Modula-2 versus Pascal na počítačích PC

Miroslav Beneš, Jan M.Honzík

1. Úvod

Modula je nejmladší ze známých programovacích jazyků. Definice jazyka je obsažena v knize N.Wirtha : Programování v Module – 2. Tato kniha byla vydána již ve třech vydáních.

"Dobrý jazyk pochází z jedné hlavy" prohlásil N.Wirth, když se zklamáním končil práci na jazyku ALGOL 68. "V roce 1968 se zrodilo monstrum" prohlásil dále a v té době začal pracovat na Pascalu a později na Module. Modula – 2 vzešla podobně jako ADA ze základního požadavku na univerzální jazyk, ve kterém lze realizovat operační systémy, programy v reálném čase i velké uživatelské aplikační programy. První verze jazyka vznikly v souvislosti s projektem osobního počítače Lilith. Dobrý kompilátor Moduly vystačí s 5000 zdrojových řádků, zatím co kompilátor ADY, umožňující velké množství jazykových elementů a konstrukcí, potřebuje okolo 350 000 řádků zdrojového programu. To je pro každého tvůrce kompilátoru "alpský sen". Dobrý kompilátor Moduly má 2 až 3 "vše". Jak mnoho "vší" může asi mít kompilátor ADY? O Module – 2 lze říci, že je to Pascal bez nevýhod Pascalu, řekl Jerry Pournel v časopise BYTE. Použitelnost v praxi však stojí a padá s využitelností vývojových systémů jazyka.

2. Porovnání jazyků Modula – 2 a Pascal

První kompilátor jazyka Pascal byl vytvořen pro počítače řady CDC 6000. Jeho používání se zpočátku omezovalo pouze na univerzity. S tím, jak se stával dostupnější na stále větším počtu počítačů, roste jeho využití již rychlým tempem. Velmi populárními se staly rozšíření jako Concurrent Pascal pro programování v reálném čase nebo UCSD Pascal pro mikropočítače. S nástupem osobních

počítačů řady IBM PC převzala vedení řada kompilátorů Turbo Pascal firmy Borland International, jejichž poslední verze vytvářejí rozsáhlé integrované vývojové systémy obsahující kromě kompilátoru a editoru i prostředky pro údržbu programových systémů a pro ladění na úrovni zdrojového textu.

Jazyk Modula - 2 vznikl v době, kdy se Pascal již asi deset let používal a kdy už tedy byly zřejmé všechny jeho výhody a nedostatky. Na rozdíl od Pascalu, který byl původně vytvořen pouze pro účely výuky, byla Modula - 2 navržena již s ohledem na její využití v praxi pro vývoj velkých a spolehlivých programových systémů. To vedlo k přepracování některých jazykových konstrukcí a k rozšíření jazyka souvisejících především s odděleným překladem, soukromými datovými strukturami, přístupem ke strojově závislým prostředkům a obsluhou pseudoparalelních procesů. V porovnání např. s posledními verzemi Turbo Pascalu se tyto vlastnosti nejeví jako nové - podstatná je však ta skutečnost, že jsou součástí standardní verze jazyka Modula - 2 a jako takové jsou tedy vysoce přenositelné i na jiné počítače nebo pod jiné implementace jazyka.

V následujících odstavcích se zaměříme na některé zásadní odlišnosti obou jazyků. Nepůjde nám o detailní popis všech nových nebo změněných prvků jazyka, spíše o nastínění některých možností, které přinášejí. V porovnání vycházíme z normy ISO 7185 (třetí návrh z roku 1981) jazyka Pascal (překlad této normy je součástí [8]) a z referenční příručky jazyka Turbo Modula - 2 [7]. Norma jazyka Modula - 2 zatím vydána nebyla, za standardní verzi se pokládá verze popsaná prof. Wirthem ve třetím vydání knihy Programming in Modula - 2 (Springer - Verlag, 1985), jíž použitá referenční příručka odpovídá.

2.1 Prostředky pro modulární výstavbu programů

Základním rozdílem mezi oběma jazyky je možnost oddělené kompilace programových jednotek v Modula - 2. Standardní verze Pascalu předpokládá, že celý program bude zapsán v jediném zdrojovém souboru a v tomto tvaru překládán. Je zřejmé, že maximální velikost takto vytvořeného programu je omezena možnostmi kompilátoru. Při práci na rozsáhlejších programech je navíc velmi ztížena koordinace práce [illegible] amatérského týmu. Proto současně používané implementace jazyka Pascal umožňují provádět tzv. nezávislý překlad, kdy jsou části

programů (nazývané často moduly) samostatně překládány a vazba mezi nimi se uskutečňuje pomocí externích jmen v průběhu sestavování. Je na zodpovědnosti programátora, aby např. stejně deklaroval pořadí, počet a typ parametrů procedur a funkcí - překladač nemá žádnou možnost kontroly a tím se značně snižuje spolehlivost celého programového díla.

Modula - 2 místo nezávislého překladu zavádí pojem odděleného překladu; programátor může odděleně kompilovat tři typy kompilačních jednotek: definiční moduly, implementační moduly a programové moduly. Definiční a implementační moduly, nazývané souhrnně knihovními moduly, obvykle reprezentují určité datové a funkcionální abstrakce. Konstanty, typy, proměnné a procedury exportované definičním modulem mohou být použity (importovány) v mnoha dalších modulech - klientech. Ke každému definičnímu modulu musí existovat příslušný implementační modul. Tento modul obsahuje implementační detaily datových typů a procedur definovaných v definičním modulu. Programový modul představuje ekvivalent hlavního programu v Pascalu.

