Jiří PEŠKA, prom.mat.
Závod výpočetní techniky OKD, Ostrava

MODULÁRNÍ PROGRAMOVÁNÍ

Modulární programování

1.0 Úvod

Modulární programování je jeden z pojmů, které se v poslední době stále častěji vyskytují v různých článcích, v odborné literatuře a v diskusích mezi programátory. Je to dáno tím, že celý programátorský svět usilovně hledá jak pokročit vpřed a jak vyřešit problémy, které před nás překotný rozvoj počítačů staví. V zásadě jde o tyto okruhy otázek:

1) jak vyhovět stále se měnícím a rostoucím požadavkům uživatelů.
2) jak využít nových možností, které nám současné typy počítačů nabízejí.
3) jak si usnadnit a zlepšit práci.

Modulární programování je schopno nám s těmito problémy pomoci. Modulární programování se systematicky zabývá metodikou organizování programu, což je věc důležitá především z toho důvodů, že málo programátorů dokáže popsat své myšlenkové pochody při tvorbě programu.

Pokud si programátor počíná při organizování programu nesystematicky, dopustí se zcela určitě závažných chyb. Když se mu pak po těžkém boji podaří chyby odstranit, stane se z programu obskurní propletenec instrukcí, v němž se částečně dokáže orientovat počítač, ale žádná lidská bytost (autora programu nevyjímaje). Jak potom do takového programu dělat změny, které přicházejí s neúprosnou pravidelností?

Jestliže si je programátor schopen vytvořit svůj vlastní systém, pak to s největší pravděpodobností nebude systém nejoptimálnější. Aplikační programátor navíc ani nemá čas, aby mimo svou vlastní práci ještě vytvářel logicky a terminologicky ucelenou teorii programování. Bez dostatečné terminologické zásoby, není schopen sdělit své zkušenosti jiným. Pak se každý nový programátor naučí instrukce, ale neví, co s nimi. Neví jak si komplikovaný program rozčlenit na logicky jednodušší celky, jak tyto vázat mezi sebou.

V poslední době právě problémy vazeb mezi programovými prvky nabývají stále větší důležitosti. Neměl by už být problém vytvořit správně fungující rutinu. Ale je umění vytvořit program, který by fungoval obecně v různých programových systémech, který se nerozloží při nesprávně zadaných vstupech, v němž jde bez větších potíží přidat novou funkci, vyřadit nebo pozměnit starou funkci.

Problémy, které jsem zde stručně popsal, pocítili nejdříve tvůrci software. Byli nuceni řešit otázky organizování velkých programových celků (operační systémy, balíky programů), otázky standardizace, zastupitelnosti programátorů, údržby a rozvoje systémů. Proto jsou nejdále. Navíc mají možnost přímo ovlivňovat vývoj počítačů a hlediska programových prostředků a tak určovat tendence a směry programován v příštích letech. Proto se od nich musíme učit a musíme se snažit využívat vhodným způsobem prostředky, které jsou již vytvořeny.

1.1. Základní pojmy

Pro zlepšení práce s dalším textem uvedu přehled některých pojmů tak, jak se vyskytují v literatuře k počítačům IBM 360 a IBM 370. Tyto pojmy by měly mít své opodstatnění i v terminologii pro jednotnou řadu počítačů EJAD.

Je mutno si uvědomit, že přitom jde často o pojmy relativní, že záleží na stupni rozlišení který používáme.

Některé termíny nepřekládám a ponechávám je v angličtině.

Modul

Obecný termín, který může znamenat rozlišitelnou a identifikovatelnou jednotku pro takové účely jako jsou kompilace, ukládání a linkage editor. Něco jiného to znamená pro některé jazyky, popřípadě pro hardware.

Programový modul může být:

source modul (zdrojový modul) - napsaný programátorem nebo převedený do formy, kterou může zpracovávat počítač

object modul (přeložený modul) - výstup z kompilátoru

load modul (ukládaný modul) - výstup z linkage editoru.

V tomto článku budeme za modul nebo rutinu uvažovat i takové části programu, které považuje za jednotku programátor nebo analytik.

