak proveď `prikaz` (jeden!!!), jinak `prikaz` přeskoč.

Syntaxe `if-else`:

```
if (BV)
    prikaz1;
else
    prikaz2;
```

Sémantika: jestliže je výsledkem `BV` `true`, pak proveď `prikaz1` (jeden!!!), jinak proveď `prikaz2` (jeden!!!).

20. Popište syntaxi a sémantiku příkazu `for`

= cyklus s předem známým počtem opakování – známe: počáteční hodnotu řídící proměnné, koncovou hodnotu řídící proměnné, způsob ovlivnění řídící proměnné po každém průchodu cyklu

Syntaxe:

```
for (IŘP, BV, ZŘP)
    prikaz;
```

Sémantika: před vstupem do cyklu – inicializace řídící proměnné `IŘP`. Poté se opakuje tělo cyklu (`prikaz`) tak dlouho, dokud má `BV` hodnotu `true`. Po každé obrátce cyklu – změna řídící proměnné daná výrazem `ZŘP`. `BV` se testuje před každou obrátkou – cyklus nemusí proběhnout ani jednou.

21. Popište syntaxi a sémantiku příkazu `switch`

Umožňuje několikanásobné větvení programu.

Syntaxe:

```
switch (vyraz)
{
    case konstanta1:
        prikaz1;
        prikaz2;
        prikazN;
        break;
    case konstantaM:
        prikaz1;
        prikaz2;
        prikazN;
        break;
    default:
        prikaz1;
        prikaz2;
        prikazN;
        break;
}
```

Sémantika: Vypočte se hodnota `vyraz` odpovídající větev `case` (hodnota `vyraz` → skok na odpovídající `konstanta?`) provedou se příkazy v dané větvi až do `break`. Pokud není odpovídající konstanta nalezena – provede se větev `default` (nemusí být uvedena, pak se neprovede nic).

Vyraz musí být: celočíselného typu (`int`, `sbyte`, atd.), výčtového typu `enum`, řetězec.

22. Popište syntaxi a sémantiku příkazu `do-while`

Syntaxe:

```
do
    prikaz;
while (BV);
```

Sémantika: Opakuj `prikaz` tak dlouho, dokud má `BV` hodnotu `true`. Cyklus testuje podmínku až po vykonání těla cyklu – cyklus proběhne alespoň **jednou**.

23. Popište syntaxi a sémantiku příkazu `while`

Vhodný, pokud ukončovací podmínka závisí na příkazu v těle cyklu – počet opakování není předem znám.

Syntaxe:

```
while (BV)
    prikaz;
```

Sémantika: opakuj `prikaz`, dokud má `BV` hodnotu `true`. `BV` se testuje před každým průchodem cyklu – cyklus nemusí proběhnout ani jednou.

24. Vysvětlete pojmy: metoda, hlavička a tělo metody, návratová hodnota, definice a volání metody, formální a skutečné parametry metod.

Metoda – „podprogram“, pro opakování části kódu. = logický celek, řešící dílčí část problému.

Hlavička metody:
`static double` NactiStranu (`int` kterou)

- `static` = modifikátor přístupu
- `double` = typ návratové hodnoty
- `NactiStranu` = jméno metody (identifikátor)
- `(int kterou)` = deklarace formálních parametrů

Tělo metody – výkonný kód, zapsaný mezi `{}`

Návratová hodnota: `return hodnota;` = výstup návratové hodnoty z metody.

Definice metody = místo ve zdrojovém kódu, kde je zasána hlavička + tělo metody.

Volání metody: `strana = NactiStranu(1);` – formální parametr byl nahrazen skutečným parametrem `1`, návratová hodnota se uloží do proměnné `strana`. Formální parametry jsou nahrazeny skutečnými.

Formální parametry – hodnota předávaná do metody z vnějšku.

Skutečné parametry – hodnota předávaná do metody v jejím volání.

25. Vysvětlete mechanizmus volání metody (volání hodnotou).

V paměti se vytvoří zásobník se všemi lokálními proměnnými a proměnnými formálních parametrů se do nich zkopírují odpovídající hodnoty skutečných parametrů. Návratová hodnota je uložena na zásobník. Po návratu z metody jsou všechny formální parametry zrušeny a původní skutečné parametry zůstanou nezměněny.

26. Vysvětlete volání metod odkazem a význam klíčových slov `ref` a `out`. Uveďte příklad.

Volání odkazem – překladač provede volání metody místo s kopiemi se skutečnými parametry funkce. Změnou formálních parametrů dojde i ke změně skutečných parametrů. Příkaz `ref` zajistí volání odkazem, pouze již inicializovaných proměnných. Příkaz `out` také volání odkazem, ale proměnné nemusí být inicializovaná.

Př.

```
// Definice metody
static void SoucetRozdil(double a, double b, out double soucet, out double rozdil)
{
    soucet = a + b;
    rozdil = a - b;
}
//Volání metody
double cislo1 = 3, cislo2 = 2, soucet, rozdil;

SoucetRozdil(cislo1, cislo2, out soucet, out rozdil);
```

27. Vysvětlete pojem přetěžování metod a uveďte možné způsoby přetížení.

Přetěžování = metody se stejným názvem, ale hlavička (parametry) různé.

