Růžena B O N S A R D O V Á , Jan S O K O L

VÚNS Praha

Jazyk pro systémové programování SYSTRAN

Úvod

Systran je pokusem o vytvoření jazyka vyšší úrovně pro účely systémového programování. Původním záměrem bylo úplné nahražení ASSEMBLERu, který byl v době, kdy jsme se touto otázkou začali zabývat, hlavním programovacím prostředkem systémových programátorů. V průběhu práce na návrhu jazyka, jeho implementaci a prověřování, došlo k určitým změnám v koncepci i vlastním provedení. Ani druhá (v tomto referátě zmíněná) verze jazyka si nečiní nárok na definitivní řešení problematiky. Jedním ze základních kritérií projektu je flexibilita kompilátoru, umožňující snadnou modifikaci jazyka podle připomínek uživatelů.

V referátu se pokoušíme vymezit oblast systémového programování a stanovit charakteristiku dnešních postupů na tomto poli. Těžištěm referátu jsou kritéria, jimž jsme podřídili projekt, a jejich diskuze. Na tomto místě je uvedena stručná zmínka o dvou dalších programovacích jazycích této sorty (MARY, SUE).

První verze jazyka je popsána zkratkovitě, spoléháme se na ilustrativnost konkrétních příkladů (předpokládáme alespoň informativní znalost jazyka PL/I). Uvádíme několik stručných poznámek k implementaci první verze. V další části stručně charakterizujeme druhou verzi jazyka, jejíž oprávnění prokázal provozní výzkum první verze. V závěru jsou stručně shrnuty dosavadní zkušenosti a nastíněn další možný vývoj jazyka.

SYSTRAN je implementován pro československý počítač EC 1021 a v současné době se prověřuje.

Vymezení problematiky

Oblast systémového programování není doposud přesně vymezena. Můžeme ji charakterizovat výčtem konkrétních objektů, spadajících společně pod pojmem "programovací prostředek". Jde tedy například o operační systémy, překladače, interprety, obslužné programy, třídící generátory. Uvedené programy patří k základnímu softwaru počítače.

Dnešek je charakterizován prudkým rozmachem nasazování počítačů do nejrůznějších odvětví národního hospodářství a s tím spojeným vytvářením t. zv. uživatelského softwaru. Těžko lze nyní, na relativním začátku tohoto procesu, určit všechny faktory, jež budou v nejbližší době ovlivňovat jeho rozvoj. Na základě nedlouhé historie vývoje softwaru lze pouze odhadnout některé pravděpodobné tendence a stanovit obecné zásady, jež by tento rozvoj kladně ovlivnily.

Většina programů, patřících k základnímu softwaru, je charakterizována několika podstatnými rysy:

- rozsáhlostí programů
- velkými nároky na rychlost a zároveň úsporné využití paměti
- rozmanitostí problematiky, ve které převládá nenumerické zpracování informace
- velkými nároky na spolehlivost programů

Domníváme se, že v těchto rysech se uživatelský software nebude podstatně lišit od základního. Proto předpokládáme, že by jazyk SYSTRAN, který jsme navrhovali s ohledem na uvedené vlastnosti, mohl mít i širší platnost využití, přestože byl navrhován tak, aby co nejlépe vyhovoval konkrétním potřebám VÚMS.

Současné způsoby tvorby softwaru

Ačkoliv jsou činěny i pokusy o použití stávajících vyšších programovacích jazyků pro tvorbu softwaru, zůstává pro drtivou většinu programů této oblasti preferovaným programovacím prostředkem ASSEMBLER.

Charakteristickým znakem přístupu k tvorbě softwaru je

všeobecně rozšířený názor, že tato činnost je spíše uměním, než soustavnou inženýrskou prací. Na toto pole působnosti prozatím nepronikly žádné inženýrské prvky, činnost probíhá intuitivně, nejsou zavedeny standardní postupy pro volbu architektury větších programů. Neexistuje báze, na níž by mohla být komplexně budována prvková základna. Následkem je duplicitní tvorba programových celků, představujících značné objemy instrukcí.

Výroba softwaru, založená na takových postupech, je sumárně neefektivní bez ohledu na kvalitu jeho jednotlivých složek.

