Programování

Algoritmus, program, proces a procesor

Slovo **algoritmus** je mnohem starší než moderní počítač. Podle intuitivního významu se jím označuje **jednoznačně popsaný postup při řešení určité třídy úloh.**

Algoritmus má tři základní vlastnosti:

- **hromadnost** - není určen pro řešení jedné konkrétní úlohy, ale celé třídy úloh,
- **jednoznačnost** - při každém opakování postupu podle algoritmu se stejnými vstupními daty se získávají stejné výsledky,
- **konečnost** - algoritmus vede k řešení po konečném počtu kroků.

Program je zápis algoritmu v programovacím jazyce, je to **posloupnost instrukcí (příkazů)** konkrétního programovacího jazyka.

Algoritmus i program znamenají statický zápis určité úlohy. **Proces** je dynamický, je to **realizace algoritmu (programu).**

Slovo **procesor** má dva významy:

1) **Realizátor procesu,** tedy to, co realizuje proces - např. robot, počítač, člověk s kalkulačkou apod. *(širší význam)*
2) Základ počítače - **řídicí a aritmetickologická jednotka počítače.** *(užší význam)*

Program

Programování je činnost při tvorbě programů. **Program** je z hlediska:

- **uživatele** – spustitelný soubor, který je někde nainstalován a lze ho opětovně spouštět k provedení nějaké úlohy,
- **programátora** – textový soubor (zpravidla) se zápisem kódu, který popisuje určitý algoritmus tak, aby počítač jej mohl přeložit do spustitelného souboru a pak opětovně spouštět.

Program zahrnuje a potřebuje:

- data
- operace
- vstupní a výstupní operace.

Strukturované programování

Profesor Dijkstra ve své koncepci **strukturovaného programování** rozlišuje:

- posloupnosti instrukcí
- cykly
- větvení
- moduly.

Dva druhy překladačů

- **interpret** *(interpreter).* Pro spuštění interpretovaného zdrojového kódu musí být v pamětí počítače trvale přítomen interpret, což je jakési prostředí nutné pro běh interpretovaného programu. Interpret opakovaně překládá jednotlivé části zdrojového kódu a ihned je provádí. Výhodou je pružnost změny zdrojového kódu – odpadá fáze překladu, ale interpretovaný program běží mnohem pomaleji než přeložený program.
- **překladač** či **kompilátor** *(compiler).* Překladač pracuje se zdrojovým kódem jako s daty jakéhokoli jiného programu. Překládá zdrojový kód ve vyšším programovacím jazyce do strojového jazyka počítače. Postará se o vygenerování spustitelného kódu. Spustitelný kód pak lze opakovaně spouštět bez přítomnosti vývojového prostředí pro tvorbu zdrojového kódu a překladače. Při jakékoliv změně zdrojového kódu je však nutné jej opětovně překládat a generovat spustitelný kód. Výhodou je větší rychlost přeloženého programu než programu interpretovaného.

Rozdíly mezi interpretem a kompilátorem se stírají, protože některé překladače potřebují interpret, aby mohly spustit program (Java) a některé interprety nepřekládají do výsledného strojového kódu, ale do vlastního mezikódu, které teprve pak provádějí (Perl, Python).

Rozdíl (z hlediska programátora) také částečně stírají moderní vývojová prostředí (např. Turbo). Vývojová prostředí jsou programy, pomocí nichž se programátorův aplikační program píše, překládá, spouští a také ladí. U těchto vývojových prostředí se programátorovi jeví styl práce s kompilátorem stejně uživatelsky přátelský jako s interpretem.

Obecná struktura programu

- **Zavaděč** – kompilátor vytvoří krátký kód, který se spustí před aplikačním programem a ten umožní běh programu v daném prostředí, tento kód se označuje jako zavaděč. U interpretu se o tuto činnost stará sám interpret.
- **Datové definice** – jazyky se zde značně liší. Některé mají podrobné datové definice (zejména kompilované jazyky), některé definují proměnnou jejím prvním použitím (zejména interpretované).
- **Příkazy programovacího jazyky** – provádějí vlastní manipulaci s daty.

Dva druhy programů

1. **Dávkové programy** – se většinou spouštějí na příkazové řádce a jsou interaktivní, to znamená, že komunikují s uživatelem. Zahrnují:
   - čtení vstupních dat
   - zpracování dat
   - výstup dat
2. **Událostmi řízení programy** (*event driven programs*) – např. řídicí aplikace:
   - inicializace interních dat
   - čekání na příchod události
   - identifikace události a odpovídající reakce na ni.

