1. Úvod 4

2. Všeobecný popis 4

3. Historie 4

4. Interpret 5

4.1 Postfixová forma PostScriptu 5

4.2 PostScript jako programovací jazyk 5

4.3 Jak se tvoří výstupní tisková stránka 5

4.4 Datové typy a objekty 7

4.4.1 Seznam typů v PostScriptu 7
4.4.2 Jednoduché objekty 7
4.4.3 Netriviální objekty 7
4.4.4 Číselné typy 8
4.4.5 Boolean 8
4.4.6 Pole (Array) 8
4.4.7 Procedurální pole (executable array) 8
4.4.8 Zhuštěné pole (packed array) 8
4.4.9 Řetězec (string) 8
4.4.10 Název (name) 8
4.4.11 Slovník (dictionary) 8
4.4.12 Operátor 9
4.4.13 Soubor (file) 9
4.4.14 Objekt bez významu (Null) 9
4.4.15 Informace o interpretu (Save) 9
4.4.16 Font (FontID) 9
4.4.17 Matice (Matrix) 9

4.5 Zásobníky 9

4.5.1 Zásobník operandů 9
4.5.2 Procedurální zásobník 10
4.5.3 Zásobník slovníků 11
4.5.4 Zásobník grafického stavu 11

4.6 Vykonávání spustitelných objektů 11

4.7 Lexikální analyzátor 12

4.8 Řízení 14

4.8.1 Bezprostřední vykonávání objektů 14
4.8.2 Odložené vykonávání objektů 14
4.8.3 Bezprostředně ohodnocená jména 14

4.9 Virtuální paměť 15

4.10 Chování systému při chybě 15

4.11 Grafika 16

4.11.1 Grafický stav 16
4.11.2 Souřadnicové systémy v PostScriptu 17
4.11.3 Transformace 18
4.11.4 Grafická cesta 18
4.11.5 Převod grafické cesty do rastrové grafické stránky 19
4.11.6 Bitové mapy (obrázky) 20

4.12 Fonty 21

4.12.1 Kódování znaků 22

5. Příklady PostScriptových programů 23
5.1.1 Příklad 1 (SQUARE.PS) 23
5.1.2 Příklad 2 (WEDGE.PS) 24
6. Účel programu EMSCRIPT 27
7. Instalace a ovládání 27
7.1 Systémové požadavky 27
7.2 Instalace 27
7.3 Používání programu 27
7.4 Okno zásobníku 28
7.5 Okno procedurálního zásobníku 29
7.6 Okno virtuální paměti 29
7.7 Okno aktuální grafické cesty 29
7.8 Okno aktuální grafické cesty ořezu 29
7.9 Okno výstupní grafické stránky 29
7.10 Jak používat program EMSCRIPT 30
7.11 Vestavěné operátory v interpretu 34
8. Vnitřní struktura programu EMSCRIPT 35
8.1 Struktura interpretu 35
8.2 Hlavní okno aplikace 36
8.3 Modul řízení interpretace 36
8.4 Virtuální paměť 37
8.4.1 Objekt TVirtMemory 37
8.4.2 Abstraktní objekt TCompObject 38
8.5 Lexikální analýza zdrojového textu 39
8.5.1 Object TFile (unit LEXIKAL.PAS) 39
8.5.2 Object TEditFile (unit LEXIKAL.PAS) 39
8.6 Objekty Zasobnik a BehZasobnik (unit ZASOBNIK.PAS) 40
8.6.1 Atributy: 40
8.6.2 Metody: 40
8.7 Objekt Slovník (unit SLOVNIKY.PAS) 40
8.7.1 Atributy: 40
8.7.2 Metody: 41
8.7.3 Vyhledávání v tabulce slovníku 41
8.8 Objekt SlovZas (unit SLOVNIKY.PAS) 42
8.8.1 Atributy: 42
8.8.2 Metody: 42
8.9 Správa řetězců, polí a procedurálních polí 43
8.9.1 Objekt Ret (unit OBJKOMP.PAS) 43
8.9.2 Objekt SubRet (unit OBJKOMP.PAS) 43
8.9.3 Objekt Proc (OBJKOMP.PAS) 44
8.9.4 Objekt SubArray (unit OBJKOMP.PAS) 44
8.10 Objekt Matrix (unit OBJKOMP.PAS) 45
8.10.1 Atributy: 45
8.10.2 Metody: 45

