Ing. Ján Duplinský
VVS Bratislava

# ŠTRUKTÚRA PROGRAMOV (v BPS)

## 1. Úvod

Návrh štruktúry programu tvorí tú etapu procesu tvorby programu, v ktorej sa na základe presného popisu požiadaviek na program kladených vytvoria presné programové špec.ikácie. Pod programovými špecifikáciami rozumieme rozdelenie budúceho programu na samostatné celky - moduly, popis ich činnosti a vzájomnej komunikácie. Po etape návrhu štruktúry programu nasleduje kódovanie a overovanie /ladenie a testovanie/ programu.

Tieto dve etapy sa vykonávali a ešte stále vykonávajú súčasne, pričom najpoužívanejším formálnym nástrojom bol programovací jazyk, v ktorom sa kódovanie prevádzalo. Terminológiu a notáciu pre návrh štruktúry programu si vytváral každý sám, najčastejšie to boli rôzne grafy, ktoré si programátori kreslili na kus papiera, ktorý im potom slúžil ako jediný podklad pri konštrukcii a overovaní programu. Pri kódovaní a overovaní programu často dochádzalo k modifikáciam, ktoré spôsobili rôzne nekonzistentnosti medzi tým, čo bolo na papieri a v programe. A tak sa stalo, že po čase kus popísaného papiera sa stal nepoužiteľný a všetky rozhodnutia sa prevádzali na texte programu, ktorý vôbec explicitne nezobrazoval štruktúru programu.

V tomto článku sa pokúsime vytvoriť určitú terminológiu, pomocou ktorej v ďalšom prevedieme analýzu štruktúry programu

a procesu jej tvorby. Budeme sa snažiť čo možno najpresnejšie definovať pojmy, ako sú modul programu, segment programu, program, štruktúra programu, hierarchia modulov, segmentov a štruktúr programu a taktiež čo rozumieme pod procesom návrhu /štruktúry/ programu.

## 2. Základné pojmy

Vstupnými informáciami pri návrhu štruktúry programu /v ďalšom iba návrh programu/ sú funkčné a prevádzkové požiadavky, ktoré sa od budúceho programu budú vyžadovať. Výstupom z procesu návrhu programu by malo byť rozdelenie budúceho programu na samostatné celky, ktoré v ďalšom budeme nazývať modulmi alebo segmentami, presný popis toho čo sa od nich vyžaduje, ako i presný popis ich vzájomnej komunikácie. Základnou požiadavkou, ktorú na metódu návrhu programu kladieme, je aby:

1. výsledný program bol spoľahlivý, zrozumiteľný, modifikovateľný a efektívny
2. proces návrhu programu pozostával z viacerých intelektuálne zvládnuteľných a kontrolovateľných krokov [1].

V ďalšom budeme vychádzať z programu-konkrétneho objektu, v ktorý návrh programu v konečnom dôsledku vyúsťuje, a ktorého /všeobecne/ vlastnosti dostatočne dobre poznáme.

### 2.1 Modul

Modulom programu budeme nazývať jednotku programu, ktorá pozostáva z dátových štruktúr, ktoré sú jej lokálne a z operácii /procedúr a funkcií/ nad objektami dátových štruktúr modulu. Prístup k jednotlivým objektom dátových štruktúr modulu je len cez operácie modulu. Nie všetky operácie modulu musia byť prístupné zvonku. Pod špecifikáciou modulu rozumieme presný popis činnosti objektov /operácií/, ktoré sú prístupné zvonku /ktoré

vyváža-exportuje/ a presný popis ako ich správne používať.

```
modul meno;
lokálne dáta
inicializačné príkazy + telo modulu.
operácia O1(...); ..... end O1;
operácia O2, ..... end O2;
    :                    :
operácia On; ..... end On;
          modul programu.
```

Modul, ktorý realizuje abstraktný dátový typ [2,3], ako je napr. tabuľka, strom, graf, zásobník, frotna a pod., budeme nazávať dátovým modulom. Moduly pozostávajúce iba z /navzájom závislých/ operácií a prakticky zo žiadných dátových štruktúr nazývame riadiacimi modulmi. Riadiaci modul buď neobsahuje žiadne dátové štruktúry, alebo ak nejaké predsa len má, tak sú vo väčšine prípadov jednoduché premenné, ktoré sa používajú na riadenie vetviacich príkazov a cyklov v operáciách. Ak nejaký modul vyváža /sprístupňuje okoliu/ niektoré zo svojich operácií, iný modul ich zasa môže používať. Nech m a n sú moduly programu, potom ak modul m používa objekty modulu n hovoríme, že modul m používa modul n alebo tiež modul m má spojenie s modulom n. Spojenie modulu m s modulom n budeme označovať s(m,n) a je to množina špecifikácií tých objektov modulu n, ktoré modul m používa.