Zdá se, že některá kriteria tvorby softwaru jsou notoricky přeceňována a jiná nedoceňována. V prvním případě jde o efektivnost výsledných subproduktů, v druhém o systémový přístup k návrhu a implementaci projektu i jeho částí, flexibilitu a portabilitu subproduktů a přenositelnost (i nehotových) subproduktů mezi členy pracovního týmu.

Domníváme se, že jedním z podpůrných psychologických momentů této situace je právě výlučnost postavení ASSEMBLERu mezi programovacími prostředky pro výrobu softwaru. ASSEMBLER je soubor nejelementárnějších programovacích prvků, který dává uživateli značnou volnost volby způsobu řešení. V žádném směru neovlivňuje programátorské myšlení a cítění. Dává širokou možnost uplatnění intuice. Tím lze vysvětlit preferenci užívání ASSEMBLERu před jinými programovacími jazyky přes zjevnou manuální pracnost tvorby ASSEMBLERských programů.

Lze to považovat i za důvod odmítavého postoje k jakémukoli pokusu o standardizaci a typizaci. Systémoví programátoři, zvyklí na určitou "svobodu" řešení, se brání jak administrativním opatřením, která by mohla situaci částečně vyřešit, tak nasazení vyššího programovacího jazyka.

Přitom vhodně navržený programovací jazyk je jedním z nutných předpokladů progresivního rozvoje inženýrského přístupu k vytváření programovacích prostředků.

Kriteria projektu

K základním kritériím, jimž jsme hodlali projekt podřídit, jsme dospěli

1) zvážením výhod a nevýhod užití vyššího programovacího jazyka všeobecně
2) vytypováním datových struktur a programovacích postupů nejčastěji používaných při tvorbě softwaru
3) studiem dostupné literatury, zabývající se problematikou systémových jazyků
4) zvážením situace na poli systémového programování ve snaze odhadnout nutné ústupky kritériím, vyplývajícím z bodů 1), 2) a 3) ve prospěch naděje jazyka na přežití a další vývoj.

Poznámky k bodu 1).

Přednosti použití vyššího programovacího jazyka (ve srovnání s užitím ASSEMBLERu) při zápisu libovolné úlohy lze shrnout v několika následujících bodech:

1.1) Doba potřebná k napsání a odzkoušení programu je podstatně kratší.
1.2) Program je přehledný a tím i snadněji dokumentovatelný.
1.3) Doba, potřebná k uskutečnění změny (nebo i předělávky) je podstatně kratší.
1.4) Zásah do hotového (chodivého) programu nese menší nebezpečí zavlečených chyb.
1.5) Programovací jazyk se stává i dorozumívacím prostředkem mezi členy pracovního týmu. Tím je podstatně snížen čas, potřebný ke koordinaci vedoucím pracovníkem.
1.6) Nehotové části programu jsou snadněji přenositelné na jiného programátora.
1.7) Doba, potřebná k převedení programu na jiný počítač, je podstatně kratší.

Těmto nesporným výhodám se v případě systémových programů staví do protikladu dva závažné nedostatky:

1.I) Nižší efektivnost výsledného programu pokud jde o rychlost výpočtu i využití paměti.

1.II) Obtížnost orientace v přeloženém programu a odtud vyplývající stížení ladění programu na strojové úrovni.

Při hledání řešení je nezbytné mít na zřeteli další dva nesporné fakty:

- programovací jazyk je vždy pouze nástrojem. Jako takový pouze umožňuje - nikoliv zaručuje. Na jedné straně i dobře navržený jazyk vyšší úrovně nezabrání psaní špatných programů, na druhé straně samo použití ASSEMBLERu ještě nezaručuje efektivnost výsledku. U rozsáhlých programů se pak riziko zvyšuje dodatečnými úpravami, kterým se nelze nikdy vyhnout, a které se musí dělat ne s ohledem na efektivnost, ale na snižování rizika zavlečených chyb.
- Vhodnost či nevhodnost použití jazyka netkví často v jazyce samém, ale v jeho implementaci a nezřídka i dokumentaci. Proto při stanovení kritérií uvažujeme jazyk za nedílný souhrn tří podstatných složek:
  - DEFINICE
  - IMPLEMENTACE
  - DOKUMENTACE

Uvedeme tedy základní kriteria, vztahující se k bodu I):