8.10.3 Použití: 45
8.11 Objekt GraphState (unit GRAFIKA.PAS) 45
8.11.1 Atributy: 46
8.11.2 Metody: 46
8.12 Objekt GraphPath (unit GRAFIKA.PAS) 46
8.12.1 Atributy: 46
8.12.2 Metody pro tvorbu grafické cesty: 47
8.12.3 Metody správy paměti pro ukládání popisu grafické cesty 47
8.12.4 Metody vykreslování grafické cesty do rastrové grafické stránky 49
8.13 Objekt GraphStack (unit GRAFIKA.PAS) 50
8.13.1 Atributy: 50
8.13.2 Metody: 51
8.14 Objekt Font (unit FONTY.PAS) 51
8.14.1 Atributy: 51
8.14.2 Popis funkce *GetGlyphOutLine*: 52
8.15 Převod HSV do RGB 53
8.16 Obsluha chybového stavu 53
8.16.1 Atributy: 53
8.16.2 Metody: 54
9. Závěr 55
10. Příloha A - Implementované operátory 56
11. Příloha B - Operátory, které nejsou implementovány 62
12. Literatura 67

1. Úvod

Mnoho článků na celosvětové síti Internet je uloženo a přenášeno ve formě PostScriptových souborů. PostScriptové soubory se též používají v profesionálních DTP pracovištích pro kvalitní přípravu černobílého i barevného tisku. PostScript se stal standardem v elektronické přípravě v tiskařském průmyslu. Vzhledem k tomu, že interprety PostScriptu jsou většinou součástí hardware tiskáren nebo osvitových jednotek, vznikl prvotní nápad vytvořit softwarovou emulaci takového interpretu. Takto vytvořený interpret měl poskytnout uživateli možnost prohlédnout si dokument v PostScriptu na počítači, který nemá k dispozici tiskárnu vybavenou odpovídajícím interpretem. V průběhu studia syntaxe jazyka a studia vnitřních mechanizmů interpretu jsem objevil několik již hotových emulací. Bylo jasné, že vytvořit dokonalý emulátor s úplnou množinou operátorů je velmi obtížný a rozsáhlý úkol. Žádný z interpretů , které jsem měl k dispozici, neumožňoval uživateli jednoduchými prostředky nahlédnout „pod povrch“ a seznámit jej tak s tím, jak pracuje s jednotlivými příkazy ze zdrojového textu. Dospěl jsem tak k rozhodnutí vytvořit emulátor, který toto bude uživateli poskytovat. Program si neklade za cíl být výkonný emulátor, který podporuje celou sadu PostScriptových operátorů, ale spíše prostředek, který dokáže ulehčit výuku PostScriptu. Aktuální verze programu obsahuje podmnožinu sady operátorů, a to takovou, aby bylo možno demonstrovat základní principy v PostScriptu. Program je však možno implementováním dalších operátorů rozšířit na celou sadu.

Diplomová práce obsahuje program EMSCRIPT, který jsem vytvořil, a text, který popisuje programovací jazyk PostScript a ovládání a strukturu programu EMSCRIPT. Text diplomové práce se skládá ze tří částí. První obsahuje popis interpretu PostScriptu, používaných typů objektů a popisuje řízení interpretu, tvorbu grafických objektů, zprávy paměti a výstupu do grafické stránky. Druhá část seznamuje čtenáře s prostředím programu EMSCRIPT a popisuje, jak tento program používat. Třetí část je pak určena pro čtenáře, kteří by chtěli zkoumat, jakým způsobem je program EMSCRIPT programován, jaké metody, postupy a datové struktury jsou zde použity.

2. Všeobecný popis

PostScript je programovací jazyk určený pro popis grafického vzhledu dokumentů. Jeho primárním úkolem je umožnit zobrazení textu, grafických tvarů a bitových map na tištěnou stránku. Tento jazyk umožňuje tisknout stejně vypadající stránky na různých výstupních zařízeních; je na výstupních zařízeních nezávislý (device independent). Popis stránky v PostScriptu může být pomocí interpretu převeden z textové podoby zdrojového textu do grafické podoby konkrétního zařízení a stránka může být zobrazena. Obyčejně je zdrojový text PostScriptu produktem jiných programů, jako je například textový nebo grafický editor. Programátoři píší své vlastní programy v PostScriptu, pouze když vytvářejí nějakou svou vlastní aplikaci nebo ve speciálních případech potřebují využít dalších možností interpretu a upravují zdrojové texty vzniklé automatickým generováním.