Pre špecifikáciu dátových modulov sa hodia definičné, resp. algebraické špecifikácie, zatiaľ čo pre riadiace moduly procedurálne špecifikácie [2.4].

Modul je abstraktnou jednotkou programu, nakoľko abstrahuje od implementačných detailov, ktoré sú pre užívateľa modulu nepodstatné, ba naopak môžu byť škodlivé, ak sa ich pokúša v programe nevhodne využívať. Modul grupuje v sebe tie dáta a operácie, ktoré spolu súvisia.

## 2.2 Segment

Segment programu pozostáva z jedného alebo viacerých modulov programu. Jeho úlohou je grupovať v sebe tie moduly programu, ktoré spolu nejakým spôsobom súvisia /ktoré sa podieľajú na riešení toho istého problému alebo podproblému/.

Segment programu pozostáva z jedného alebo viacerých modulov programu, pre ktoré platí, že:

1. existuje medzi nimi jeden modul, ktorý má zvláštne postavenie v rámci segmentu a budeme ho nazývať hlavným modulom segmentu.
2. Pre každý modul segmentu okrem /hlavného/ musí existovať aspoň jeden iný modul segmentu, ktorý ho používa. Do každého modulu segmentu, okrem hlavného modulu segmentu vedie cesta /výpočet/ z hlavného modulu segmentu. To znamená, že segment neobsahuje taký modul /okrem hlavného/, ktorý by nebol v segmente použitý.
3. Žiadny modul segmentu okrem hlavného nesmie byť používaný modulmi mimo daného segmentu.

Každý segment je navonok jednoznačne charakterizovaný hlavným modulom segmentu. Keď hovoríme, že segment m používa segment n, znamená to, že hociktorý modul segmentu m môže používať /vyvážané/ objekty segmentu n. Spojenie segmentu m so segmentom n, s(m,n) pozostáva zo špecifikácií tých objektov /hlavného modulu/ segmentu n, ktoré segment m používa.

Hlavný modul segmentu môže vyvážať viac operácií, ktoré sprostredkujú "vstup" do segmentu.

Segment má veľmi dobré vlastnosti, ktoré sa pri tvorbe, analýze a vôbec pri manipulácii s programom využívajú. Je to koncepčná /problémovo orientovaná/ a manipulačná jednotka programu, ktorá sa, ako v ďalšom uvidíme dá využiť na rôznych abstraktných úrovniach programu a v každom kroku procesu tvorby programu.

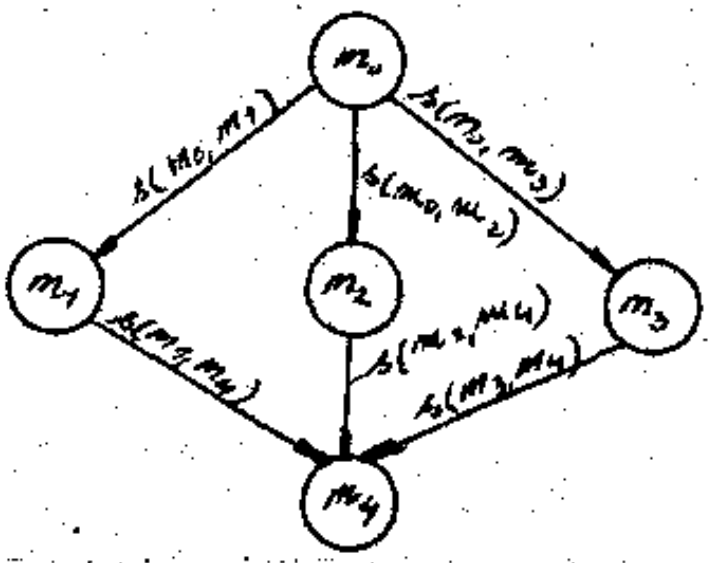
## 2.3 Program a štruktúra programu

Program pozostáva z jedného alebo viacerých segmentov, pre ktoré platí,

1. existuje medzi nimi jeden tzv. hlavný segment programu, ktorým začína výpočet programu
2. pre každý segment programu, okrem hlavného segmentu programu, vedie cesta /výpočet/ z hlavného segmentu programu.