Program

Výstup z linkage editoru. Může být uložen do počítače, který podle něj řídí svou činnost.

To nemusí být možné pro load modul. Některé load moduly mohou být zpracovány pouze ve spojení s nějakým programem, nikoli samostatně.

Linkage editor

Program pro úpravu vazeb. Vytváří load moduly tak, že transformuje object moduly do formátu, který je vhodný pro natažení. Tento program kombinuje oddělené vytvořené object moduly a dříve spracované load moduly do jednoduchých load modulů, řeší křížové odkazy a symboly mezi nimi, automaticky na požádaní nahrazuje, vymazává a dodává řídicí sekce a pro moduly, které to požadují obstarává náležitosti pro overlay.

Zdroj

Vybavení výpočetního systému nebo operačního syatému požadované džobem nebo úlohou (task), počitaje v to vnitřní paměť, vstupní/výstupní zařízení, centrální jednotku, soubory, řídicí a pracovní programy a moduly.

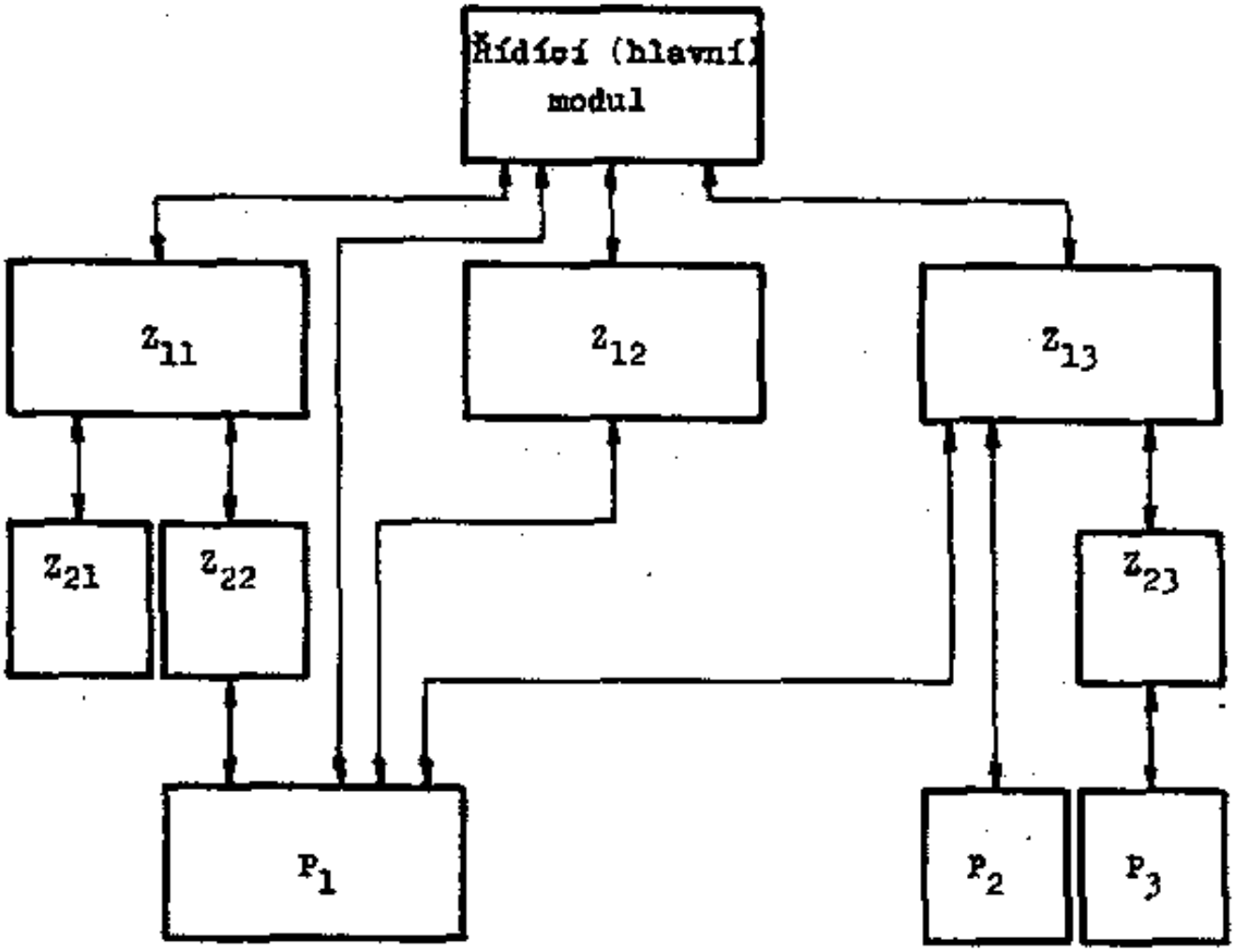
2.0 Modulární programování

Při tradičním neboli monolytickém programování se začne programovat podle specifikace od začátku a postupně se programuje instrukce po instrukci, dokud není program dokončen. Pak se ladí. Pokud se v různých programech vyskytují podobné části, jsou často znovu programovány a laděny. Tento postup vyhovuje u menších počítačů starších typů, na kterých se neřeší rozsáhlé úlohy.

Pro řešení rozsáhlých úloh a pro využití vlastností moderních počítačů je vhodné používat modulární programování.

Modulární programování se používá k rozložení řešení problému na logické části tak, že každá rutina může být programována, kompilována a laděna nezávisle. To umožňuje, aby byly složité problémy rozladěny do mnoha jednoduchých sekcí.

Prvním krokem při tvorbě modulárního programu je vypracování základního vývojového diagramu (obr. 1).



Z_{nm} m-tý zpracovatelský modul n-té úrovně

P_i moduly mnohonásobného použití