Vývoj programového systému v jazyce Modula - 2 spočívá v návrhu rozhraní (definičních modulů) mezi jednotlivými prvky systému (implementačními moduly), vytvoření konkrétních implementací a hlavního programového modulu. Pokud se nemění rozhraní modulů, nemá změna implementace vliv na ostatní složky systému; při změně definičního modulu je však třeba překompilovat znovu všechny moduly, které jsou jeho klienty - kontrola konzistence celého systému se provádí automaticky při překladu a sestavování programu, což je další z podstatných vlastností jazyka. Oddělením definice datového typu od jeho implementace je možné dosáhnout vysoké úrovně datové abstrakce používané především v objektově orientovaném programování. Programátor má možnost volit různé reprezentace abstraktních datových typů bez vlivu na další složky systému.

2.2 Práce s prostředky strojové úrovně

Standardní verze jazyka Pascal nedefinuje žádné prostředky pro práci na úrovni pojmů strojového kódu. Má to své opodstatnění, neboť je-li cílem normalizace jazyka zajištění přenositelnosti programů, není možné tyto prostředky definovat univerzálně. Vzhledem k tomu, že je Modula - 2 určena pro vytváření

systémových programů, byl do definice jazyka zařazen standardní modul SYSTEM definující datové typy BYTE, WORD a ADDRESS, procedury pro získání adresy a velikosti proměnné a několik procedur pro podporu abstrakce typu proces na úrovni pseudoparalelních úloh. Přístup k dalším strukturám (registry, instrukce, přerušovací systém, vybrány atd.) je realizován dalšími moduly, které jsou součástí knihovny modulů dodávané obvykle s překladačem.

2.3 Datové typy

Množinu standardních datových typů jazyka Pascal rozšiřuje Modula – 2 o celá čísla bez znaménka (typ CARDINAL). Jazyk klade přísnější požadavky na kompatibilitu typů ve výrazech – není možné např. kombinovat reálné a celočíselné operandy, programátor musí použít explicitního volání vhodné konverzní funkce tak, aby všechny operandy výrazu měly identické typy. Na druhé straně však zavádí poměrně nejasně definovanou funkci přetypování, kdy nedochází ke konverzi binárního obrazu dat, pouze se změní formálně jejich typ.

Významným rozšířením možností jazyka je zavedení typu procedura. Tím je dán mechanismus, jímž se mohou předávat procedury jako parametry jiným procedurám; programátor má rovněž možnost deklarovat proměnné typu procedura a pomocí nich provádět nepřímé volání podprogramů.

2.4 Procedury a předávání parametrů

Předávání parametrů mezi procedurami se v Module – 2 provádí obdobně jako v Pascalu hodnotou nebo odkazem. Procedurální parametry byly nahrazeny, jak už bylo uvedeno, koncepcí procedurálních typů.

Podstatnou změnou prošel způsob předávání polí s proměnnou délkou. ISO norma Pascalu zavedla jako novinku konformní schéma pole, pomocí něhož lze předávat do podprogramu zároveň s obsahem pole i hodnoty mezí jeho indexu. V Module – 2 jsou pro tento účel určena tzv. otevřená pole deklarovaná formálně bez indexu (ARRAY OF typ). V odkazech na prvky takto předaného pole se indexování provádí hodnotami typu CARDINAL v rozsahu od 0 po hodnotu vrácenou standardní funkcí HIGH, jejímž parametrem je konkrétní otevřené pole. Speciálně parametr typu ARRAY OF WORD (ARRAY OF BYTE) umožňuje zpracovávat data

libovolných typů – skutečným parametrem nemusí být nutně pole; proceduře je předána jeho adresa a délka ve slovech (stabilkách).

Počet standardních procedur a funkcí je v Modula-2 mnohem menší než v Pascalu. Je to dáno tím, že jejich velká část není přímou součástí jazyka, ale jsou k dispozici ve formě knihovních modulů. Tím se značně zvyšuje flexibilita jazyka, neboť uživatel si může vytvářet své vlastní moduly pro vstup/výstup, přidělování paměti nebo pro matematické operace. Samozřejmě jsou součástí systému překladače i standardní knihovny pro všechny běžné operace.

2.6 Příkazy

Oproti Pascalu obsahuje Modula-2 nový příkaz cyklu LOOP s možností ukončení uvnitř těla příkazem EXIT. Tato konstrukce umožňuje vytvářet složitější řídicí struktury bez nutnosti použít příkaz skoku. Tolik diskutovaný pascalovský příkaz skoku byl zcela zrušen; příkazy RETURN pro návrat z procedury a standardní procedura HALT pro ukončení programu však nejsou dostatečně ekvivalentní náhradou.

Příkaz cyklu FOR byl rozšířen o možnost uvedení kroku cyklu, vzhledem ke způsobu implementace příkazu cyklu však musí být hodnota kroku dána konstantním výrazem.