PostScript není jen sekvenčním popisem grafického vzhledu stránky, ale je programovacím jazykem, který podporuje několik datových typů, jako jsou celá i desetinná čísla, pole a řetězce, zahrnuje v sobě též řídící příkazy, např. podmíněné příkazy a cykly a některé méně obvyklé datové struktury, jako jsou slovníky (tabulky přiřazení). To umožňuje programátorům definovat operace, které plně vyhovují potřebám aplikace, a potom generovat stránku při plném uplatnění výhod interpretu. Popis grafického vzhledu stránky se pak stává v mnoha ohledech snažší a přehlednější. Programy v PostScriptu jsou vytvářeny a přenášeny ve tvaru zdrojových textů. Celý zdrojový text je psán v textové ASCII podobě. Tento tvar je jednak vhodný pro programátory, kteří se ve zdrojovém textu mohou okamžitě orientovat, a je též použitelný pro 7-bitový přenos (lze jej například posílat jako dopis elektronickou poštou).

3. Historie

Jazyk PostScript byl vytvořen ve firmě Adobe Systems Incorporated jako nový standard pro elektronický tisk. Předchůdce tohoto jazyka je možno nalézt u společnosti Evans & Sutherland Computer Corporation v roce 1976. Zde měl název Design system. Byl to výsledek výzkumného projektu, který zkoumal možnosti definice jazyka pro třídimenzionální grafické databáze. Projekt skončil úspěšně a tento jazyk byl použit jako podpora pro návrhářské aplikace. Přestože konstrukce jazyka byla podobná jazyku Forth, jejich koncepce a vývoj byl zcela nezávislý.

Hlavní myšlenky původního jazyka přinesl John Gaffney, který pracoval u Johna Warnocka (zakladatele firmy Adobe Systems). V roce 1978 se John Warnock připojil k firmě Xerox Pal Alto Reseach Center a spolu s Martinem Newellem upravili jazyk Design system a nazvali ho JaM. Martin Newell ho použil pro návrhy VLSI obvodů, zatímco John Warnock pro oblast tisku a grafického umění. Jedním z výsledků jeho práce v PARC byl vývoj tiskového protokolu Interpress. Mnoho myšlenek Interpressu nebo PostScriptu vychází z jazyků Design system a JaM.

V roce 1982 založili John Warnock a Chuck Geschke společnost Adobe Systems Incorporated a vytvořili třetí generaci tohoto jazyka. Nově vzniklý jazyk nazvali PostScript. PostScript je také určen pro popis grafiky, ale podporuje pouze dvoudimenzionální prostor tiskové stránky. Interpret PostScriptu je ve většině případů součástí tiskárny.

4. Interpret

Tato kapitola má za úkol seznámit čtenáře se základními principy programování v jazyce PostScript a obeznámit jej s vnitřními objekty, jako jsou např. zásobníky a virtuální paměť. Pro napsání této kapitoly byl použit uživatelský manuál firmy Adobe [1]. Pro sledování činností interpretu můžete použít program EMSCRIPT, který je součástí této diplomové práce a jehož ovládání je popsáno v kapitole 7. 3 Používání programu. Předesílám však, že tento text seznámí čtenáře se základními rysy jazyka PostScript a neklade si za cíl být důslednou učebnicí tohoto jazyka. V tom případě mohu odkázat na [12].

4. 1 Postfixová forma PostScriptu

Forma jazyka je postfixová, tzn. že první jsou ve zdrojovém textu zapsány operandy a teprve potom operátor. Příklad: *40 60 add* provede součet čísel 40+60. Tato forma je výhodná z toho důvodu, že není nutno používat závorek a pořadí provedení operací se řídí pouze pořadím operandů a operátorů. Tento rys velmi zjednodušuje syntaktickou analýzu PostScriptových programů, je však pro začátečníka nezvyklý a je třeba si na tento způsob zápisu zvyknout. Podrobnější popis s příklady je uveden v kapitole 4. 5 Zásobníky.

4. 2 PostScript jako programovací jazyk

Přestože primární úloha PostScriptu je tisk grafiky, jedná se o zcela obecný počítačový jazyk, nejvíce podobný jazyku Forth. Má mnoho obdobných datových typů s ostatními programovacími jazyky, ale definuje též odlišné typy jako je typ ***mark*** nebo ***dictionary*** (viz 4. 4 .1 Seznam typů v PostScriptu).