Štruktúrou programu P rozumieme súvislý orientovaný graf s označenými hranami, v ktorom

1. vrcholmi sú segmenty /moduly/ programu P. Z vrcholu m vedie orientová hrana do vrcholu n práve vtedy, ak segment m používa segment n. Táto hrana bude označená spojením s(m,n).
2. Počiatočným vrcholom grafu je hlavný modul programu, ktorým výpočet začína, a ktorý dominuje nad ostatnými vrcholmi grafu.



Obr. Štruktúra programu /segmentu/

Z daného programu P je možné zostrojiť niekoľko štruktúr zodpovedajúcich jednotlivým abstraktným úrovniam programu. Nech $S_i$ je štruktúra programu P, v ktorej niektoré vrcholy reprezentujú segmenty programu, potom ak v štruktúre $S_i$ nahradíme niektorý /alebo viacero/ segment jeho štruktúrou dostaneme štruktúru, ktorú označme ako $S_j$. Štruktúra $S_j$ je podrobnejšia, detailnejšia ako štruktúra $S_i$ a hovoríme, že $S_i$ je na vyššej abstraktnej úrovni ako $S_j$. Štruktúra programu, v ktorej každému vrcholu odpovedá /jeden/ modul programu nazývame základnou štruktúrou programu.

Zo základnej štruktúry programu sa dá vytvoriť hierar-

chická postupnosť štruktúr programu $S_0\ S_1\ \ldots\ S_m$ (1) taká, že

1. štruktúra $S_m$ je základná štruktúra programu
2. štruktúra $S_0$ pozostáva iba z jediného vrcholu /hlavného/ modulu programu reprezentujúceho celý program /počiatočná štruktúra/
3. štruktúra $S_i$ je na vyššej abstraktnej úrovni, ako štruktúra $S_j$ pre každé $i < j$, $1 \leq i$, $j \leq n$.

Segmenty programu môžeme čiastočne usporiadať vzhľadom na reláciu spojenia. Ak zo segmentu $s_i$ vedie cesta do segmentu $s_j$, potom hovoríme, že segment $s_i$ je na vyššej abstraktnej úrovni ako segment $s_j$.

Postupnosti segmentov $s_0, s_1\ \ldots\ s_m$ hovoríme, že je usporiadaná zhora-nadol /zdola-nahor/, ak pre každé $s_i$, $s_j$, pre $1 \leq i \leq m$ platí, že ak $s_i$ je na vyššej abstraktnej úrovni, ako $s_j$, potom $s_i$ je horeuvedenej /nižšej/ postupnosti pred /za/ $s_j$.

Je rozumné predpokladať, že štruktúra veľkého programu sa nevytvára na jedenkrát, ale postupne od jednoduchej k zložitejšej a končiac základnou.

Pod procesom tvorby štruktúry programu P rozumieme proces tvorby hierarchickej postupnosti štruktúr $S_0, S_1\ \ldots\ S_m$, kde

1. $S_0$ je štruktúra programu pozostávajúca z jedného vrcholu reprezentujúceho program P.
2. Štruktúru $S_i$ získame zo štruktúry $S_{j-1}$ $(0 < i \leq m)$ tým, že v nej za vrchol reprezentujúci segment nahradíme /substituujeme/ jeho štruktúru. Prechod od štruktúry $S_{i-1}$ k štruktúre $S_i$, nazývame jedným krokom procesu tvorby štruktúry programu. V každom kroku vytvoríme /zjemníme/ štruktúru iba jedného segmentu.
3. Proces končí základnou štruktúrou $S_m$ /ďalej už nemáme čo rozvíjať, nakoľko všetky vrcholy reprezentujú konkrétne moduly v budúcom programe/.

## 2.4 Acyklickosť štruktúry programu

Tam, kde nie je možné vytvoriť presné procedúry, metódy

postupu, ukazuje sa, že sú účinné i vhodné obmedzenia, ktoré nemusia byť vždy a za každých okolností konštruktívne. Ako príklad nám v tomto smere môže poslúžiť zákaz alebo obmedzenie použitia príkazu go to v programovaní, i keď sú známe prípady, kde jeho použitie vedie k elegantnému a efektívnemu riešeniu niektorých problémov [5].

Z čisto teoretického hľadiska, zložitosť rôznych operácií nad acyklickým grafom je v porovnaní s cyklickým grafom aspoň o rád nižšia.

Z hľadiska analýzy, modifikácie, ladenia a testovania programu a vôbec z hľadiska rôznych manipulácii s programom /spojovanie segmentov v jeden celok, vytváranie hierarchickej dokumentácie programu, kontrola na konzistentnosť jednotlivých častí programu a pod [6.7] / je acyklický graf lepší nez cyklický.