Metodu **lze** přetížit: počtem parametrů, typem parametrů, pořadím parametrů – nedělat!!!, kombinace.
Metodu **nelze** přetížit pouze typem návratové hodnoty.

Příklady: stejná činnost na různých typech parametrů, různým počtem parametrů.

28. Vysvětlete pojem referenční proměnná a ukažte na příkladu základní principy práce (např. na poli)

Referenční proměnná obsahuje adresu paměti, kde jsou uloženy data. Při práci s referenční proměnnou pracujeme pouze s odkazem.

Př.

```
int[] pole; // Vytvoření pole, které neobsahuje žádná data, na žádná
            //neukazuje
int[] pole1 = new int[3]; // Vytvoření odkazu a 3 prvků pole
int[] pole2 = new int[5]; // Vytvoření odkazu a 5 prvků pole

pole1 = pole2; // Původní pole1 zanikne a nyní obsahuje pole1 stejná data
               // jako pole2
```

29. Popište princip uložení znaků v počítači. Vysvětlete pojem znaková sada, charakterizujte znakové sady: ASCII, 8b sady, Unicode.

Každému znaku je přiřazeno nějaké nezáporné celé číslo → kódování.

Znaková sada – které znaky budou kódovány na jaká čísla.

ASCII – 7-bitová, čísla 0-127, uložení jako 1B (horních 128b nevyužito)

8b sady – rozšíření ASCII, např. Win1250 až Win1258 - Windows, ISO8859-1 až ISO8859-15 – mezinárodní standard, starší distribuce Linuxu. Dolních 128b → ASCII, horních 128b → znaky národních abeced.

Unicode – znaková sada, která pojme všechny národní abecedy. Původně 16b, verze 2.0 32b (používá jen 21b). Různé charsety UTF-8, UTF-16, UTF-32.

30. Popište princip algoritmu Select sort.

1. V poli nalezneme prvek s nejmenší hodnotou a zapamatujeme si jeho pořadí.
2. Vyměníme tento prvek s prvkem na prvním místě.
3. Postupně opakujeme body 1. a 2. nad zbývající, nesetříděnou částí pole.

31. Popište princip algoritmu Insert sort

1. Máme setříděno `k` prvků.
2. Pro `(k+1)`-tý prvek nalezneme místo v již setříděné posloupnosti → zbytek posloupnosti se odsune
3. Postupně opakujeme body 1. a 2. mad zbývající, nesetříděnou částí pole.

32. Popište princip algoritmu Bubble sort

1. Porovnáváme vždy dva sousední prvky směrem od konce posloupnosti
2. Pokud `pole[i+1] < pole[i]` → prohodíme je; konec na prvním prvku
   - nejmenší prvek → na první pozici – „probublá“ posloupností
   - částečně se setřídí posloupnost
3. Postupně opakujeme body 1. a 2. → konec u druhého kroku v pořadí atd.

33. Výjimky – základní pojmy, principy (blok `try-catch`, propagace výjimky)

Výjimky = moderní způsob ošetření chyb

Princip a základní pojmy:

1. V programu dojde k chybě
2. Běh programu se přeruší → vyhodí se výjimka (throw exception)
3. Program se pokusí výjimku zachytit (catch exception) = ošetřit chybu ve spec. části kódu

Ošetření výjimky – blok `try-catch`

- „nebezpečný" kód do bloku `try`
- vyhození výjimky → program pokračuje v bloku `catch`

Propagace výjimky – pokud je výjimka vyhozena v metodě a není zachycena, šíří se do nadřazené úrovně volání.

34. Vysvětlete pojem složitost algoritmů a způsoby zjištění složitosti.

Složitost – vztah daného algoritmu k daným prostředkům. Omezení zejména **časem** a **velikostí paměti**. Časová složitost – každé množině vstupních dat přiřazuje počet operací při výpočtu podle stanoveného algoritmu. Paměťová složitost – závislost paměťových nároků na vstupních datech.

Zjištění složitosti:

- přesné zjištění složitosti – stanovení složitosti v závislosti na konkrétních datech a rozsahu dat.
- odhad složitosti (u triviálních algoritmů) – každý vnořený cyklus zvyšuje mocninu složitosti o jednu, případné zkracování cyklů se zanedbává, u pole je C(n) = n – složitost je lineární, u matice je $C(n)=n^2$ – složitost je kvadratická.

35. Napište základní třídy složitosti spolu s jejich základní charakteristikou (příklad algoritmu s danou třídou složitosti)

Konstantní O(1) – ideální případ – jednoduché matematické operace.

Lineální O(n) – jeden cyklus v algoritmu, např. sekvenční vyhledávání, násobení vektoru skalárem, …

Kvadratická $O(n^2)$ – algoritmus se dvěma vnořenými cykly, např. třídící algoritmy, sčítání matic.

Logaritmická O(log n) – binární vyhledávání, v každém kroku se rozsah dat snižuje na polovinu.

Loglineární O(n log n) – pro algoritmy „rozděl a panuj", třídící algoritmus Quick sort, FFT (rychlá Fourierova transformace).