Velkou změnou, i když jen formální, která je zřejmá na první pohled při porovnání ekvivalentních programů v Pascalu a v Modula-2, je odstranění složeného příkazu BEGIN...END ve formě příkazových závorek. Syntaxe všech strukturovaných příkazů je zavedena tak, že umožňují přímý zápis posloupnosti příkazů i tam, kde Pascal dovoluje jen jediný příkaz – např. IF...ELSIF...ELSE...END. Nepříjemným důsledkem toho, že s výjimkou cyklu REPEAT jsou všechny složené příkazy ukončeny klíčovým slovem END, je obtížné zotavování kompilátoru po syntaktických chybách.

Ve výčtu rozdílů mezi Pascalem a Modulou-2 bychom mohli ještě dále pokračovat, ale pro základní seznámení postačí uvedené nejpodstatnější odlišnosti. Modulou-2 řada jazyků navržených prof. Wirthem nekončí, v poslední době se objevil např. jazyk Oberon, který se pokouší spojit – pro většinu uživatelů zcela pře-

kvalitním způsobem – prostředky datové abstrakce jazyka Modula – 2 se strukturami obdobnými jazyku C.

3. Srovnání vývojových systémů Moduly – 2 pro počítače IBM PC/AT

V [1] bylo uvedeno zevrubné srovnání 4 současně na trhu dostupných implementací Moduly – 2 pro počítače IBM PC. Výrobce, verze a cena jsou uvedeny v tab.3.1.

Interface Technologies (ITC)	M2SPS XP 2.1b	998 DM
Jensen & Partners International (JPI)	Topspeed Modula – 2	281 DM
LogiTech	Modula – 2 V1.1	754 DM
Stony Brook	Stony Brook M – 2 V1.1	884 DM

Tab 3.1. Výrobce, verze a cena vývojových systémů Modula – 2

Vlastnosti jednotlivých produktů jsou hodnoceny z mnoha různých hledisek, a o Modula – 2 se hovoří vždy jako o komplexu programů, které vytvářejí tzv. vývojový systém, jehož jednou částí je kompilátor jazyka. Do vývojového systému neoddělitelně patří editor, program MAKE (kontrolující konzistenci rozhraní mezi samostatně překládanými moduly např. na základě data poslední modifikace dováženého modulu), sestavovací program, knihovny programových modulů a programy pro podporu v době výpočtu a pro ladění programu. Mezi hodnocené vlastnosti patří např. uživatelský komfort při práci v integrovaném prostředí, možnost a úroveň spolupráce s produkty jiných programovacích jazyků, bohatost dodávaných knihoven a existence jejich zdrojových tvarů (která se cení vysoce), rychlost kompilace, efektivnost generovaného kódu, úroveň dokumentace apod.

ITC je nejstarším systémem. Má vlastnosti programů typu "desk – top" dovoluje vyvolání řady aplikačních programů, z nichž mnohé zvyšují spíše lesk než užitečnost. Má syntaxí řízený editor, rychlý kompilátor a výbornou knihovnu. Má však také některé vady a neumožňuje kombinaci s produkty jiných jazyků.

JPI je nejmladším systémem. Jeho cena, kvalita a komfort nápadně připomínají Turbo Pascal. Má rychlý jednoprůchodový kompilátor s dokonalou optimalizací, knihovny ve zdrojovém tvaru, vlastní assembler a ladicí program. Práce v jeho prostředí je zážitkem, který v lecčems předčí poslední verzi Turbo Pascalu

(současná editace ve čtyřech snadno zaměnných oknech s možností vzájemného přesunu bloků aj.). Jeho kvality a nízká cena vytvářejí vážného konkurenta ostatním hodnoceným systémům, ale i Turbo Pascalu.

Logitech je sobční systém známé firmy. Má ze všech nejbohatší knihovnu, mimo jiné i pro podporu práce s myší a pro grafiku (ale jen CGA). Má velmi rozsáhlou dokumentaci. Generovaný kód nevyniká, podobně jako u ITC, dokonalou optimalizací.

Stony Brook má dobré vlastnosti pro spolupráci produktů různých jazyků. Je rychlý a má bezchybný optimalizovaný kód. Pracuje i pod MS Windows a OS2. Některé chybějící prostředky omezují speciální aplikace. Nepracuje v integrovaném prostředí, jeho editor je však kompaktní a obsahuje všechny typické funkce pro práci se soubory.

4. Srovnání vývojového systému Top – Speed Moduly – 2 a Turbo Pascalu

Detailní srovnání obou jazyků by překročilo možnosti tohoto příspěvku. Dá se říci, že Turbo Pascal svými rozšířeními překonal mnohé potíže, které pro aplikace přinášel Pascal držící se normy. Mnohý vyspělejší programátor se však dostane do stadia, kdy všechna rozšíření přestávají pomáhat a to co je nutné, je nástroj na nové, vyšší úrovni. Tu nabízí Modula – 2. Při tvorbě rozsáhlých programů bude i pro znalce obou jazyků výběr jednoho z nich spíše otázkou osobního vkusu, ale u rozsáhlých projektů s týmovým přístupem k řešení, nebo pro problémy z oblasti systémů a paralelního programování, je volba Moduly - 2 zřejmá. Případem, snad nejvíce svědčícím pascalovské programátory, je parametr procedury typu pole. Ani netypový parametr neusnadní Turbo Pascalu situaci tak, jako parametr typu "otevřené pole" (např. "ARRAY OF integer"), který nespecifikuje rozsah indexu jednorozměrného pole. Verze 5.0 Turbo Pascalu, která vyšla brzy po JPI Modula – 2, opět téměř vyrovnává náskok Moduly (proměnná typu procedura, klauzule "usese", která jako součást implementační části umožňuje cyklické odkazy a ukrývání dat), ale v platnosti zůstává rozdíl mezi vylepšeným jazykem a novým jazykem.