Chyby v programu

- **syntaktické chyby** – jsou chyby vzniklé porušením pravidel zápisu kódu. Odhalí je překladač či interpret,
- **běhové chyby** *(run-time errors)* zvané též **výjimky** (*exceptions*) se projeví až při běhu programu (např. pokus o dělení nulou).
- **logické chyby** neboli chyby v logickém návrhu. Zjednodušeně řečeno „program nedělá to, co dělat má“. Tyto chyby se odhalují nejhůře.

Činnost při odstraňování chyb se nazývá **ladění** a připomíná detektivní pátrání či vědecký výzkum. Využívá se testovacích dat, bodů zastavení, při kterém se vypisuje obsah některých proměnných a hledá se zdroj chybných mezivýsledků apod.

Programovací jazyky

Programovací jazyky lze rozdělit:

Podle závislosti na typu počítače na:

- **Strojově orientované** - patří sem strojový kód a z něj vycházející jazyky symbolických instrukcí. Jejich výhodou je (při dostatečné erudici programátora) optimální využití možností prostředků počítače a vysoká rychlost běhu programu. Nevýhodou je velká složitost programování a nemožnost přenosu programu na jiný typ počítače.
- **Vyšší programovací jazyky,** které jsou orientovány ne na počítač, ale na určitý typ úlohy. Program vyšším jazyku se sestává nikoli z jednotlivých instrukcí počítače, nýbrž z příkazů, které jsou však počítači nesrozumitelné, takže před spuštěním musí být program převeden speciálním programem do strojového kódu počítače. Výhodou je snazší programování a možnost přenášení programu na jiný typ počítače. U moderních jazyků a počítačů nemusí být rychlost zpracování programu napsaného ve vyššímu jazyce významně pomalejší než u programu napsaném ve strojově orientovaném jazyce.

Podle charakteru jazyka lze programovací jazyky dělit na:

- **Procedurální** - jejich základem jsou procedury popisují algoritmy.
- **Neprocedurální** - jazyky logického programování a umělé inteligence.

Podle způsobu implementace lze rozdělit programovací jazyky na:

- **kompilační**
- **interpretační.**

Hlavní programovací jazyky

Jazyk	Vznik	Charakter a aplikační oblast
FORTRAN	1957	Kompilační, procedurální jazyk pro vědeckotechnické výpočty
Algol 60	1960	Kompilační, procedurální jazyk pro vědeckotechnické výpočty
COBOL	1960	Kompilační, procedurální jazyk pro hromadné zpracování dat (ekonomické aplikace)
LISP	1962	Neprocedurální jazyk užívaný v oblasti umělé inteligence, ovlivnil vývoj dalších jazyků např. Prologu
BASIC	1985	Převážně interpretační, jednoduchý procedurální jazyk pro vědeckotechnické výpočty, rozšířený především na mikropočítačích
Pascal	1971	Kompilační, procedurální jazyk, obecně použitelný, ovlivnil vývoj dalších jazyků, vhodný pro výuku programování, první jazyk podporující strukturované programování
C	1974	Kompilační, procedurální jazyk vhodný pro systémové programování, protože má prostředky pro optimální využití prostředí konkrétního počítače, vyvinul se z něj jazyk C++

Vývojové etapy programování

Za skoro 60 let svého vývoje prodělalo programování řadu etap. V celém vývoji programování se střetávají *„opravdoví programátoři“,* kteří usilují o tvorbu programů, které maximálně využívají prostředků počítače s praktiky (zvanými svými kritiky *„pojídači koláčů*“), kteří přistupují k tvorbě programů tak, aby výsledné programy vznikly pokud možno rychle, byly maximálně univerzální a přenositelné, aby mohly programovat týmy programátorů, které budou pracovat maximálně efektivně.

Předpokladem zvládnutí dalších etap ve vývoji programování je dokonalé zvládnutí většiny dosavadních technologií.

Ve vývoji programování přeskočíme první etapu – **programování ve strojovém kódu** a následnou – **programování v jazyce symbolických adres a instrukcí** (zvanou též programování **v jazyce assembleru**), protože šlo buď o úplně první programátorské pokusy či stále jde o programování speciálních (především časově kritických) aplikací. Ale i v programování blízkém technickým prostředkům počítače lze uplatnit některé z dále popisovaných postupů.

Vyšší programovací jazyky

První zásadní změnou v technologii programování byl vznik programovacích jazyků, které umožnily programátorům odhlížet od konkrétních technických prostředků daného počítače a soustředit se především na aplikační oblast příslušného programu. Se vznikem vyšších programovacích jazyků vznikly jazyky pro různé okruhy aplikací. Vyšší programovací jazyky také umožnily využívat počítače k tvorbě programů i neprogramátorům – specialistům na určitý obor.