- Při implementaci jazyka by měl být kladen důraz na optimalizaci.
- Jazyk by měl být překládán do ASSEMBLERu. Výsledný kód by měl být přehledný, respektující sémantiku uživatele a umožňující eventuální (nutnou) dodatečnou ruční optimalizaci.
- Jazyk by neměl obsahovat prvky, které nutně vedou k podstatné neefektivnosti kódu, již nelze ovlivnit optimalizací, nebo jejichž optimalisace vede ke ztrátě jeho přehlednosti.
- Jazyk by měl poskytnout uživateli prostředky pro přímé řízení efektivnosti kódu.
- Jazyk by měl umožňovat zápis ASSEMBLERských úseků.
- Dokumentace by měla umožnit učení se jazyku v několika úrovních obtížnosti a v případě potřeby poskytovat přesné informace o efektivnosti ASSEMBLERských ekvivalentů definovaných programovacích prvků.

- Kompilátor by měl být konstruován pro bohatou diagnostiku a umožnit překladový režim pro generování kódu, usnadňujícího hledání dynamických chyb.
- Z obecných výhod použití vyššího programovacího jazyka, jež jsou jmenovány na začátku tohoto odstavce, by měl být kladen hlavní důraz na rozvinutí vlastností, charakterizovaných bodem 1.5), 1.6) vzhledem k tomu, že systémové programy jsou většinou rozsáhlé a vyžadují si týmovou práci.

Jazyk uvedených vlastností by dal do ruky vedoucího pracovníka nástroj, usnadňující koordinaci práce na větším programu, umožňující nasazení slabších programátorů na ty části programu, pro něž hledisko efektivnosti nestojí na prvním místě, a zapojení začátečníků po poměrně krátké době zácviku.

Ve prospěch racionalizace přístupu k tvorbě větších programových celků by bylo i zavedení některých administrativních opatření, vztahujících se k použití jazyka. Jde na př. o:
- určení podmnožiny jazyka, vhodné k řešení dané části celku
- explicitní výčet těch částí celku, pro které je výslovně povolena ruční optimalizace a pro ty případy
- přesné stanovení pravidel pro komentování těchto částí

Poznámky k bodu 2)

Jak bylo řečeno, lze s jistotou očekávat prudký nárůst problematiky, vyžadující nové programovací prostředky.

2.1) Jazyk pro systémové programování by měl být proto pružný a poměrně málo specifikovaný

2.2) Měl by poskytovat vhodné prostředky pro archivaci hotových produktů

2.3) Jazyk by měl vhodně navazovat na ostatní prostředky automatizace přístupu k hotovým produktům nebo jejich částem

2.4) Při návrhu i implementaci jazyka by mělo být počítáno s jeho dalším vývojem.

Poznámky k bodu 3)

Autoři článků z oblasti dané problematiky se z velké části shodují v požadavcích na vlastnosti jazyků pro systémové programování.

Jde na př. o požadavek na efektivnost kódu, průhlednost jazyka, ovlivnitelnost kódu tvarem zdrojového textu, jednoduchost syntaktických i sémantických pravidel, "čitelnost", dokonalou diagnostiku.

Menší objem problematiky je zatím předmětem diskuse. Jde například o automatickou konverzi typů, zařazení modulů, zapsaných v jiných jazycích, míru závislosti jazyka na počítači.

Vzhledem k tomu, že většina úvah i navržených jazyků pochází z univerzit, je všeobecně kladen větší důraz na čistotu jazyka a méně brána v potaz dosavadní praxe. V našem záměru je daleko více zdůrazněn fakt týmové spolupráce.

Žádný z publikovaných systémových programů se prozatím všeobecně neprosadil.

Zmíníme se krátce o některých aspektech dvou jazyků (MARY autorů M. Rains, H. Conradiho a P. Holagera - Univerzita - Trondheim a SUE autorů B. L. Clarka a J. J. Horninga - Univerzita - Toronto). Oba jazyky jsou myšlenkově blízké jazyku PASCAL.

Na jazyku SUE nás zaujalo především celkové pojetí: autoři kritizují přístup k tvorbě operačního systému OS/360 a kladou si za cíl vytvoření jednoduchého flexibilního rozšiřitelného operačního systému pro řadu IBM. Jazyk, který navrhují, považují za programovací prostředek pro vytvoření tohoto jediného operačního systému. Za hodnotící kritérium kvality jazyka berou úspěch (resp. selhání) celého projektu.