(např. modul pro třídění ve vnitřní paměti)

Obr1. Příklad základního schematu modulárního programování.
(Modul vyšší úrovně předává řízení modulu nižší úrovně a ten mu řízení po skončení práce vrací.)

O přidělování práce jednotlivým modulům rozhoduje řídicí rutina. Jestliže je z nějakých důvodů některá zpracovatelská rutina změněna nebo vyřazena, nedotkne se to ostatních rutin. Může to však znamenat změnu řídicí rutiny.

Modulární programování má tři hlavní kriteria:
- srozumitelnost
- snadné provádění změn
- standardní struktura programu

Toho lze dosáhnout:
1. Modulárním vývojovým diagramem, který dává přehledný obraz většiny složek a vazeb mezi nimi. Programový vývojový diagram je pak zpracován na požadované úrovni podrobnosti. Zde v maximální možné míře aplikujeme normalizované programování.
2. Podrobným popisem každé rutiny, který zachycuje účel rutiny, popis dat zpracovávaných rutinou a vysvětluje každý krok programové logiky, jak je znázorněn v modulárním vývojovém diagramu.
3. Programovými konvencemi. Užívání standardních konvencí pro strukturu programu, pro názvy proměnných, konstant, návěští a užívání standardní dokumentace umožňuje i osobám neobeznámeným s programem rozumět obsahu programu.

2.1 Navrhování modulů

Vývoj modulárního programu zahrnuje následující kroky:
1. Úvahu o začlenění programu v systému programů.
2. Zajistit modularitu vstupních a výstupních dat, tzn. vhodnou strukturu souborů, jednotné členění vět v souborech a jejich popisy.
3. Určení parametrů programu.
4. Seznam požadavků, částí a funkcí programu tak, jak člověka napadají, bez ohledu na jejich logické vazby. Zjistit, zda pro některé části nebo funkce již nejsou příslušné moduly naprogramovány. V opačném případě uvážit, zda požadované

části nemohou být použity v jiných programech. Jestliže ano, je třeba takový modul zařadit do knihovny obecných modulů.