Od štruktúry programu budeme požadovať, aby bola acyklická /t.j. aby neobsahovala cyklus/.

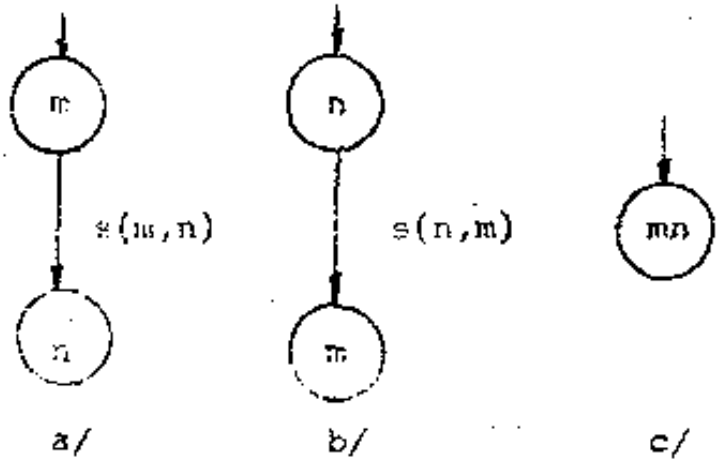
Z hľadiska procesu tvorby štruktúry programu, acyklickosť vyžaduje od každého kroku dôslednejšie premyslenie každého rozhodnutia, hlbší pohľad do nasledujúcich krokov a dodržanie rozhodnutí vytvorených v predchádzajúcich krokoch. Po skoro dvojročnom používaní systému BPS, ktorého cieľom je podporovať tu prezentovanú metódu, môžeme zodpovedne prehlásiť, že acyklickosť štruktúry programu kladie na proces jej tvorby rovnaké nároky, ako zákaz používania príkazu go to pri programovaní a rovnaký je i dôsledok na kvalitu programu.

V ďalšom sa pokúsime analyzovať dôsledky tohto obmedzenia na proces tvorby štruktúry programu. Prvým problémom, na ktorý pri návrhu programu pravdepodobne najskôr tvorca narazí je nasledovný: segment m používa segment n a ten zasa chce použiť segment m, t.j.

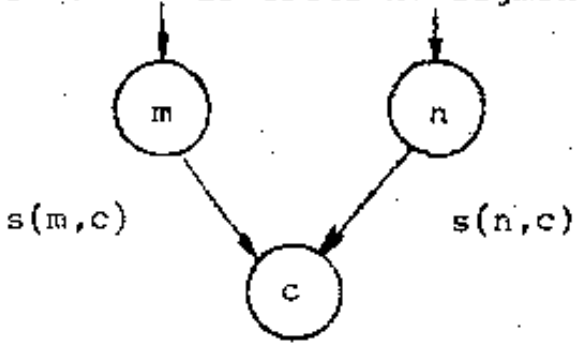

Cyklus môže samozrejme vzniknúť i cez niekoľko segmentov, my však pre jednoduchosť ďalšieho výkladu budeme uvažovať

priamy cyklus znázornený na predchádzajúcom obrázku. Riešení je niekoľko, a treba sa rozhodnúť pre to, ktoré v danej situácii najlepšie vyhovuje.

1. Prvé riešenie spočíva v tom, že objekty, ktoré cyklus spôsobujú presunieme buď do segmentu m alebo segmentu n, čím sa jeden segment stane nadradený druhému alebo opačne /prípady a,b/. Niekedy segmenty m a n sú tak vzájomne závislé, že je výhodné ich spojiť v jeden segment /prípad c/. Týmto spôsobom sa napr. musí riešiť problém navzájom rekurzívnych operácií. Modul, ktorý takto vznikne /označme ho mn/ môže byť pomerne veľký, čo z metodologického hľadiska nie je najlepšie. V takom prípade v module ponecháme iba navzájom rekurzívne operácie, zatiaľ čo pre ostatné vytvoríme ďalší /v prípade potreby viac/ modul, ktorý bude podriadený modulu mn. Vytvoríme teda nový segment mn.