Jazyk SUE si vypůjčuje dle potřeby vnější podobu i myšlenky z jazyka PASCAL a částečně i z XPL, LSD a BLISSu. Jednotlivé prvky jsou sestaveny do konečné podoby v souladu s množstvím kritérií a obohaceny o prvky, vynucené povahou konkrétní problematiky.

Uvedeme ta kritéria projektu SUE, která v našem původním záměru nebyla obsažena, nebo která jsme pod tlakem jiných požadavků byli nuceni opustit.

- Na první místo je kladen požadavek dobře strukturovaného programu (příkaz GOTO je nahrazen příkazem RETURN pro návrat z procedury a EXIT pro opuštění složeného příkazu nebo bloku).
- Do jazyka není zaveden žádný prvek, který by v sobě skrýval nebezpečí dynamické chyby, již nelze odhalit ladícím během (např. každý pointer musí mít přesně definován rozsah působnosti, nejsou definována dynamická pole).
- Je počítáno s tím, že jazyk je zcela soběstačný (nepočítá se se zařazením ASSEMBLERských úseků).
- Jazyk má umožnit kompilaci separátních částí hierarchicky uspořádaných (částí může být výkonný program nebo datová struktura) a to takovým způsobem, aby přeložený program nevyžadoval další zpracování LINK - editorem.

Také v jazyce MARY je patrná snaha umožnit strukturované programování. Možnosti cyklů a podmínek jsou natolik bohaté, že lze předpokládat užití příkazu GOTO (který je v MARY na rozdíl od SUE definován) jen ve zcela výjimečných případech.

V jazyce MARY je nestandardním způsobem řešeno přiřazení. Znak přiřazení (=:) je chápán na úrovni operátorů. Může se vyskytovat kdekoliv uvnitř výrazu. Výraz je pak chápán jako posloupnost příkazů (podobně jako v polské notaci), neuvažuje se precedence operátorů a výrazové závorky jsou nahrazeny příkazovými závorkami BEGIN a END. Tato filosofie činí bezprecedenční výrazy zcela přirozenými.
Na příklad výraz 2=:K+3=:K=:I; je co do účinku ekvivalentní posloupnosti ALGOLských příkazů K:=2; I:=(K+3)=:K; a zápisem Z + BEGIN INT(A); READ(A); A END=:V je proměnné V přiřazen součet hodnoty proměnné Z a hodnoty, přečtené z vnějšího média.

Stručná charakteristika první verse jazyka

Koncepce dat

V jazyce jsou definovány 4 typy položek - aritmetický (A), hexadecimální (X), znakový (C), binární (B). Položkou rozumíme konstantu nebo proměnnou. Aritmetické položky mohou nabývat hodnot celých čísel (může jim být přiřazena i adresa: I=A'ALFA+2'). Hodnotami hexadecimálních (resp. znakových) položek mohou být hexadecimální (resp. znakové) řetězce délky až 256 byte. Délka položky musí být vždy uvedena, pro A - položky je možno přidat požadavek na zarovnání (půlslovo, slovo). Binární položky nabývají Booleovských hodnot TRUE (1) nebo FALSE (0). Každá proměnná musí být explicitně deklarována, a to kdekoli v programu, překladač uložení deklarací v rámci programu respektuje. (DEF A:XL30, DEF M:AL2/2). Binární položky, jejichž deklarace následují bezprostředně po sobě, se skládají po osmi do jednoho bytu. Pro položky typu C a X je definován trojí formát - pevný, proměnný (délka položky je obsažena v jejím prvním bytu) a nedefinovaný (délka položky je zadána obsahem jiné proměnné).

Jednoduché položky lze skládat do větších celků - polí (jednodimenzionálních) a tabulek. Pole je soubor položek téhož typu a formátu. Tabulka je soubor stejně strukturovaných řádků. Každý řádek tabulky může obsahovat data různých typů i formátů. Tabulka může být definována jako virtuální. V tom případě je spojena s některým vnějším médiem, na němž se předpokládá celá tabulka. Ve vymezené části vnitřní paměti se pak zpracovává její aktuální část.

