INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA STROJNÍ

ZÁKLADY INFORMATIKY

Ing. Roman Danel, Ph.D.

Ostrava 2013

ISBN 978-80-248-3052-0

Tento studijní materiál vznikl za finanční podpory Evropského sociálního fondu (ESF) a rozpočtu České republiky v rámci řešení projektu: CZ.1.07/2.2.00/15.0463, MODERNIZACE VÝUKOVÝCH MATERIÁLŮ A DIDAKTICKÝCH METOD

OBSAH

1 ALGORITMY A PROGRAMOVACÍ JAZYKY

OBSAH KAPITOLY:

MOTIVACE:

V této kapitole se naučíte co to je algoritmus, jaké má vlastnosti a v další části se budeme zabývat programovacími jazyky. Vysvětlíme si také členění programovacích jazyků a stručný náhled na jejich historický vývoj. V závěru se zmíníme o CASE nástrojích.

CÍL:

Po prostudování budete znát, jaký je rozdíl mezi kompilací a interpretací programovacích jazyků a k čemu je při kompilaci linker. Budete znát základní členění programovací jazyků a seznámíte se se základním programovacími konstrukcemi, jako j cyklus, rozhodovací blok a co to je proměnná.

1.1 ALOGRITMY

Algoritmus je **návod či postup jak vyřešit určitou úlohu**.

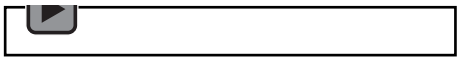
Příkladem algoritmu může být kuchyňský recept.

Matematický základ algoritmu:
- Lambda kalkul
- Turingův stroj (algoritmus je procedura, proveditelná turingovým strojem)

Poznámka
Turingův stroj je teoretický model počítače – konečný automat - program v podobě pravidel a nekonečná páska pro zápis mezivýsledků.

Algoritmus může být vyjádřen různými způsoby:
- Slovním popisem
- Vývojovým diagramem
- Pseudo-kódem
- Zdrojovým kódem
- Schémata, grafy

Audio 1.1



1.1.1 Vlastnosti algoritmů

Konečnost (finitnost)
- Musí skončit v konečném počtu kroků

Obecnost
- Neřeší jeden problém, ale třídu problémů

Determinovanost
- V každé situaci musí být jasné, co se má provést a jak má provádění pokračovat

Resultativnost (výstup)
- Musí mít aspoň jeden výstup (odpověď na řešený problém)

Elementárnost
- Skládá se z konečného počtu elementárních kroků

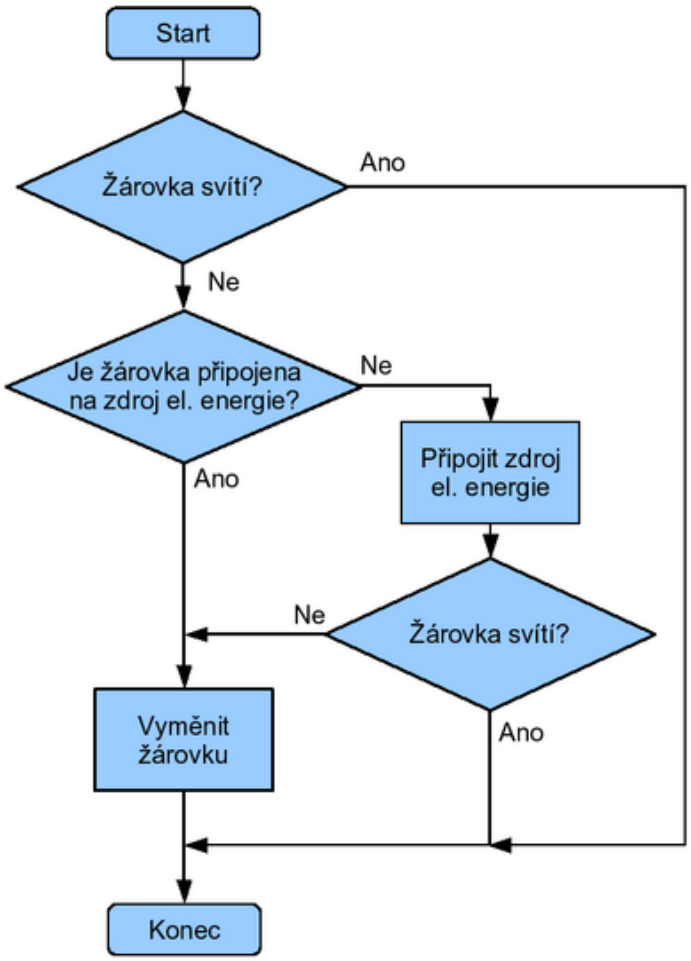
1.1.2 Složitost algoritmu

- **Algoritmická analýza** – zabývá se efektivitou algoritmů (jak z množiny možných algoritmů vybrat ten nejlepší)
- **Teorie složitosti** – otázka efektivity algoritmů - složitost – jak je algoritmus rychlý
- **Konečnost algoritmu** – lze úlohu vyřešit?