2. Druhé riešenie spočíva v tom, že objekty, ktoré cyklus spôsobujú sústredíme do tretieho segmentu c.



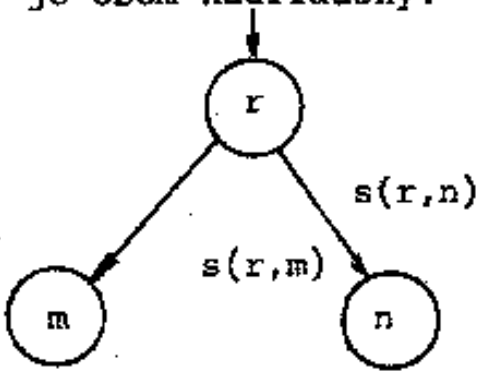

Toto riešenie je najčastejšie používané. Segment c buď obsahuje tie objekty, cez ktoré segmenty m a n navzájom komunikujú, alebo objekty, ktoré sú spoločné obom segmentom.

V ďalšom budeme segment tohoto druhu nazávať komunikačným alebo tiež spoločným. Komunikačný segment sprostredkujúci komunikáciu medzi segmentami, je vo väčšine prípadov realizáciou abstraktného dátového typu. Najčastejšie realizuje tabuľku, strom, graf, zásobník a pod. Segment tohoto druhu nazývame dátovým segmentom. Komunikačné segmenty, ktoré nie sú dátovými, ale pre svoju univerzálnosť v danom okolí ich používa viac segmentov, budeme nazývať univerzálnymi.

Pre komunikačné segmenty je charakteristické to, že majú viacerých užívateľov. I keď na druhej strane dátový segment môže mať iba jedného užívateľa.

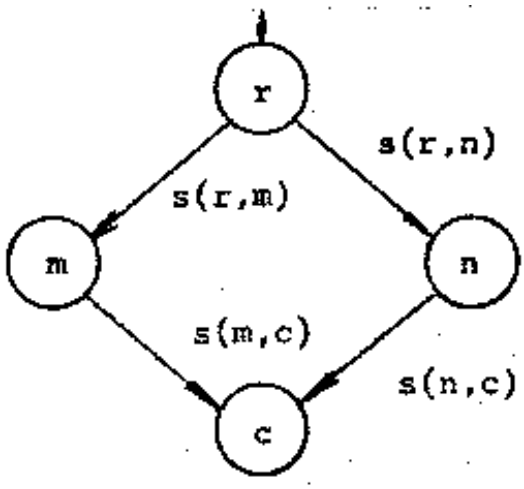
Ďalšie riešenia ani tak nesúvisia s riešením nastoleného problému, ale kvôli úplnosti ich uvedieme.

3. Tretie riešenie je podobné predchádzajúcemu v tom, že sprostredkuje komunikáciu medzi dvoma segmentami cez tretí segment, ktorý je obom nadriadený.




Segment r má prístup cez operácie segmentu m (n) k jeho dátam, ktoré zasa potom môže cez operácie odovzdať segmentu n(m) a opačne. Je to skutočne veľmi zriedkavý spôsob komunikácie, i keď nie je vylúčený. Segment r tu slúži ako kontrolor, ktorý celú komunikáciu medzi segmentami m a n riadi a kontroluje. Pre jeho riadiacu funkciu ho v ďalšom budeme nazývať riadiacím segmentom. Vo väčšine prípadov /zo skúsenosti z tvorby systému BPS/ riadiaci segment pozostáva z jediného modulu.

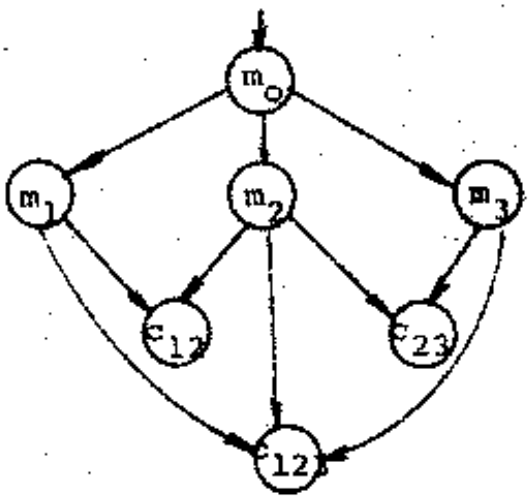
4. Posledné riešenie je kombináciou druhého a tretieho riešenia




## 2.5 Analýza procesu tvorby štruktúry programu

Návrh štruktúry programu prebieha vo viacerých krokoch, pričom každý krok pozostáva z návrhu štruktúry jedného segmentu. Tým sa splní jedna z požiadaviek na metódu návrhu programu kladených t.j. rozdelenie celého procesu návrhu na viac krokov. Ďalšia požiadavka súvisí s intelektuálnou zvládnuteľnosťou jednotlivých krokov a ich kontrolovateľnosti vzhľadom na prijaté obmedzenia. V ďalšom sa budeme venovať práve tomuto problému.