5. Určit logické pořadí zpracovatelských rutin a návrh řídicí rutiny. Řídicí rutina musí být konstruována tak, aby velké objemy dat byly zpracovány nejrychlejší cestou.
6. Nakreslit přehledný vývojový diagram. Je třeba mu věnovat značnou péči, protože zde je možnost objevit většinu logických chyb.
7. Určit které části programu budou samostatné moduly a které budou pevně začleněny v programu. Uvážit otázku programovacích jazyků, ve kterých budou jednotlivé moduly programovány.
8. Nakreslit vývojový diagram modulů, ze kterých se program skládá.

2.2 Konvence pro řídicí rutinu

Řídicí rutina provádí všechna rozhodnutí ovládající tok dat pro vlastní zpracovatelské rutiny. Žádná zpracovatelská rutina nesmůže přímo předat data jiné zpracovatelské rutině na téže úrovni.

Do dalších úrovní zpracování se může vstupovat ze zpracovatelských rutin. Vstupní a výstupní funkce, které jsou společné více rutinám, jsou řízeny řídicí rutinou. Všechny společné oblasti jsou definovány jako část hlavního programu.

2.3 Konvence pro zpracovatelské rutiny

Zpracovatelské rutiny se vytvářejí pro každou logickou část programu.

Rutinu programujeme jako samostatný modul v následujících případech:

- je-li rutina natolik obecná, aby mohla být využita v jiných programech

- vyžaduje-li to zvláštní charakter zpracovávané úlohy, např. jde o softwarový program, jehož struktura má jiný charakter než struktura programu pro zpracování hromadných dat.
- jde-li o zpracování úlohy natolik rozsáhlé, že je třeba ladit jednotlivé moduly postupně.
- vymyká-li se rutina svým charakterem ze zpracovávané úlohy, např. budou se v ní velmi často provádět změny nebo jevli vhodné ji programovat v jiném jazyku než zbytek programu.

V opačném případě necháme rutinu začleněnu v programu. Je-li správně udělána, mělo by být její případné osamostatnění pouze formální záležitostí.

Každý modul je kompletní, má své vlastní oblasti, které jsou definovány pro výlučné používání tímto modulem. Rozhodnutí učiněné vně rutiny může mít vliv na to, zda bude rutina vyvolána nebo na to, jaké parametry jí budou předány, ale nemůže ovlivnit zpracování uvnitř rutiny. Rozhodnutí uvnitř rutiny nemohou zase ovlivňovat zpracování mimo rutinu.

Zpracovatelské rutiny mohou předávat řízení rutinám s mnohonásobným použitím. Po skončení vyvolávané rutiny je řízení vraceno volající rutině.

Vstupní a výstupní funkce, které slouží jen pro zpracovatelskou rutinu, mohou v ní být zpracovány.

Každá rutina obsahuje svou pomocnou část (nulování, nastavování výhybek atd.).

Každý výsledek zpracování, který je předáván zpět volajícímu modulu, by měl být uložen buď do oblasti volajícího modulu nebo do registrů.

3.0 Tři stupně modularity v Závodě výpočetní techniky OKR

I když řadu zásad modulárního programování jsme používali již dříve, rozhodli jsme se používat modulárního programování jako systém teprve v roce 1974. Vzhledem k vlastnostem a možnostem operačního systému počítače IBM 370/145, který je v našem závodě instalován a vzhledem k tomu, že počítáme s tím, že se bude modulární programování prosazovat postupně a nikoli skokem, rozvrhli jsme si modularitu do 3 stupňů:

1. Modularita zdrojových programů.
2. Modularita na úrovni linkage editoru.
3. Dynamická modularita.

3.1 Modularita zdrojových programů

Tento stupeň je technicky i organisačně pro aplikační programátory nejméně náročný. Proto předpokládáme, že se zpočátku bude nejvíce využívat možností tohoto stupně.
Tento stupeň předpokládá:
- v maximální možné míře využívat již dříve napsaných programů a jejich částí
- využívání a tvorbu generátorů programů a parametrických programů.

Druhý bod se vymyká předmětu tohoto článku.

Prvního je možno dosáhnout využíváním možností, které poskytují kompilátory jazyků COBOL a PL1.