Existuje bohatý sortiment prostředků přístupu k datům. Lze se odvolat na konstantu, proměnnou (jednoduchou či indexovanou) i na části proměnné (jednotlivé byty či bity). Lze se odvolávat na objekty, jejichž adresa je obsahem položky (adresa druhého řádu I=⟨ALFA⟩). Lze definovat změnu interpretace dat pro jediný příkaz (X=A:XL3). S každým polem (resp. tabulkou) je spojen t. zv. hlavní pointer, jenž je při deklaraci inicializován adresou první položky pole (resp. prvního řádku tabulky). Jeho hodnota může být

měněna příkazy jazyka (RISE, POINT). Libovolné proměnné může být dočasně přiřazen doplňkový atribut POINTER TO (tím jsou definovány pomocné pointery, svázané s polem či tabulkou: DET ALFA POINTER TO TAB). K datům lze pohodlně a efektivně přistupovat prostřednictvím hlavního (a vedlejších) pointeru (TAB(,2) - odvolání na třetí položku řádku tabulky TAB, na níž ukazuje její hlavní pointer.) Kavio posuv hlavního pointeru po virtuální tabulce řídí fyzický vstup (resp. výstup) z (resp. na) příslušného média. Je možno se odvolat na délku řádku tabulky (L.TAB(,)), délku položky (L.A(3)), adresu položky (P.A(3)).

Pro usnadnění komunikace mezi jednotlivými částmi programu je v jazyce definován způsob přístupu ke společným datům (jakási obdoba DUMMY sekce z ASSEMBLERu a širším uplatněním) pomocí pseudoobjektů. (DEF(IN TAB X:LOCL)). Pseudoobjekty mohou napomáhat i ke zlepšení přehlednosti textu v rámci jednoho modulu.

Výrazy

Aritmetický výraz je definován bez závorek, vyhodnocuje se zleva doprava bez ohledu na precedenci operátorů. Relace je definována na objektech typu A nebo X. Pro logický výraz jsou definovány operátory negace (NOT), konjukce (AND), disjunkce (OR). Logický výraz je definován bez závorek, vyhodnocuje se zleva doprava bez ohledu na precedenci operátorů. (A+B-L.D, TAB(2,3) GT X).

Příkazy

Všude tam, kde to bylo smysluplné, jsme přebrali vnější podobu jazyka PL/I. V podstatě byly převzaty příkazy CALL, DO, END, GOTO, IF, prázdný, RETURN. U příkazu DO bylo klíčové slovo BY nahrazeno dvojicí UP-DOWN pro jasné rozlišení vzestupného a sestupného cyklu. Pro rozskok je použito FORTRANského příkazu skoku podle přiřazení. Volání procedury se děje příkazem CALL nebo zápisem funkce. Parametry se volají zásadně hodnotou, volání jménem lze zajistit přes pseudoobjekty.

Jsou definovány dva typy přiřazovacího příkazu. Při při-

řazení dle levé strany (značeno = nebo :=) je zajištěna automatická konverze typů. Při přiřazení dle pravé strany jde o pouhý přesun definovaného počtu bitů. Zřetězení umožňuje jasný a přehledný zápis požadavku na tvar výstupu (zřetězení je definováno jen pro znakové položky. Položky jiných typů jsou na typ C automaticky převáděny).
(A=B||C||C'KONEC'||X)

Následuje výčet speciálních SYSTRANských příkazů a jejich stručná charakteristika. Účinek všech těchto příkazů (s výjimkou CATAL) je statický. Příkazem DET lze proměnné dočasně přiřadit atribut POINTER TO nebo vyjádřit požadavek na to, aby hodnota proměnné či pointeru byla až do odvolání držena v registru. Příkaz REDET ruší účinek příkazu DET. Příkazy RISE a POINT řídí posuv pointerů a v případě virtuálních tabulek i (případný) vstup a výstup dat. (RISE M.TAB BY 2, POINT TAB (230)). Příkazem RELEAS dává programátor k dispozici kompilátoru jeden nebo více uživatelských registrů. Příkaz REQUIRE ruší účinek příkazu RELEAS. Příkazy CODE a NCODE jsou "kódové závorky", uzavírající ASSEMBLERský úsek programu. (Je možno zařazovat i jednotlivé ASSEMBLERské štítky, označené speciálním znakem v prvním sloupci). Příkaz PREFIX umožňuje modifikaci některých identifikátorů, které za ním následují (jeho účinek ruší příkaz NPREFIX). Příkazem CATAL můžeme zapisovat do knihovny konvencí informace o specifikacích formálních parametrů procedury nebo separátní bloky dat. Lze uvést i požadavek na katalogizaci poznámek k zapisovaným objektům (na př. stručný popis funkce procedury nebo informace o možné interpretaci zapisovaných dat).