PostScriptové objekty typu dictionary (slovník) jsou používány pro ukládání definic procedur, proměnných a konstant (více viz 4. 4 .11 Slovník (dictionary))

V PostScriptu je možno definovat procedury, avšak procedury se seznamem parametrů jak je obvyklé např. v Pascalu nebo C, zde neexistují. Procedury jsou definovány a užívány stejně jako vestavěné operátory, parametry jsou předávány pomocí zásobníku operandů. Ve většině případů se komunikace mezi operátory nebo funkcemi provádí pomocí tohoto zásobníku (viz 4. 5 Zásobníky).

Vzhledem k procedurám není PostScript jazyk se statickou množinou příkazů a je možno používat některé popisy stránky v jiných složitějších popisech. Velmi dobrý příklad je Příklad 2 v kapitole 5 Příklady PostScriptových programů, kde je v první části definice procedury pro vytvoření jednoho lístku květu a v druhé části je pak tato procedura použita pro nakreslení celého květu.

4. 3 Jak se tvoří výstupní tisková stránka

Cílem této kapitoly je vysvětlit princip, jakým způsobem převádí interpret zdrojový text do výsledné stránky vytištěné na tiskárně. Interpret netiskne výslednou stránku přímo, ale nejprve si připravuje grafický návrh v paměti. Teprve, když je sestavena celá stránka, vytiskne ji na výstupní zařízení. Jsou tři skupiny výsledných tvarů:

1. Grafické tvary složené z posloupností úseček a křivek. Tyto tvary je pak možno libovolně zvětšovat, zmenšovat, rotovat a posouvat bez ztráty na kvalitě.

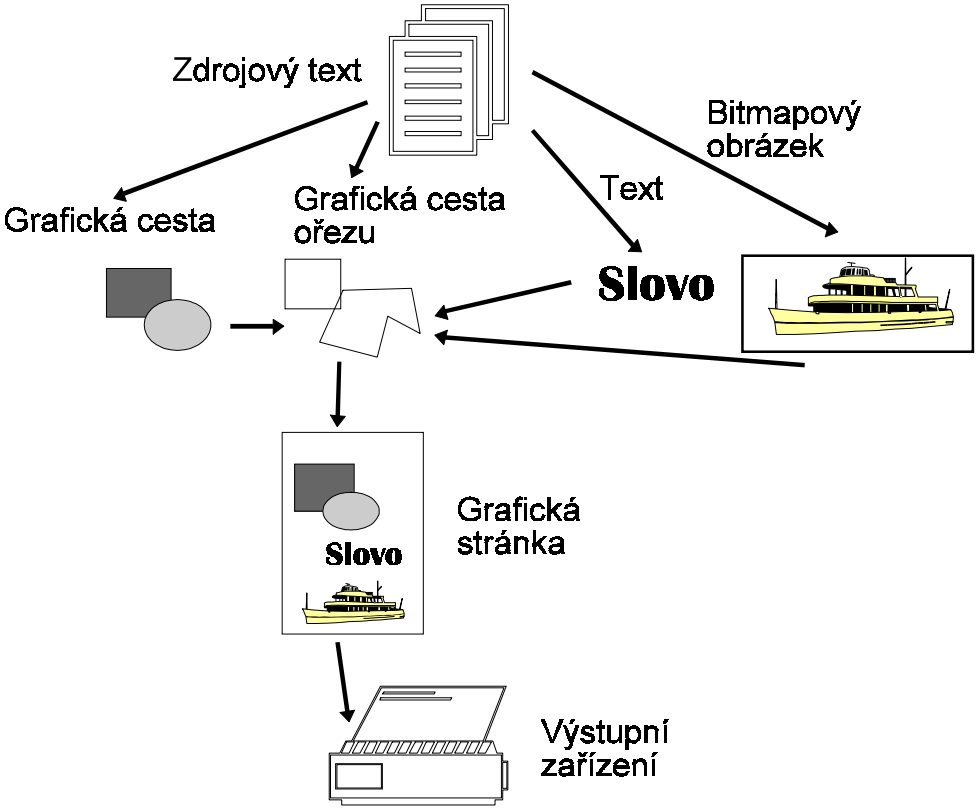
2. Text libovolné velikosti, umístění a otočení. Svou definicí jsou tvary jednotlivých písem podobné grafickým tvarům, avšak pro práci s textem má PostScript k dispozici jiné operátory, které jsou specializované na tisk písma, a tisk textu je prováděn jinou cestou, než tisk grafických tvarů.