1.1.3 Některé typy algoritmů

- **Rekurzivní –** volají ve smyčce samy sebe
- **„Hladové“** (Greedy Search) – řešení po jednotlivých rozhodnutích, která jsou-li učiněna, už nejsou revidována, lokální minima, do této kategorie spadá například problém „obchodního cestujícího“
- **Dynamické programování** – řeší problém od nejjednodušších částí po nejsložitější a využívá výsledky již vyřešených podproblémů – metody z optimalizace
- **„rozděl a panuj“** (Divide and Conquer) – dělí problém na triviální části, které lze řešit přímo – FFT (Fast Fourier Transformation), Quick Sort, …
- **Pravděpodobnostní** (probabilistické) – binární výpočetní strom, v každém uzlu „hod mincí“
- **Genetické** (evoluční) – napodobování biologických evolučních procesů
- **Heuristické** – nemá za cíl nalezení přesného řešení, ale pouze vhodného přiblížení (používají je například antivirové programy pro odhad sekvencí neznámých virů).

1.1.4 Vyjádření algoritmu vývojovým diagramem (flowcharts)



Obr. 1 Ukázka řešení algoritmu pomocí vývojového diagramu
[Zdroj: http://cs.wikipedia.org/wiki/Soubor:Vyvojovy_diagram_zarovka.png]

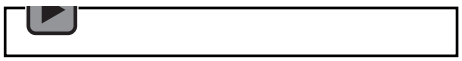
1.2 PROGRAMOVACÍ JAZYKY

Program - zápis algoritmu v programovacím jazyku.
Programovací jazyk – soubor pravidel pro zápis algoritmu.
Programovací jazyk má svoji **syntaxi** (souhrn pravidel) a **sémantiku** (množina slov a pravidla, která jim přiřazují význam).
Zdrojový text programu – algoritmus zapsaný v programovacím jazyku.
Strojový kód – instrukce srozumitelné pro daný stroj.
Instrukce – příkaz, kterému rozumí CPU (procesor).
Instrukční sada – soubor instrukcí daného procesoru.

Podle míry abstrakce dělíme programovací jazyky do dvou skupin:

- Nižší – závislé na HW (na procesoru) – např. assembler
- Vyšší – nezávislé na HW

Audio 1.2



Členění dle způsobu překladu:

- **Interpretované**
- **Kompilované**

Při **kompilaci** dochází k převodu programu zapsaného v programovacím jazyce do strojového kódu. Program je **kompilátorem** příslušného programovacího jazyka zkompilován pouze tehdy, pokud neobsahuje žádné syntaktické chyby. Kompilací vzniklý soubor ještě není přímo spustitelný, ale musí být sestaven do spustitelného tvaru pomocí **spojovače (linkeru)**. Linker spojí dohromady moduly a knihovní soubory podle závislostí, řeší globální proměnné a relativní adresaci. Výsledkem linkování je spustitelný program (v případě OS Windows soubor s příponou EXE).

Používané techniky optimalizace – odstranění mrtvého kódu, vkládání těl metod, rozbalení smyček, odstraněné shodných podvýrazů…

U **interpretace** je zdrojový kód čten řádek po řádku a ihned vykonáván. Výhodou je pružnost a možnost přepínání mezi vývojovým prostředím a běžícím programem. Vykonávání programu je ale pomalejší, protože na rozdíl od kompilace zde není možnost optimalizace kódu.

Formy interpretace:

- Čistá interpretace
- Překlad do pseudokódu a jeho interpretace (např. Java)
- JIT (Just In Time) interpretace – před interpretací je program kompilován a optimalizován

1.2.1 Chyby v programech

- Syntaktické
- Chyby při zpracování
- Sémantické (logické) chyby

Proces odstraňování chyb v programu se nazývá **ladění (debugging)**. Prostředek pro procházení programu po jednotlivých příkazech a možností výpisu hodnoty proměnných se nazývá debuger.

Audio 1.3

1.2.2 Assembler

- Nízkoúrovňový – binární zápis instrukcí
- Adresy se označily symbolickými jmény
- Vznik v 50. letech – druhá generace
- „assembler“ (z anglického Assembly Language)
- Assembler je technicky překladač jazyka symbolických adres
- Symbolická reprezentace strojových instrukcí

1.2.3 Vyšší programovací jazyky

Procedurální – strukturované – Fortran, Algol, Cobol, Basic, PL/1, dbase, clipper, FoxPro,…
Neprocedurální (deklarativní) - SQL
Funkcionální – LISP, APL, Haskell, Erlang
Objektově orientované – SmallTalk, C++, Java
Logické – Prolog, Absys, Planner
Značkové – HTML, XML (1996)
Skriptovací – JavaScript, PHP, Lua, Perl, Python,…

CASE sensitivita – je-li programovací jazyk case sensitivní, znamená to, že rozlišuje velká a malá písmena v názvech proměnných, příkazů, funkcí…

Poznámka
Deklarativní programovací jazyky jsou založeny na popisu cíle – přesný algoritmus je záležitostí překladače. Příkladem je databázový dotazovací jazyk SQL – příkazem SQL říkáme databázovému stroji co chceme dělat (např. načíst data za určitých podmínek), ale neříkáme mu, jak má tento příkaz provést.