	TP4	M2		TP4	M2
integer 2 byty			double precision 8 bytů		
i: = i	3.2		d: = d	49.9	8.2
i: = i + i	3.3	1.6	d: = d + d	86.1	75.8
i: = i - i	4.8	1.9	d: = d - d	91.1	75.3
i: = i*i	4.3	4.0	d: = d*d	101.6	90.1
i: = i div i	7.2	5.6	d: = d/d	116.0	114.7
longinteger 4 byty			single 4 byty		
li: = li	6.1	1.9	s: = s	43.3	7.1
li: = li + li	6.8	6.0	s: = s + s	76.2	64.2
li: = li - li	9.2	6.1	s: = s - s	78.6	64.9
li: = li*li	32.9	47.3	s: = s*s	77.9	78.6
li: = li div 8	304.9	57.8	s: = s/s	102.3	103.3

	TP4	M2
Plnění plochy body vertikálně (224000 bodů)	124.68	61.58
Plnění plochy body horizontálně -"-	123.85	61.40
Plnění plochy přímkami horizontálně	2.91	70.30
Plnění plochy "horizontálními" přímkami		6.10
Plnění plochy přímkami vertikálně	15.99	68.50
Šikmé přímky (640)	7.58	28.40
100 soustředných kružnic	21.7	8.87
Vyplněné obdélníky (175)	63.27	57.83

Tab.4.1. Rychlost aritmetických operací (80287) [mikros/oper] a základních grafických operací [s] (Graf.adapt.EGA)

Modula – 2 má však i své méně výhodné rysy. Někomu může vadit citlivost na velká a malá písmena, i použití jen velkých písmen pro rezervované identifikátory. Nutnost rozpisu kumulovaných V/V příkazů na sekvenci příkazů s jedním V/V parametrem programátora netěší, i když má své racionální příčiny. Protože je možné a vhodné přijímat Modulu – 2 jako nástroj navazující na Turbo Pascal a na jeho konkurenta, není pěkné, jsou – li podobné nebo shodné objekty (procedury, funkce, standardní identifikátory) pojmenovány trochu jinak v Module, než v Pascalu. Uživatele s vyznáním "pod obojí" to zbytečně plete a obtěžuje. Současný knihovní modul pro podporu grafiky je jen "chudším příbuzným" podobného

modulu v Turbo Pascalu. Očekávání, že jedna firma (Borland) dodá vynikající jazyk třídy Pascal i Modula s vysokým stupněm kompatibility, bylo odsunuto nástupem netradičně modulárního Turbo Pascalu 4.0 a potvrzeno verzí 5.0. Turbo Modula – 2, k níž vyšla i příručka pod názvem Modula – 2 Wizard [7] se zatím nekoná...

V tabulce (tab.4.1.) je uvedeno srovnání rychlosti aritmetických operací (kódovaný cyklus 100000 operací, procesor 80286 6MHz, koprocesor 80287) a rychlosti základních grafických operací. Z tabulky je zřejmá nevýrazná převaha Moduly v aritmetice, ale závěr ze srovnání v grafice závisí na typu častěji používaných operací. "Na body a kružnice" vede Modula, "na přímky" Turbo Pascal.

5. Ilustrační příklad použití Moduly - 2

Dynamické přidělování paměti (DPP) se používá v různých aplikacích a na jeho vlastnosti mohou být různé požadavky. Operační systém přiděluje např. rozsáhlé paměťové oblasti pro programy a data a frekvence požadavků přidělování a vracení není vysoká. Naopak při práci s dynamickými zřetězenými strukturami se přidělují poměrně málo rozsáhlé paměťové úseky, ale frekvence přidělování a vracení je vysoká, a proto i rychlost těchto operací je významná.

V tomto odstavci bude uvedena metoda DPP s regenerací paměťového prostoru při vracení paměťových jednotek nazvaná "Fibonacciho párový systém".

Definiční modul systému DPP má následující tvar:

```
DEFINITION  MODULE  Storage;
FROM  SYSTEM  IMPORT
(* type *) ADDRESS;      (* Dovoz datového typu "adresa" z modulu
                          SYSTEM  *)
EXPORT     QUALIFIED
    (* proc *) ALLOCATE,DEALLOCATE; (* vývoz kvalifikovaných
                                       identifikátorů pro operace přidělování
                                       a vracení paměťového prostoru *)
PROCEDURE     ALLOCATE(VAR     BlockAddress:ADDRESS;
                           BlockSize:CARDINAL);
PROCEDURE     DEALLOCATE(VAR     BlockAddress:ADDRESS;
                             BlockSize:CARDINAL);
```

5.1. Princip metody

Metoda vytváří pole seznamů volných bloků podle dané velikosti. Množina velikostí je vymezena hodnotami prvků Fibonacciho posloupnosti. Daný po-

žadavek na přidělení paměti se uspokojuje blokem o velikosti Fibonacciho čísla, které je nejblíže větší (nebo rovno) požadavku, a to s vědomím, že ve většině přidělených bloků je část prostoru nevyužita.