3. Bitmapové obrázky složené z jednotlivých bodů. Mohou to být fotografie, nakreslené bitové mapy nebo různé vzory. Oproti grafickým tvarům mají výhodu, že dokáží popsat tak složité obrazce, jako je běžná fotografie, na druhou stranu však dochází ke ztrátě kvality, pokud nejsou zobrazovány v původní velikosti, to znamená, že jim nesvědčí zmenšování a zvětšování.

V případě grafických tvarů čte interpret zdrojový text a připravuje analytický popis tvarů, jak je předepsáno ve zdrojovém textu. Tento analytický popis se nazývá ***grafická cesta*** (viz 4. 11 .4 Grafická cesta). Dále interpret spravuje ***grafickou cestu ořezu***, což je analytický popis tvaru, který ohraničuje oblast, do níž je povoleno vykreslovat grafickou cestu. Jak grafická cesta, tak grafická cesta ořezu se může skládat z několika oddělených otevřených i uzavřených segmentů. Tyto segmenty se mohou libovolně protínat. Tento segment neboli podcesta je uzavřená, jestliže má totožný první a poslední bod podcesty. U uzavřených podcest je pak možno určit, co se nachází uvnitř a co vně podcesty (viz 4. 11 .5 .2 Operátor fill). Mimo grafické cesty a grafické cesty ořezu udržuje interpret ***grafickou stránku***, do které je přenášena kresba podle analytického předpisu grafické cesty. Tato grafická stránka je oblast v paměti, kde se již připravuje celá plocha jedné výstupní stránky. Pomocí operátorů **stroke** nebo **fill** se vykreslí vždy aktuálně nadefinovaná grafická cesta do grafické stránky. Pak je možno nadefinovat další grafickou cestu a opět ji přenést do grafické stránky. Tímto postupem je možno sestavit výslednou stránku určenou pro tisk. Teprve, když je v grafické stránce připraven výsledný obraz celé tiskové stránky, přenese se operátorem **showpage** tato grafická stránka na konkrétní výstupní zařízení.

V případě textů a bitmapových obrázků se nevytváří grafická cesta, ale interpret přímo zapisuje do grafické stránky. Ve všech případech se vždy respektuje grafická cesta ořezu, která ořízne vše, co je vně tohoto regionu. Tím je možno vytvářet různé efekty na výstupní stránce.

Cestu od zdrojového textu k vytištěné stránce ilustruje následující diagram:



V následujících řádcích ukáži na jednoduchém příkladu zdrojového textu, jak definovat grafickou cestu, jakým způsobem ji přenést do grafické stránky a jak dát příkaz k jejímu vytištění:

```
% Definice grafické cesty.
20 20 moveto                % definice počátečního bodu grafické cesty na [20; 20]
100 20 lineto               % definice úsečky [20; 20] - [100; 20]
100 100 lineto              % definice úsečky [100; 20] - [100; 100]
closepath                   % uzavření grafické cesty (vznikne tak trojúhelník)
% Přenesení grafické cesty do grafické stránky
stroke                      % vykreslení obrysu trojúhelníku do grafické stránky
% Vypsání textu do grafické stránky
50 300 moveto               % V bodě [50; 300] bude levý dolní roh textu
(Toto se vytiskne) show
% Přenesení obrazu grafické stránky na výstupní zařízení
showpage
```

Definicemi grafické cesty a vykreslováním do grafické stránky se podrobněji zabývá kapitola 4. 11 Grafika. V této kapitole je též popsán systém souřadnic používaný v PostScriptu.

4. 4 Datové typy a objekty

Všechna data přístupná programům PostScriptu zahrnující též procedury, které jsou součástí programu, existují ve formě objektů. Objekty jsou vytvářeny a využívány operátory PostScriptu. Objekty jsou vytvářeny přímo čtením vstupního textu. Každý objekt má přiřazen typ, atributy a hodnotu.

Poznámka: Ve třetí části textu diplomové práce se též hovoří o objektech. Tam se však jedná o objekty programovacího jazyka Pascal, ve kterém byl interpret naprogramován. Je tedy potřeba dát pozor na to, zda se hovoří o objektu PostScriptu nebo o typu programovacího jazyka Pascal.