Přístup ke katalogizovaným objektům se děje pomocí mechanizmu, manipulujícího s pseudoobjekty.

Poznámky k implementaci

Překladač první verze jazyka je tříprůchodový.
1. běh je syntakticky řízený kompilátor z jazyka SYSTRAN do mezijazyka INTERSYS1. Výsledkem je úplný adresář a řetěz

operací. Jazyk INTERSYS1 (při jehož návrhu jsme vycházeli z polské notace) je soubor vzájemně nezávislých operací, komunikujících přes společný stack.
2. běh sekvenčně zpracovává operace jazyka INTERSYS1. Výsledkem je posloupnost instrukcí mezijazyka INTERSYS2. Každá z těchto instrukcí má tvar volání makroinstrukce (můstku), kterému je jednoznačně přiřazena posloupnost strojových instrukcí.
Ve 3. běhu dochází k rozvinutí jednotlivých můstků pomocí makrogenerátoru. Výsledný program je v ASSEMBLERu. Pro zápis můstků jsme vytvořili vlastní jednoduchý makrojazyk. Různé překladové režimy se uskutečňují výměnou můstků pro 3. běh kompilátoru.

Druhá verze jazyka

Vycházíme z toho, že nutným předpokladem přežití jazyka a jeho dalšího rozvoje je prověření praxí. Je známým faktem, že kvalita jazyka není ještě zárukou jeho širokého používání. Zkušení programátoři se většinou neradi vzdávají zavedených myšlenkových postupů, založených na analogii doposud preferovaného programovacího prostředku. Je to konečně pochopitelné. Jejich hlavním zájmem je vyřešit problém. Programovací prostředek má k tomu napomáhat. Nový programovací jazyk nese v sobě nebezpečí ztráty efektivity jejich práce, protože vstupuje další faktor - nutnost učení se. Programátor tedy (není-li k tomu vnějšími okolnostmi donucen) sáhne po novém prostředku jedině v tom případě, připadá-li mu ve srovnání s jeho přínosem uvedený faktor zanedbatelný.

Tuto skutečnost lze kritisovat, ale nelze ji přehlížet. Ve prospěch věci je poskytnout zkušenému programátorovi to, co je ochoten přijmout, a začátečníky zacvičovat již s určitým uplatněním administrativních opatření. Takový přístup má velké výhody:

- jednak zamezí tomu, aby programovací prostředek a jeho uživatel stáli proti sobě (zkušený programátor rád použije nového prostředku, který navazuje na jeho znalosti a vžité

postupy, začátečník dá přednost vyššímu programovacímu jazyku před ASSEMBLERem).
- jednak lze očekávat cenné připomínky a rady od zkušených programátorů, užívajících jazyk byť jen v omezené míře.

Tento fakt jsme uvažovali již v původním záměru ne však v té míře, v jaké bylo třeba. Zdá se, že je prozatím nutno uvažovat o SYSTRANu nikoli jako o náhradě ASSEMBLERu, ale jako o jeho partnerovi. Velká část původních kritérií zůstává tímto pojetím nedotčena. Praktickým důsledkem je pak nutnost zavedení změn, týkajících se sjednocení ASSEMBLERských a SYSTRANských přístupů k některým aspektům programování.

Jde především o sjednocení formátu zdrojového textu (včetně odstranění středníku jako oddělovače jednotlivých příkazů SYSTRANu), zajištění kompability mezi ASSEMBLERským a SYSTRANským typy dat (na př. zavedeny typy P, H do SYSTRANu) a pozvednutí jednotlivých ASSEMBLERských příkazů na úroveň příkazů SYSTRANu. (Lze tedy např. napsat tuto posloupnost příkazů:

```
      IF A EQ 0 THEN
%     OC /,B
      F/SE
%     OC B,A                    ).
```

Příklady, uvedené na obrázku 1, mají výhradně ilustrativní charakter. Oba programy provádějí tutéž činnost za použití různých výrazových prostředků:

přečtou štítek, zjistí délku přečteného řetězce znaků (mezery uvnitř textu jsou považovány za významné) a pro každý řetězec vypíše na tiskárnu text

DELKA n-TEHO RETEZCE = d ,

kde n je pořadí štítku, d je programem zjištěná délka přečtenéhо řetězce.