Vytvoríme si model - schému štruktúry segmentu, ktorá by mala byť akýmsi prototypom štruktúry, s ktorou sa v každom kroku návrhu štruktúry programu stretávame. K tomuto modelu sme dospeli skutočne analýzou programu systému BPS, ktorý sme vytvárali popísaným spôsobom bez toho, aby sme si to explicitne uvedomovali.




Obr. 1. Štruktúra segmentu

Model štruktúry segmentu na obr. 1 je intelektuálne zvládnuteľný a odpovedá aplikácii princípu "rozdeľuj a panuj" a princípu abstrakcie.

V každom kroku návrhu štruktúry programu riešime určitý problém alebo podproblém v rámci nejakého väčšieho problému. Ak je riešený problém zložitý, komplikovaný postupujeme tak, že sa ho snažíme rozdeliť na niekoľko menších, jednoduchších podproblémov, ktoré síce môžu byť navzájom závislé, ale pomocou nich už by sme vedeli vyriešiť daný problém. Riešenie jednotlivých podproblémov nás na tejto úrovni /v tomto kroku/ nezaujíma, skrátka abstrahujeme od nich.

Štruktúra segmentu pozostáva z riadiacej časti $m_0$, funkčných častí $m_1$, $m_2$, $m_3$ a komunikačných častí $c_{12}$, $c_{23}$, $c_{123}$. Funkčných a komunikačných častí môže byť menej alebo viac, ich počet závisí od riešeného problému a jeho rozdelenia na jednotlivé podproblémy.

Úlohou riadiacej časti segmentu je riešiť určitý problém, k tomu využíva služieb funkčných častí segmentu, ktoré sa podieľajú na riešení jednotlivých podproblémov v rámci daného problému. Riadiacu časť vo väčšine prípadov tvorí hlavný modul segmentu a v ojedinelých prípadoch veľmi jednoduchý segment. V ďalšom pre zjednodušenie výkladu budeme predpokladať, že riadiaci segment tvorí iba hlavný modul segmentu.

Funkčné časti segmentu nás momentálne nezaujímajú a zatiaľ nám postačí o nich vedieť iba to, že sa jedná o segmenty. V ďalšom ich budeme nazývať funkčnými segmentami.

Oveľa zaujímavejšie sú komunikačné časti segmentu. Každá komunikačná časť segmentu môže pozostávať z viacerých segmentov, ktoré môžu byť používané odpovedajúcimi fukčnými segmentami.

Segmenty zgrupované v jednej komunikačnej časti sú vo väčšine prípadov navzájom nezávislé komunikačné segmenty, ktoré majú spoločné iba to, že ich používajú tie isté funkčné segmenty. Ak sa riešený problém podarí rozdeliť na úplne nezávisle podproblémy, potom segment nemusí mať žiadne komunikačné časti. V takom prípade štruktúrou segmentu je strom. To sa ale samozrejme vždy nemusí podariť.

Komunikačné segmenty sú v programe veľmi dôležité, nakoľko

- majú vždy aspoň dvoch užívateľov a každý zmena v ich špecifikáciách môže postihnúť potenciálne všetkých jeho užívateľov
- vo väčšine prípadov realizujú abstraktný dátový typ, t.j. veľké dátové štruktúry a operácie nad nimi /báza dát, adresár, súbor, tabuľka a pod./, v ktorých prebieha väčšina výpočtu programu
- realizujú tú časť kódu programu, ktorá je spoločná viacerým častiam v programe, t.j. redukujú duplicitu
- po vytvorení ich špecifikácií - spojenia s funkčnými segmentami, na fukčných častiach segmentu môžeme pokračovať paralelne.

Z horeuvedených dôvodov je preto veľmi dôležité, aby spojenia s komunikačnými segmentami boli jednoduché, zrozumiteľné, pokiaľ je to možné nezávisle od reprezentácie modulu /to sa vyžaduje od každého spojenia/ a implementovane efektívne.