4. 4 .1 Seznam typů v PostScriptu

integer	celočíselná hodnota -2147483648 až 2147483647
real	číslo s plovoucí řádovou čárkou $\pm 10^{38}$ $\pm 10^{-38}$ na 8 platných cifer.
boolean	pravdivostní hodnota
array	pole obsahující jiné objekty
executable array	pole spustitelných objektů
packed array	zhuštěné pole
string	řetězec znaků
name	řetězec určující název
dictionary	slovník přiřazení
operator	implementovaná činnost interpretu
file	externí soubor
mark	značka
null	objekt bez významu
save	objekt uložení aktuálního stavu systému
fontID	identifikace fontu

4. 4 .2 Jednoduché objekty

Jednoduché objekty jsou takové objekty, které nemají přiřazeny žádné podstruktury a typ, jejich atributy a hodnota jsou uzavřeny v jednom celku, manipulace s nimi je tedy velmi jednoduchá.

4. 4 .3 Netriviální objekty

Jedná se objekty se složitou strukturou, které zabírají větší oblasti paměti, než je kapacita pro jeden prvek v zásobníku operandů. Tyto objekty obsahují mimo popisu, který je používán jako reference, ještě podstruktury. Tyto se při kopírování objektů nekopírují, dochází pouze ke kopírování popisu a přiřazené datové podstruktury jsou sdílené.

4. 4 .4 Číselné typy

Typ Real představuje číslo s pohyblivou řádovou čárkou, s rozsahem hodnot $\pm 10^{38}$ na 8 platných cifer. Typ Integer je 4-bytové číslo v rozsahu celočíselných hodnot -2147483648 až 2147483647. Většina PostScriptových operátorů podporuje oba číselné typy a v případě potřeby provede interpret automatickou konverzi. Některé operátory však striktně vyžadují typ integer, potom má interpret k dispozici operátory konverze.

4. 4 .5 Boolean

Tyto objekty mohou nabývat hodnot true a false. Jsou převážně používány jako operandy pro řídící operátory if a ifelse.

4. 4 .6 Pole (Array)

Jednodimenzionální pole. Toto pole je heterogenní, může obsahovat kombinaci různých datových typů. Tím se liší od polí většiny programovacích jazyků, kde pole může obsahovat pouze jeden datový typ. Všechna pole mají indexy od 0 do n-1, kde n je velikost pole. PostScript podporuje pouze jednodimenzionální pole. Vícedimenzionální pole lze vytvořit jako pole polí. Pole je netriviální objekt, hodnoty elementů jsou sdíleny. Dokonce vytvořením pole jako podpole z nějakého existujícího pole budou tyto objekty hodnoty elementů sdílet.

4. 4 .7 Procedurální pole (executable array)

Pole, které se skládá z jednotlivých objektů PostScriptu. Toto pole může být vykonáváno interpretem PostScriptu. a je možno tímto způsobem definovat procedury.

4. 4 .8 Zhuštěné pole (packed array)

Zhuštěné pole je více kompaktní reprezentace pole. Je určeno pouze pro čtení. Výsledkem použití tohoto pole je úspora paměti na úkor rychlosti, protože elementy packed array jsou na rozdíl od obyčejného pole nestejně veliké (je nutno k poli přistupovat sekvenčně).

4. 4 .9 Řetězec (string)

Tyto objekty představují řetězce celočíselných hodnot v rozsahu 0 až 255. Řetězec se navenek chová jako pole objektů integer, avšak jeho vnitřní struktura je kompaktnější. Řetězce definované v částech zdrojového textu by se měly skládat ze základní množiny znaků ASCII. Znaky, které nejsou v této sadě je nutno zapisovat ve tvaru \000, kde 000 je oktalová hodnota. Řetězec je netriviální objekt.

4. 4 .10 Název (name)

Operandy typu name se v programu PostScriptu vyskytují ve formě identifikátorů, jak je známe v jiných programovacích jazycích. Název vzniká především přímo při interpretaci zdrojového textu, ale je možno jej získat i použitím operátoru konverze. Na rozdíl od typu string je name jednoduchý objekt. Přestože je Název definován jako řetězec znaků, tyto znaky se nechovají jako elementy pole. Interpret PostScriptu rozlišuje malá a velká písmena (case sensitive). Typu Name se používá jako klíče ve slovnících.