První příklad je uveden v celém rozsahu, z druhého jen ty jeho části, kterými je třeba nahradit text v prvním příkladu. Text, který má být nahražen, je vyznačen svislou čarou na levém okraji.

```
    PROGRAM LSTRING1                        ;STATICKY ZACATEK PROGRAMU
    DEF TISK:50(CL40)SYSLST                 ;DEFINICE VSTUPNIHO SOUBORU
    DEF SNIM:1(CL80)SYSIPT-KON              ;DEFINICE VYSTUPNIHO SOUBORU
    DEF POR:H
    DEF LENGH:H
    DEF (IN SNIM) STITEK:80C                ;DEFINICE PSEUDOOBJEKTU
    DEF I:H
ZAC POR =1
    POINT TISK(0)                           ;INICIALIZACE HL. POINTERU
A   RISE M.SNIM                             ;POSUV POINTERU-VSTUP STITKU
    LENGH=0
    DO I=79 DOWNTO 0                        ;SESTUPNY CYKLUS
    IF STITEK(I) EQ C'' THEN
    LENGH=LENGH+1
    ELSE
    GOTO B
    END
B   TISK(,0)=CL6'DELKA' || POR ||C'-TEHO ' ||                              C
          CL10'RETEZCE =' || 80-LENGH       ;ZRETEZENI A PRIRAZENI VYSLEDKU RADKU
M                                            TABULKY TISK, NA NIZ UKAZUJE JEJI
M                                            HLAVNI POINTER
    RISE M.TISK                             ;ZVYSENI POINTERU VYST.BUF.
    POR=POR+1
    GOTO A
```

Obrázek 1/1 strana

```
KON EXIT                                    ;DYNAMICKY KONEC PROGRAMU
    END LSTRING1,ZAC                        ;STATICKY KONEC PROGRAMU

                                . . .

    DEF PSNIM:F

                                . . .

    PSNIM=ADDR(ARSNIM+79)                   ;PROM.PSNIM PRIRAZENA ADRESA
    DO WHILE   <PSNIM>:X EQ X' 40'          ;PODMINKA JE SPLNENA,KDYZ
           AND LENGH LT 80                  ;JEDEN BYTE,JEHOZ ADRESA
                                            ;JE OBSAHEM PROMENNE PSNIM,
                                            ;MA OBSAH 40 HEXADECIMALNE
                                            ;A PLATI LENGH <80
    LENGH=LENGH+1
    PSNIM=PSNIM-1
    END

                                . . .
```

Obrázek 1/2 strana

Závěr

V průběhu práce na návrhu a implementaci jazyka se osvědčil původní záměr flexibility kompilátoru. Při práci na druhé verzi překladače bylo sice nutno přidat jednu fázi (pro zvýšení vyváženosti komponent), nicméně větší část naprogramovaných prvků se mohla použít bez podstatných změn. Práce na druhé verzi jazyka se stala součástí provozního výzkumu první verze; nové části kompilátoru byly programovány v SYSTRANu. Zdá se, že by bylo užitečné obohatit jazyk SYSTRAN jednoduchým makrojazykem a zvýšit jeho návaznost na operační systém zavedením prostředků pro odvolání na některé jeho rezidentní části.

Literatura

(1.) Jazyk pro automatizaci systémového programování SYSTRAN, R.Bonhardová, J.Sokol, Acta polytechnica, IV, 1973.

(2.) Proceedings of A Singplan Symposium on Languages for System Implementation, SINGPLAN Notices, Vo6, No9.

(3.) MARY. Users dokumentation by Mark Rain, Reidar Conradi and Per Holager, RUNIT Rapport, 1973.

(4.) SYSTRAN - jazyk pro systémové programování, R. Bonhardová, J. Sokol, Sborník přednášek semináře "Překladače programovacích jazyků" ČVTS-FEL-ČVUT, 1974.