Blok se v procesu přidělení může rozdělit na dvojici bloků, jejichž velikosti jsou dány dvěma předchozími Fibonacciho čísly. Blok se dělí tak, že větší část bude na menší adrese – shodné s adresou otcovského bloku. Menší část bude na větší adrese. Každá část vzniklé dvojice (páru) obdrží rozdělovací kód, který bude určovat, zda jde o nižší (větší) nebo vyšší (menší) část dvojice. Kód větší části má o 1 větší hodnotu, než kód otcovského bloku. Kód menší části má vždy hodnotu 0. Tento kód umožní při vracení bloku do DPP rychleji nalézt "partnera", se kterým se spojí do většího bloku. Je - li kód bloku nulový, pak jeho partner má velikost o jedno Fibonacciho číslo větší a má nižší adresu. Protože každý blok obsahuje režijní informaci na svém začátku, lze adresu nižšího partnera snadno určit. Možnosti dělení jsou znázorněny na obr.5.1.

Menší ze dvou hodnot definujících Fibonacciho posloupnost musí být dostatečně velká na to, aby kromě nabízeného prostoru mohla zahrnout i režijní informaci (indikátor volnosti / obsazenosti, velikost bloku, rozdělovací kód a prostor pro dva ukazatele, protože seznam volných bloků je dvojsměrný). V dané implementaci jsou první dvě čísla 20 a 32 a posloupnost tvoří hodnoty 20, 32, 52, 84, 136, 220, 356, 576, 932, 1508, 2440.

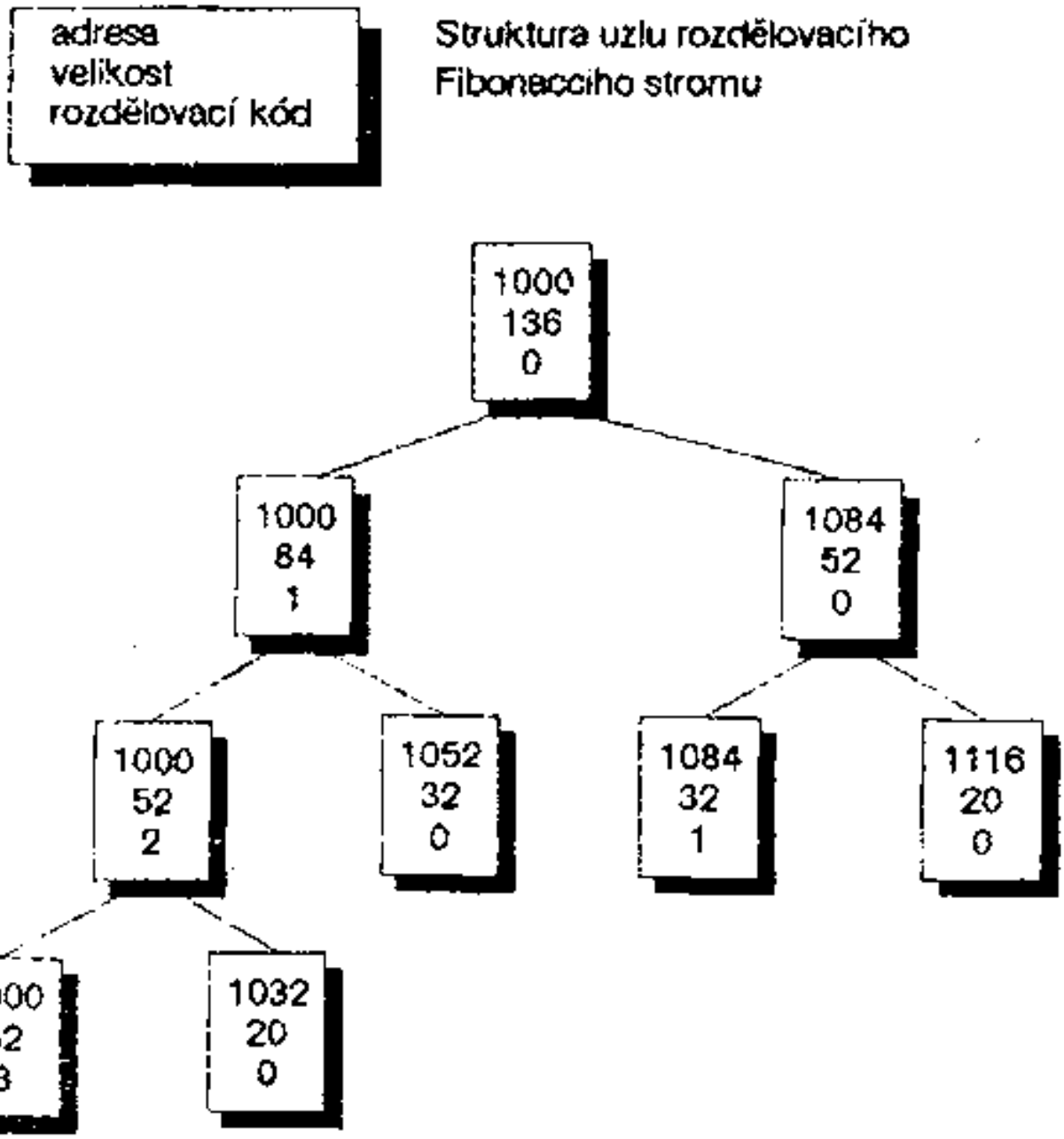
Velikost bloku je uložena ve formě indexu 0, 1, 2, ..., 10, protože volné bloky budou zřetězeny v deseti seznamech a v každém seznamu budou bloky o jisté velikosti. Přístup k seznamům bude zprostředkovat index pole FreeList. Kromě toho bude k dispozici pomocné převodní pole "LengthArray", které obsahuje na každém indexu jemu odpovídající délku bloku.