4. 4 .11 Slovník (dictionary)

Dictionary je tabulka přiřazení. Jejími elementy jsou páry PostScriptových objektů. První objekt z páru se nazývá klíč, druhý objekt je hodnota. Jazyk obsahuje operátory, které ukládají tyto páry do slovníků. Interpret pak vyhledává v těchto tabulkách a vrací místo klíčů odpovídající hodnoty. Klíče jsou obyčejně typu Name (přesto mohou být klíče jakékoli objekty mimo typu Null). Pokud chcete použít typ String jako klíč, je konvertován na typ Name. Slovníky se používají především k definici proměnných, obdobně jako v jiných programovacích jazycích, ale mají mnohem obecnější použití. Např. PostScriptový font je slovník, ve kterém je ke každému znaku přiřazena procedura pro jeho nakreslení.

Pokud interpret narazí na operand typu Name, automaticky prohledává slovníky uložené na zásobníku slovníků, (viz 4. 5 .3 Zásobník slovníků) až nalezne odpovídající hodnotu.

Slovník je netriviální objekt.

4. 4 .12 Operátor

Typ operátor obsahuje informaci o implementované proceduře interpretu. Operátory jsou asociovány se jménem ve slovníku SystemDict (viz 4. 5 .3 Zásobník slovníků). Tento objekt je jednoduchý a spustitelný. Interpret vykoná tento objekt tak, že zavolá implementovanou proceduru.

4. 4 .13 Soubor (file)

Soubor je vstupně-výstupní proud znaků používaný pro přenos dat mezi PostScriptem a prostředím. Přístup do souboru má několik variací: může číst nebo zapisovat řetězce znaků, atomické formule PostScriptu nebo binární data. PostScript podporuje standardní vstup a výstup. Standardní vstup je používán pro vstup zdrojového textu, standardní výstup pak pro chybová a stavová hlášení. Soubor je netriviální objekt.

4. 4 .14 Objekt bez významu (Null)

Objekt typu Null se používá pro naplnění prázdných nebo neinicializovaných hodnot v netriviálních objektech. Null je jednoduchý objekt.

4. 4 .15 Informace o interpretu (Save)

Objekt typu Save reprezentuje uchovaný stav systému. Je vytvářen a používán operátory Save a Restore.

4. 4 .16 Font (FontID)

Objekt typu font popisuje typ písma.

4. 4 .17 Matice (Matrix)

Objekt typu pole, které obsahuje 6 číselných prvků nutných k popisu transformace souřadnic. Matice je netriviální objekt.

4. 5 Zásobníky

Interpret PostScriptu obhospodařuje několik zásobníků, které spolu se systémovými proměnnými určují aktuální stav interpretu.

4. 5 .1 Zásobník operandů

Zásobník operandů může obsahovat libovolný objekt PostScriptu. Může se jednat o vstupní operand nebo o výsledek činnosti interpretu. Pokud operátor PostScriptu potřebuje jeden nebo více operandů, pak je obdrží právě z tohoto zásobníku. Výsledky operací pak interpret ukládá též na tento zásobník. Hloubka zásobníku operandů je 500.

Poznámka: Konkrétní hodnoty uvedené zde a na dalších místech v textu jsou převzaty z referenčního manuálu PostScriptu firmy Adobe Systems Incorporated [1].

Nyní uvedu příklad ukládání čísel do zásobníku a aritmetických operací, které s těmito uloženými čísly mohu provádět. Jakékoli číslo, které se objeví ve zdrojovém textu, je uloženo do zásobníku. Tzn. pokud zdrojový text obsahuje následující řádku:

10 50 -1.56

provede interpret následující činnosti:

1. Uloží číslo 10 na vrchol zásobníku.
2. Na zásobník uloží číslo 50 a číslo 10 posune o jednu hladinu níž.
3. Na zásobník uloží číslo -1.56 a první dvě čísla opět posune o hladinu níž.

Číslo -1.56 je nyní na vrcholu zásobníku a bude první vyzdviženo ze zásobníku. Po vyzvednutí tohoto čísla se obě zbývající čísla v zásobníku posunou o hladinu výš. Takže číslo 50 pak bude na vrcholu zásobníku.