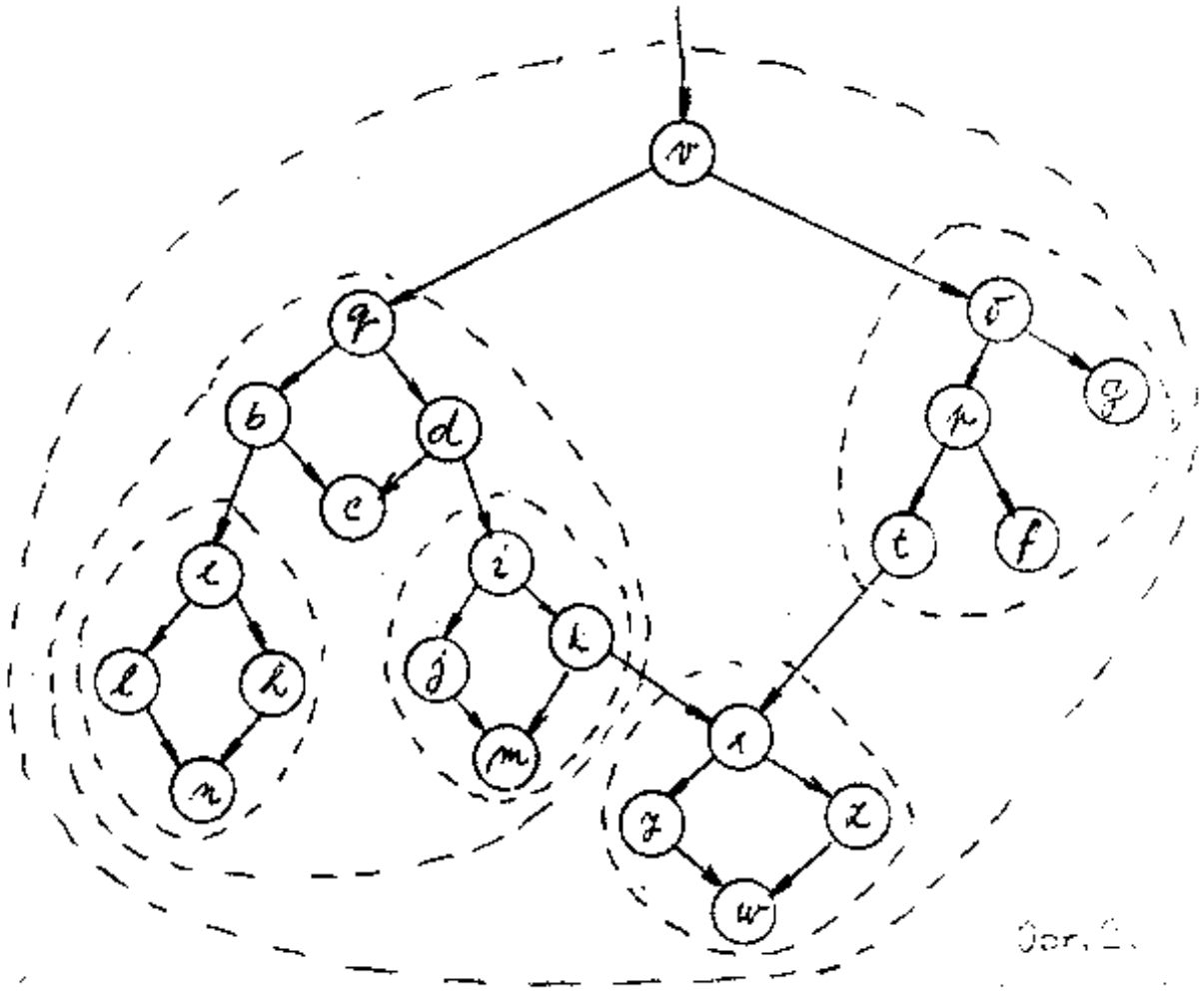
Z definície segmentu vyplýva, že hlavný /riadiaci/ modul segmentu dominuje nad modulmi segmentu. Znamená to, že všetky objekty segmentu, ktoré majú byť priamo prístupné zvonku musia byť sústredené v hlavnom module segmentu.

Spojenie funkčného segmentu s komunikačným v štruktúre segmentu umožňuje ľubovoľnému /na rôznej úrovni "zahniezdenia"/ modulu funkčného segmentu používať objekty komunikačného segmentu. To ma svoje výhody i nedostatky ako to v ďalšej časti uvidíme. Pre lepšie pochopenie uvedieme najskôr príklad /obr 2/. Segment v sa skladá z

- hlavného modulu v /riadiaca časť segmentu/
- funkčných segmentov q a o,
- komunikačného segmentu x.