Dvojsměrné seznamy volných bloků mají hlavičku (pomocný první prvek seznamu), která je při inicializaci nastavena tak, že ukazuje sama na sebe. Pro práci se seznamem slouží soukromé operace InsertBlock, RemoveBlock a funkce EmptyList.

Modul obsahuje implementačně závislou operaci GetFirstBlock, o které se předpokládá, že je schopna využít možností operačního systému přidělit

počáteční blok paměti. Nemá - li systém tuto službu, přidělí se první blok staticky, jako velké pole.

Pozn. Písmena v závorkách v následujících dvou odstavcích se vztahují k o - značeným částem programu.



Obr. 5.1 Fibonacciho rozdělovací strom

Procedura ALLOCATE napřed určí nejmenší velikost bloku, která uspokojí daný požadavek (*A*) Velikost zahrnuje požadovaný prostor a potřebný režijní prostor. Pak procedura hledá první neprázdný seznam s bloky, které nejsou menší,

než je požadovaná velikost.(*B*) První nalezený blok se vyčlení ze seznamu.(*C*) Je – li blok příliš velký, začne se dělit tolikrát, až se vytvoří blok postačující velikosti.(*D*) Všechny nepoužité bloky vzniklé dělením, se vrátí do odpovídajících seznamů prázdných bloků.(*E*) Nakonec se zvýší adresa přidělovaného bloku o režijní velikost tak aby uživatel dostal adresu přiděleného prostoru až za režijním prostorem.

Procedura DEALLOCATE napřed sníží adresu vraceného bloku o režijní velikost, čímž získá adresu skutečného začátku vraceného bloku.(*G*) Pak zahájí cyklus, ve kterém se k vracenému bloku hledá partner z dvojice.(*I*) Z rozdělovacího kódu se určí, zda je partner na vyšší nebo nižší adrese a adresa partnera se spočítá. Otestuje se indikátor obsazenosti partnera a je – li partner volný, vyjme se ze seznamu volných bloků.(*J*) Pak se dvojice bloků spojí do jediného bloku a cyklus se opakuje ve snaze najít opět partnera k bloku právě vzniklému spojením dvou bloků. Cyklus končí, není – li partner volný, nebo má – li již regenerovaný blok velikost prvotního maximálního bloku. Výsledný blok se vloží do seznamu volných bloků.(*K*)