Hlavní důraz však budeme klást na vytváření zdrojových programů pomocí programu ZG30, který umožňuje libovolně kombinovat vlastní kódování programátora s částmi různých programů uložených v knihovně zdrojových programů. Výstup z tohoto programu tvoří nový zdrojový program, který je uložen v knihovně zdrojových programů.

Předpokládáme, že pomocí tohoto programu dosáhneme úspory především při popisech dat, které jsou např. v jazyku COBOL velmi rozsáhlé, a které by měly být nyní přebírány do všech programů ze vzorového zdrojového modulu.

Někdy se při zpracování hromadných dat vyskytují série podobných programů. I zde je možnost, aby zkušený programátor vytvořil vzorový program, který pak budou upravovat a modifikovat méně zkušení a kvalifikovaní programátoři nebo programátoři v zácviku, kteří potřebují teprve získat základní programátorské návyky.

Tento stupeň modularity:
- ušetří mnoho mechanické práce
- zlepší úroveň standardizace.

Na druhé straně však klade zvýšené nároky na pozornost programátora, protože se lehce dají do programu převzít věci, které tam nepatří.

3.2 Modularita na úrovni linkage editoru

Spočívá v tom, že object moduly nebo load moduly jsou v linkage editoru spojovány do jednoduchého modulu, který je jako celek zpracovávan v počítači.

To již představuje skutečně stavebnicovou výstavbu programu se všemi výhodami modulárního programování.

Hlavní výhodou je možnost vytvářet obecné moduly a dále to, že se v případě změny opravuje a znovu kompiluje pouze jeden modul. Programy, které tento modul používají, projdou pouze fází linkage editoru.

Je zde však nutnost vytvářet a udržovat knihovny object a load modulů.

Zvýšení nároků na programátory je přitom minimální, protože tento způsob práce je pro počítače 3. generace charakteristický. Při "totálním" modulárním programování leží těžiště práce ve 2. a 3. typu modularity. Programy jsou často rozdělovány do modulů, ať to stojí, co chce. Je nutno uvážit, že přehnané vytváření object a load modulů klade zvýšené nároky na budování pomocného aparátu pro ladění, který má simulovat práci dosud neexistujících modulů. Tak může zbytečně růst strojový čas potřebný na odladění.

V některých časopisech se již na toto téma rozpoutaly diskuse dokládající testy a porovnáváním času jisté nevýhody modulárního programování. Domnívám se, že při rozumném členění programů se lze těmto nevýhodám vyhnout.

3.3 Dynamická modularita

Je to nejvyšší typ modularity.

Programy mohou během svého chodu vyvolat jiný program.

Programy mohou mít tyto struktury:

1. Jednoduchou strukturu

Jeden load modul obsahuje vše, nepředává řízení žádnému jménu modulu. Do paměti je ukládán celý.

2. Overlay struktura

V pamětí zůstává kořenový (root) segment, který vyvolává další segmenty, jež se postupně překrývají. Tato struktura ztratila význam u počítačů s virtuální pamětí, kde velikosti programů již nemají charakter limitujícího činitele. Protože náš počítač je virtuální pamětí vybaven, nezabýváme se touto strukturou.

3. Dynamická struktura

Během úlohy je vyvolán více než jeden load modul. Zde funguje řídící program jako zprostředkovatel. Zpracování modulů může být sériové nebo paralelní (paralelní neužíváme). Každý

z load modulů může mít jeden ze 3 strukturních typů.

Tato struktura je výhodná u složitějších programů, zvláště když logické cesty v programu závisí na zpracovávaných datech. Load moduly jsou přineseny do paměti v okamžiku, kdy jsou požadovány a v příhodném okamžiku mohou být z paměti vymazány.

U programu této struktury je třeba uvažovat možnosti nového použití programu.
Programy mohou být:

A. not reusable - opakovaně nepoužitelný.

Programy této kategorie jsou natahovány přímo z knihovny, ze ktré jsou požadovány. Tyto programy se během svého plnění mění a při opětovném vstupu do změněného programu bychom nedostali správný výsledek. Program je proto nutno znovu vyvolávat z knihovny.