Modulární programování

S rostoucí složitostí programů začala klesat efektivita programátorské práce. Začalo se ukazovat že efektivní programy netvoří geniální jedinci, kteří vytvářejí sami celé aplikace, ale týmy, které program rozdělí na menší části – **moduly,** přičemž přesně definují **rozhraní pro vzájemnou komunikaci** modulů.

Modul je tedy část programu, která popisuje přesně definovanou akci a má zcela jasně popsané rozhraní – tedy vstup a výstup daného modulu. Spolupracující týmy potřebují přesně znát, co modul dělá, s kterými daty pracuje, které výsledky poskytuje, ale konkrétní vnitřní algoritmy znát nepotřebují.

Strukturované programování

Strukturované programování definovalo několik základních pravidel, které musí programátor dodržovat, aby jeho programy byly správně strukturovány. Jeho idea vyvolala hodně diskusí, které lze charakterizovat známým členěním programátorů na *„opravdové programátory“* a *„pojídače koláčů“.* Hodně diskusí se vedlo kolem pokusu dokázat, že lze při dodržování těchto pravidel napsal skutečně všechny programy.

Zapomínalo se na to, že na počátku všeho stála zásada: „Piš program tak, aby jej po tobě mohl kdokoli (včetně tebe po letech) přečíst a pochopit.“ a že přísné dodržování zásad strukturovaného programování skutečně výrazně zvyšovalo produktivitu programátorské práce.

Objektově orientované programování

Myšlenka **objektově orientovaného programování (OOP)** byla sice už začátkem 90. let minulého století známá dvacet let, ale prakticky se začala prosazovat až s nástupem jazyka C++ a objektových verzí jazyka Turbo Pascal.

Co přináší OOP:

- zvýšení hladiny abstrakce, v níž uvažujeme o zadaném problému
- zvýšení stability knihoven, které důsledným **zapouzdřením** komunikují s okolím prostřednictvím bezpečného rozhraní
- zvýšení pružnosti vyvinutého kódu prostřednictvím **dědičnosti a polymorfismu**
- zvýšení stability celého řešení a rozdělení celého problému do tříd, ve svém důsledku výrazně snižuje výskyt chyb

Hlavní myšlenkou OOP je: „Řešení nemá být znovu objeveno, ale znovu použito.“

Nové jazyky a platformy

Dalším významným mezníkem byl vznik a masové rozšíření jazyku **Java**. Java je vhodná pro výuku programování, je zdarma, objektová orientace je v ní dobře implementovaná a Java se rozšířila prakticky ve všech platformách.

Vznikly dvě koncepce:

1. firmy **Sun** – platforma **Java 2 Enterprise Edition (J2EE)**
2. firmy **Microsoft** – platforma **.Net**

Co je oběma koncepcím společné:

- **Jazyk** – obě koncepce stojí na důsledně objektově orientovaných jazycích, které zavádějí speciální typ třídy označované jako rozhraní, mají automatickou správu paměti (*garbage collector*) a zavádějí zpracování výjimek a paralelních procesů přímo v definici jazyka. V koncepci J2EE je to **Java**, v koncepci .Net jazyk **C#** a připraveno je dalších celkem asi 20 jazyků.
- **Virtuální stroj** – zdrojový kód se nepřekládá do strojového kódu, ale do **mezikódu,** který po spuštění programu je předán speciálnímu programu označovanému jako virtuální stroj, který mezikód interpretuje. Určitá nevýhoda menší rychlosti je zde vyvážená tím, že program pracuje na všech platformách, pro které je implementován virtuální stroj.
- Silný důraz na implementaci prostředků umožňujících **vývoj webových služeb**.

Komponenty

Jazyk **Visual Basic** přinesl **komponentové programování. Komponenty** jsou dotažené moduly, jsou to objekty, které jsou schopné bezproblémového použití v různých aplikacích. Na počátku to byly hlavně prvky grafického návrhu (tlačítko, zaškrtávací políčko apod.), dnes jsou to komponenty, z nichž jsou skládány celé systémy.

XML

XML se stává univerzálním jazykem pro různé platformy. V jazyku XML se ukládají různé instalační a konfigurační parametry aplikací. Hlavní výhodou je nejen univerzálnost z hlediska počítače, ale přímá čitelnost XML dokumentu člověkem.

UML

Je grafický jazyk pro zobrazování všech etap vývoje programu – od návrhu, přes vývoj a následné nasazení.

Literatura:

[1] Pecinovský, R.: Quo vadis, programování, Computerworld 10/2003

[2] Rubeš, J.: Nebojte se programovat, Computer Media, Bedihošť 2001