Tento mechanizmus DPP je poměrně velmi rychlý jak při přidělování, tak při vracení paměťového prostoru. Přiděluje však více paměti, než je požadováno, protože bloky přidělované paměti mohou mít jen velikosti čísel Fibonacciho posloupnosti. Jsou – li však první dvě hodnoty Fibonacciho posloupnosti zvoleny vhodně, pak nevyužitý prostor nepřekročí 10 - 20% celkového prostoru. Celkově lze říci, že ztráta prostoru je vyvážena rychlostí práce DPP.

```
IMPLEMENTATION  MODULE  Storage;
(* Fibonacciho párový systém DPP *)

FROM  SYSTEM  IMPORT
(*  type  *)  ADDRESS,
(*  proc  *)  TSIZE,ADR;

FROM  operatingsystem  IMPORT
(*  PROC  *)  obsysallocate;  (*  přidělení paměti  *)

CONST
    Block0Length = 20;    (* První hodnota Fibonacciho posloupnosti *)
    Block1Length = 32;    (* Druhá hodnota Fibonacciho posloupnosti *)
    MaxSizeIndex = 10;
    MaxSplitCode = MaxSizeIndex
```

```
TYPE
    SizeIndexRange = [0..MaxSizeIndex];
    SplitCodeRange = [0..MaxSplitCode];
    FreeBlockPtr = POINTER TO FreeBlock;
    FreeBlock = RECORD      (* Struktura bloku volné paměti *)
                    Free:BOOLEAN;   (* Indikátor volný/obsazený *)
                    SizeIndex:SizeIndexRange; (* Index velikosti bloku *)
                    SplitCode:SplitCodeRange; (* Rozdělovací kód *)
                    Next,Prev:FreeBlockPtr    (* Ukazatel na předchůdce
                                                 a následníka v seznamu *)
                END (* record *);

HeadArray = ARRAY SizeIndexRange OF FreeBlock;(* Pole hlaviček seznamů *)
LengthArray = ARRAY SizeIndexRange OF CARDINAL;(* převod index/velikost *)

VAR
    SystemOverhead:CARDINAL; (* velikost režijního prostoru *)
    FreeList:HeadArray;
    BlockLength:LengthArray;
    Index:SizeIndexRange;
    Block:FreeBlockPtr;

PROCEDURE   Failure;
BEGIN   (* Soukromá procedura reagující na nemožnost
           přidělit prostor *)
END Failure;

PROCEDURE       EmptyList(Index:SizeIndexRange):BOOLEAN;
(* predikát prázdnosti seznamu bloků na indexu Index *)
BEGIN
    RETURN      FreeList[Index].Next = ADR(FreeList[Index])
END EmptyList;

PROCEDURE   InsertBlock(Block:FreeBlockPtr);
(* vložení bloku Block do seznamu FreeList na odpovídající index *)
BEGIN
    Block^.Free := TRUE;  (* Indikátor "volný" *)
    Block^.Next := FreeList[Block^.SizeIndex].Next;
    Block^.Prev :=      ADR(FreeList[Block^.SizeIndex]);
    FreeList[Block^.SizeIndex].Next^.Prev := Block;
    FreeList[Block^.SizeIndex].Next := Block
END InsertBlock;

PROCEDURE       RemoveBlock(Block:FreeBlockPtr);
(* zrušení bloku Block *)
BEGIN
    Block^.Next^.Prev := Block^.Prev;
    Block^.Prev^.Next := Block^.Next
END RemoveBlock;
```

```
PROCEDURE    GetFirstBlock;
(* vytvoření prvního bloku a jeho vložení do seznamu *)
BEGIN
    opsysallocate(Block,BlockLength[MaxSizeIndex]);
    Block^.SizeIndex := MaxSizeIndex;
    Block^.SplitCode := 0;
    InsertBlock(Block)
END  GetFirstBlock;

PROCEDURE   ALLOCATE   (VAR   BlockAddress:ADDRESS;
                        BlockSize:CARDINAL);
VAR
    Block,Buddy:FreeBlockPtr;
    RequestSize,ActualSize:SizeIndexRange;
    BlockFound:BOOLEAN;
BEGIN
    IF(BlockSize + SystemOverhead)  < =  BlockLength[MaxSizeIndex)
        THEN (* požadavek je přijatelný *)
            RequestSize := 0;
(*A*)    WHILE  (BlockLength[RequestSize] < (BlockSize +  SystemOverhead))
                AND  (RequstSize < MaxSizeIndex)  DO  INC(RequestSize)
            END  (* while *);
            (* cyklus nalezl nejmenší vhodnou velikost bloku *)

            ActualSize := RequestSize;
(* B*)    WHILE  EmptyList(ActualSize)  AND
                    (ActualSize  <  MaxIndex) DO  INC(ActualSize)
            END  (* while *);
        (* cyklus nalezl nejmenší index neprázdného seznamu
    obsahujícího volné bloky o velikosti větší nebo rovné požadavku *)

            BlockFound := NOT   EmptyList(ActualSize);
            IF  BlockFound
            THEN (* přidělení tohoto bloku, nebo jeho části *)
(*C*)            Block := FreeList[ActualSize].Next;
                RemoveBlock(Block);

                (* následující cyklus rozdělí  blok tolikrát,
                    kolikrát je to nutné, aby se získal nejmenší
                    blok, který právě uspokojí daný požadavek *)

(*D*)            WHILE (ActualSize  <  RequestSize) AND
                    (ActualSize  > = 2) DO
                    Buddy := FreeBlockPtr(CARDINAL(Block)        +
                                                BlockLength[ActualSize - 1];
                        (* adresa pravého partnera v páru *)

                    Buddy^.SplitCode := 0;
                    INC(Block^.SplitCode);
```

```
                    DEC(Block^.SizeIndex);

                    IF RequestSize > (ActualSize - 2)
(*E*)               THEN (* přiděl větší část a menší
                            vrať zpátky *)

                        Buddy^.SizeIndex = ActualSize - 2;
                        InsertBlock(Buddy);
                        DEC(ActualSize) (* velikost o
                                           1 řád menší *)
                    ELSE (* přiděl z menší části,
                            větší vrať zpět *)
                        InsertBlock(Block);
(*E*)               DEC(ActualSize,2); (* velikost o
                                          2 řády menší *)
                        Buddy^.SizeIndex := ActualSize;
                        Block: = Buddy
                    END (* IF RequestSize > ...*)

                END (* WHILE *);

                (* přiděl blok a posuň jeho adresu
                    o režijní prostor *)
                Block^.Free: = FALSE;
(*F*)           BlockAddress: = FreeBlockPtr(CARDINAL(Block)
                                             + SystemOverhead)
            ELSE (* nebyl nalezen dostatečně velký blok,
                    je třeba žádat více od operačního systému *)
                GetFirstBlock;
                ALLOCATE(BlockAddress,BlockSize)
            END (* IF BlockFound *)
        ELSE (* požadovaná velikost je větší, než maximální
                možná velikost bloku *)
            Failure
    END (* IF (BlockSize + ... *)
END   ALLOCATE;

PROCEDURE   DEALLOCATE(VAR   BlockAddress:Address;
                             BlockSize:CARDINAL);
VAR
    Block,Buddy:FreeBlockPtr;
    BuddyFree:BOOLEAN;
BEGIN
(*G*)   Block: = FreeBlockPtr(CARDINAL(Blockaddress) - SystemOverhead);
    (* Block je skutečná adresa vraceného bloku *)
    BuddyFree: = TRUE;
(*H*) WHILE BuddyFree AND (Block^.SizeIndex < MaxSizeIndex) DO
    (* Opakovaná snaha o spojení vraceného bloku s partnerem a
       vytvoření nového bloku. Snaha končí nenalezne - li se volný
       partner, nebo má - li vytvořený blok již maximální možnou
```

```
        velikost *)

        IF Block^.SplitCode  >  0
            THEN (* blok je levým partnerem v páru, partner má
                    větší adresu *)
                Buddy. = FreeBlockPtr(CARDINAL(Block) +
                        BlockLength(Block^.SizeIndex));
                (* Buddy je adresa pravého partnera *)

                BuddyFree: = Buddy^.Free    AND
                            (Buddy^.SizeIndex = Block^.SizeIndex   1)
                (*BuddyFree vyjadřuje, zda pravý partner je celý
                    volný (nerozdělený) a dá se spojit s levým
                    partnerem do jednoho bloku *)

                IF  BuddyFree
                    THEN (* spoj obě části do jednoho celku *)
                        DEC(Block^.SplitCode);
                        INC(Block^.SizeIndex);
                        RemoveBlock(Buddy)
                    ELSE (* spojit nelze , partner je obsazen *)
                        InsertBlock(Block)
                END (* IF BuddyFree *)
            ELSE (* Block je pravým partnerem páru,
                    partner má menší adresu *)
                Buddy: =
                    FreeBlockPtr(CARDINAL(Block)
                                - BlockLength(Block^.SizeIndex + 1));
                (* Buddy je adresa levého partnera v páru *)
                BuddyFree: = Buddy^.Free    AND
                            (Buddy^.SizeIndex = Block^.SizeIndex + 1);
                (* BuddyFree vyjadřuje, že levý partner je celý
                    volný, a může se spojit do většího bloku *)
                IF  BuddyFree
                    THEN (* Spoj oba partnery do jednoho bloku *)
(*F*)                   RemoveBlock(Buddy);
                        Block: = Buddy;
                        Dec(Block^.SplitCode);
                        INC(Block^.SizeIndex);
                    ELSE (* spojit nelze, partner je obsazen *)
                        InsertBlock(Block)
                END (* IF BuddyFree *)
        END (* IF Block^.SplitCode  >  0 *)
    END (* while *);

    IF  Block^.SizeIndex = MaxSizeIndex
        THEN (* blok s největší velikostí nemá partnera *)
(*J*)   InsertBlock(Block)
    END (* if *)
END    DEALLOCATE;
```

```
BEGIN (* inicializační tělo modulu Storage *)
    (* stanovení velikosti režijního prostoru; v této implementaci
       musí být dělitelná čtyřmi *)
    SystemOverhead := TSIZE(FreeBlock) - 2 * TSIZE(FreeBlockPtr);
    IF SystemOverhead MOD 4 # 0
       THEN SystemOverhead :=
                          SystemOverhead + 4 - SystemOverhead MOD 4
    END (* if *);
    (* vytvoření převodní tabulky BlockLength pro převod mezi
       indexem a délkou bloku *)
    BlockLength[0] := Block0Length;
    BlockLength[1] := Block1Length;
    FOR Index := 2 TO MaxSizeIndex DO
       BlockLength[Index] :=
                          BlockLength[Index - 1] + BlockLength[Index - 2]
    END (* for *);
    (* vytvoření pole seznamů prázdných bloků
       (ukazatel ukazuje sám na sebe)   *)
    FOR Index := 0 TO MaxSizeIndex DO
       FreeList[Index].Next := ADR(FreeList[Index]);
       FreeList[Index].Prev := ADR(FreeList[Index]
    END (* for *);
    (* získání jednoho bloku největší velikosti od
       operačního systému *)
GetFirstBlock
END Storage.
```