Funkcionální jazyky zacházejí s výpočtem jako s vyhodnocením funkcí, aplikace je složena z funkcí, masivní využití rekurzí. Funkcionální jazyky jsou nejčistší implementací lambda kalkulu. Využití v umělé inteligenci, AUTOCAD (AutoLisp), unixový editor Emacs atd.

Audio 1.4

1.2.4 Základní prvky programu

- Proměnná
- Přiřazení
- Cyklus
- Podmínky pro větvení

Proměnná = úložiště informace – má vyhrazené místo v paměti
Proměnná má **typ** a **hodnotu.**

Příklad deklarace proměnné z jazyka C:

```
int i;              // deklaruji proměnnou i typu integer (celé číslo bez desetin)
i = 5;              // do proměnné i ukládám hodnotu 5
```

Ukázka v jazyce Basic (textová proměnná):

```
Dimension Txt as string
Txt = „Ahoj“
```

Ukázka v jazyce C:

```
Char *Txt;
Malloc(Txt,10);
Strcpy(Txt, „Ahoj“);
```

Porovnání

```
if  ( Hodnota == 0)
{
        // příkazy pokud podmínka platí
        Vysledek = TRUE;
}
Else
{
        // příkazy pokud podmínka neplatí
        Vysledek = FALSE;
}
```

Cyklus

```
Cyklus = TRUE;
while (Cyklus == TRUE)
{
   // příkazy
   if (Podmínka == TRUE)
   {
       Cyklus = FALSE;
   }
}
```

V příkladu výše je uveden tzv. zdánlivě nekonečný cyklus. Cyklus probíhá, dokud platí podmínka (porovnání ve „while“, dokud má proměnná „cyklus“ hodnotu true, pokud se vyhodnotí podmínka v if jako pravda, nastaví se proměnná cyklus na hodnotu „false“ a tím je cyklus ukončen. Zdánlivě nekonečný je z toho důvodu, že pokud by podmínka v konstrukci if nebyla nikdy splněna, cyklus by běžel do nekonečna).

Jiným typem cyklu je **for** cyklus, skládá se z: inicializátor, podmínka, přírůstek, tělo cyklu.

```
For ($i=1, $<10; $i++)
{
        // proveď příkaz(y) = tělo cyklu
}
```

1.2.5 Rekurze

Rekurze je opakované, vnořené volání stejné funkce.
Typy:

a) Přímá – podprogram volá sám sebe
b) Nepřímá – vzájemné volání podprogramů vytvoří kruh

Příklad
Výpočet faktoriálu N! = N * (N – 1)!
Př. 5! = 5*4*3*2*1 = 120

```
Sub Faktorial(N)
(
        if N = 0 then
                Vysledek = 1
        else
                Vysledek = N * Faktorial(N – 1)
        endif
)
```

1.2.6 Frameworky

SW poskytující prostředí a základní funkcionalitu. Příklad: NET, COCOA (framewrok pro počítače Apple), Oracle Application Development Framework, Eclipse, PHP Zend, Nette, …

1.2.7 CASE nástroje

CASE = Computer Aided Software Engineering

- Nástroj k vývoji, modernizaci a údržbě SW
- Kombinace SW nástrojů a strukturovaných metodologií

Účel: modelování, generování zdrojových kódů, reverse engineering (zpětné inženýrství – získání zdrojového kódu z aplikace)

Komponenty CASE nástrojů:

- Grafické ovládací prostředí
- Repository – centrální databáze objektů, proměnných…
- Podpora normalizace dat
- Verifikace konzistentní dat
- Nástroj pro návrh
- Textový editor pro popis objektů
- Generátor zdrojových kódů
- Import/export dat

Typy CASE nástrojů:

- Upper CASE – globální analýza, plánování
- Middle CASE – detailní analýza, návrh IS
- Lower CASE – fyzická (programová) realizace

I-CASE (integrovaný CASE) – podporuje všechny životní cykly projektu – včetně generování SW, tvorbu a údržbu dokumentace atd.

Přínosy použití CASE

- Eliminace neproduktivního času projektantů
- Zvýšení kvality SW
- Urychlení procesu vývoje
- Jednodušší provádění změn

1.3 KONTROLNÍ OTÁZKY

1. Co je to algoritmus?
2. Jaký je rozdíl mezi příkazem programovacího jazyka a instrukcí?
3. Jaký je rozdíl mezi vyšším a nižším programovacím jazykem?
4. Co je to kompilátor a linker?
5. Jaký je rozdíl mezi kompilací a interpretací programu?
6. Jaké znáte programovací jazyky?
7. Co je to assembler?
8. Co je to proměnná?
9. Co je to CASE nástroj?