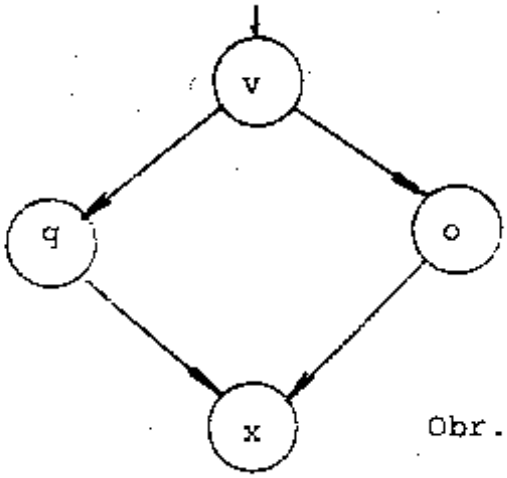
Funkčný segment q sa skladá z funkčných segmentov b a d a komunikačného segmentu c atď.

Nech štruktúra na obr. 2 je základnou štruktúrou programu P a nech štruktúra S´ na obr. 3 je štruktúrou programu P, ale na vyššej abstraktnej úrovni ako štruktúra S.




Obr. 2.

Zo štruktúry S´ vyplýva, že ľubovoľný modul segmentu q a o môže používať segment x. Zo základnej štruktúry S /obr. 2/ vidíme, že v skutočnosti segment x používajú iba segment /modul/ i a t.




Obr. 3 Štruktúra segmentu v S´

Komunikačný segment x v štruktúre segmentu je na rovnakej /abstraktnej/ úrovni, ako funkčné segmenty /q,o/ i napriek tomu, že ho používajú segmenty /moduly/, ktoré sú na nižšej

abstraktnej úrovni /k,t/. Táto skutočnosť vyplýva z jeho veľmi dôležitého poslania v rámci celého segmentu.

Komunikačné segmenty vykonávajú veľmi dôležité úlohy, najmä pri systémovom programovaní /dátabázové systémy, operačné systémy a pod./ a ich nesprávne použitie by sa dalo buď vedome alebo nevedome zneužiť. Ukazuje sa preto potreba obmedziť rozsah použiteľnosti komunikačných segmentov na podmnožinu funkčných segmentov. Napr. ukazuje sa rozumné rozsah použiteľnosti komunikačného segmentu x viazať na segment i a t alebo segment k a t. Tým, že rozsah použiteľnosti segmentu viažeme s menším okolím, zjednodušujeme tým analýzu jeho okolia, ktorá je pri modifikácii segmentu často potrebná. Ukazuje sa užitočné, aby tvorca segmentu mal možnosť explicitne určiť rozsah jeho použiteľnosti. To sa na úrovni segmentu /modulu/ realizuje tak, že v jeho hlavičke je uvedený zoznam segmentov, ktoré ho môžu používať.

V súčasnej dobe nie sme schopní podať presný recept, ktorý by z daných požiadaviek na program kladených dal jednoznačne vytvoriť štruktúru budúceho programu. Uviedli sme však niekoľko obmedzení, ktoré do istej miery ako vodidla pri "transformovaní" požiadaviek do funkcií operácií jednotlivých segmentov programu. Segment v tomto smere slúži ako abstraktná-problémovo orientovaná jednotka. Pri návrhu štruktúry programu /alebo kódovanie programu/ môžeme postupovať metódou zhora-nadol, počnúc segmentami na najvyššej abstraktnej úrovni a končiac segmentami - modulmi na najnižšej abstraktnej úrovni, alebo metódou zdola-nahor, pri ktorej je postup opačný. V oboch prípadoch celý proces návrhu programu sa skladá z viacerých krokov. Na to, aby základná /konečná/ štruktúra programu mala požadované vlastnosti, postačí, ak sa postaráme o to, aby v každom kroku vzniklá podštruktúra mala tieto vlastnosti.

LITERATÚRA


1. BOYD, D.L., PIZZARELLO, A.: Introduction to the WELLMADE Design Methodology. IEE Transaction on Software Engineering, Vol. SE-4, NO.4, July 1978

2. GUTTAG, J.V.: The specification and Application to Programming of Abstraft Data Types, TR-CSRG-59, TORONTO, 1975

3. ČERNÝ, J., GRUSKA, J., WIEDERMANN, J.: Typy a štruktúry dát. Zborník referátor "SOFSEM '78", VVS Bratislava, 1978

4. LISKOV, B.H., ZILLES, S.N.: Specification Techniques for Data Abstraction. IEEE Transactions on Software Engineering SE-1.1, pp 7-19, March 1975

5. KNUTH, D.E.: Structured Programming with goto Statements, Computing Surveys, Vol. 6, No.4, December 1974

6. FISCHER, P. a kol.: BPS, Výskumná práca č.144, VVS Bratislava, 1978

7. FISCHER, P. a kol.: BPS-Rozpracovanie prvej verzie. Výskumná práca, VVS Bratislava, 1978