6. Závěr

Modula – 2 patří k nejmladším programovacím jazykům. Rozsah jazyka i kompilátoru je malý. Řada funkcí jazyka je převedena do knihoven, které mají modulární výstavbu. Uživatel může knihovnu modifikovat, zcela změnit nebo rozšířit. Modula jako jazyk, úzce navazuje na pascalovskou tradici. Změny oproti Pascalu představují zjasnění a zjednodušení zápisu. Modulu - 2 lze jednoznačně doporučit jako jazyk, který vytváří přechod k další generaci jazyků, zaměřených zejména na objektové programování. Při volbě lze jednoznačně doporučit TopSpeed Modulu - 2 firmy Jensen & Partners International. Bylo by významným počinem, kdyby nějaká organizace získala licenci na distribuci tohoto produktu v našem prostředí. Na katedře počítačů FE VUT se připravuje implementace kompilátoru Modula – 2 na počítač EC 1027 pro potřeby výuky.

7. Literatura

[1] Braun,M., Guther,J.: Modula-2 wird erwachsen, Microcomputer - MC, September 1988

[2] Tyler,S.: The Confess of Converting to Modula-2 The Journal of the Turbo User group, 26, August/September 1988

[3] Corbeil,R.C., Anderson,A.H.; Modula-2 as a systems programming language, Byte, May 1988

[4] Cornelius,B.J.: Problems with the Language Modula-2 Software - Practice and experience, Vol.18(6),June 1988

[5] Honzík,J.M.: Technologie programování v programovacím jazyku Modula-2 , AK Slušovice 1988

[6] TopSpeed Modula-2 Dokumentace programového vybavení fy Jensen & Partners International

[7] Wiener,R.S.: Modula-2 Wizard John Wiley & Sons,Inc.,1986

[8] Jinoch,J.,Müller,K.,Vogel,J.: Programování v jazyku Pascal, Praha SNTL 1987

Doc. Ing. Jan Honzík, CSc.,
Ing. Miroslav Beneš,
katedra počítačů FE VUT v Brně,
Božetěchova 2, 612 66 Brno