B. Serially reusable - opakovaně použitelné

Takovéto moduly jsou vybudovány tak, aby všechny části, které se během chodu mění, byly před novým použitím uvedeny do původního stavu. Proto může být program tohoto typu, když je jednou uložen do paměti, prováděn opakovaně. Modul může být v každém okamžiku užíván pouze jednou úlohou. Další úloha může začít, až skončí s používáním předchozí.

Naší snahou je, aby většina programů byla tohoto typu, i když nebudou dynamicky vyvolávány.

C. Reenterable

Modul je vytvořen tak, aby se neměnil během svého plnění. Takovéto moduly jsou po natažení uloženy do zvláštní oblasti, v níž jsou chráněny před náhodnými modifikacemi z uživatelova programu. Jsou uloženy pouze jednou a mohou být užívány libovolnou úlohou v libovolnou dobu.

Vytvoření modulu tohoto typu vyžaduje větší programátorskou zkušenost a nepatří k běžnému repertoáru programátora.

Moduly tohoto druhu máme v úmyslu dělat pouze v odůvodněných případech.

S dynamickou modularitou zatím počítáme pro softwarové programy psané v jazyku assembler, v němž je dynamická modularita dobře zpracována. V problémově orientovaných jazycích jsou jisté technické potíže.

4.0 Zavádění modulárního programování

Jednotlivé prvky modulárního programování jsou známé a používané. Jako systém bude však muset modulární programování překonat řadu potíží.

V první řadě bude třeba ovlivnit úvahy při návrhu programu. Na program je třeba pohlížet jako na prvek celého systému programů. Jen tak je možno zobecňovat. Tyto změny myšlení musí postihnout analytiky ve stejné míře jako programátory.

Rozčlenání specifikace programu se stane jednou z nejdůležitějších činností. Bude důležité, aby někteří analytici a vedoucích programovacích týmů věnovali větší část svého pracovního času projekci programů.

Dalším důležitým činitelem bude, že programátoři budou muset častěji dávat svou práci z rukou. Modul, který programátor vyrobí, bude muset dát v plen obecnému použití. Takovýto modul musí dokázat obejít se v drsném světě bez neustálé patronace a záštity svého tvůrce. Programátoři jej budou využívat a pak budou mít tendenci považovat tento cizorodý prvek ve svém programu za zdroj většiny potíží, které se v programu vyskytnou.

Aby modul s úspěchem obstál, musí se dokázat sám obhájit. Proto musí, pokud je ve zdrojové formě, mít vlastnosti, které byly uvedany v kapitole 2.0 a v load formě musí mít:

- odolnost vůči parametrům a vatupním datům.
- účinnou a standardní diagnostiku.

Modul, který pod vlivem špatně zadaných parametrů začne provádět nekontrolované činnosti je špatný modul. Odolnost můžeme zajistit dvěma způsoby:

1. Striktně odmítat vše, co se vymyká z limitu; trvat na vyplňování všech hodnot parametrů. Při nesplnění požadavků přerušit zpracování s jasným udáním důvodů.
2. Snažit se zachránit, co se zachránit dá. Při neudání nebo zadání špatných hodnot dosadit předpokládané hodnoty.

Standardní diagnostika musí programátoru umožnit, aby si mohl zkontrolovat, zda modul úspěšně proběhl, popřípadě, aby mohl zjistit důvod abnormálního konce. Chce to budovat diagnostické oblasti a vypisovat zprávy, z jejichž obsahu to bude jasné. Není to všechno lehké, ale vyplatí se to.

Na závěr je možno říci, že budeme modulárně programovat, jestliže si svou práci vždycky důkladně rozmyslíme před tím, než s ní vůbec začneme.

Literatura:

1. Management Planning Guide for a Manual of Data Processing Standards
   GS20-1670

2. IBM System/360 Operating System
   Supervisor Services and Macro Instructions
   GC28-6646

3. IBM System/360 Operating System
   Concepts and Facilities
   GC28-6535

4. IBM OS
   Linkage Editor and Loader
   GC28-6538