Tak jako se ukládají na zásobník čísla, je možno ukládat libovolné PostScriptové objekty. Pro manipulaci se zásobníkem operandů je určeno několik speciálních operátorů (operandy nalevo od znaku ➔ zobrazují obsah zásobníku před vykonáním operátoru, operandy vpravo zobrazují obsah zásobníku po vykonání operátoru. Vrchol zásobníku je vždy poslední v seznamu operandů):

clear $obj_1..obj_n$ ➔ —
vyprázdní celý zásobník operandů

dup obj ➔ obj obj
provede kopii objektu uloženého na vrcholu zásobníku a tuto kopii uloží na vrchol zásobníku

exch obj_1 obj_2 ➔ obj_2 obj_1
zamění pořadí dvou nejvýše uložených objektů na zásobníku

pop obj_1 obj_2 ➔ obj_1
vybere ze zásobníku jeden objekt

roll obj_{n-1} .. obj_0 n j ➔ $obj_{(j-1) \bmod n}$ ..obj_0.. $obj_{j \bmod n}$
provede rotaci n prvních objektů na zásobníku o j kroků.

Operandy uložené na zásobníku slouží jako vstupní hodnoty pro implementované operátory. Pro názornost zde vysvětlím zápis a způsob provádění aritmetických výrazů v PostScriptu:

Mějme operátor **add**. Tento operátor představuje součet dvou čísel. V případě vykonávání této operace se jako první činnost provede vybrání dvou objektů ze zásobníku. Je tedy zřejmé, že první dva objekty od vrcholu zásobníku musí být číselné objekty. Operátor **add** tato čísla sečte a výsledek opět uloží na zásobník operandů.

Např. 50 60 add ➔ 110

V PostScriptu se nepoužívají závorky a navíc používání zásobníku dovoluje určitou volnost při definici, jak má být výraz vypočten:

Např. výraz *(6 / 3)+8* je možno v PostScriptu zapsat těmito způsoby:

8 6 3 div add Vstup: 8 6 3 div add
Zásobník:

Vrchol zásobníku:	3	2	10	
	6	8		
	8			

6 3 div 8 add Vstup: 6 3 div 8 add
Zásobník:

Vrchol zásobníku:	3	2	8	10
	6		2	

4. 5 .2 Procedurální zásobník

Procedurální zásobník připouští pouze uložení spustitelných objektů. To, že je objekt spustitelný, určuje jeden z jeho atributů a lze ho nastavit operátorem **cvx**. V případě, že nemá tento atribut nastaven, je rozpoznáván jako data a k odznačení tohoto atributu lze použít operátor **cvlit** (podrobněji viz 4. 6 Vykonávání spustitelných objektů). Tento zásobník reprezentuje call stack programu. Kdykoli interpret přeruší vykonávání některého spustitelného objektu, aby spustil jiný objekt, přerušený objekt uloží na procedurální zásobník. Po vykonání nového objektu pak interpret vyzvedne přerušený objekt ze zásobníku a pokračuje v jeho vykonávání. Tento zásobník je používán též pro řízení cyklů. Hloubka zásobníku je 250.

4. 5 .3 Zásobník slovníků

V zásobníku slovníků mohou být uloženy pouze objekty typu slovník a slouží k udržování aktuálních slovníků, ve kterých se budou vyhledávat objekty typu jména. V případě vyhledávání ve slovnících se prochází jednotlivé slovníky od vrcholu zásobníku a vyhledání se zastaví po nalezení prvního výskytu hledané hodnoty. Interpret například rozpozná objekt typu jméno, automaticky se spustí vyhledávání tohoto jména. V případě, že se hledané jméno nevyskytuje ani v jednom ze slovníků uložených na zásobníku, generuje se chybové hlášení. Aktuální množina slovníků na zásobníku tak určuje, jakým způsobem bude substituován vyhledávaný objekt. V zásobníku jsou implicitně zabudovány slovníky SystemDict a UserDict. SystemDict je na dně zásobníku a přiřazuje jména všech zabudovaných PostScriptových operátorů s implementovanými činnostmi interpretu. UserDict se nachází po spuštění interpretu na vrcholu zásobníku a dovoluje definovat nové dvojice přiřazení do systému slovníků aniž by bylo nutno deklarovat nový slovník. Kapacita UserDict je 200 párů jméno - hodnota. Do zásobníku slovníků je možno přidávat nově vytvořené slovníky, jeho hloubka je 20.

4. 5 .4 Zásobník grafického stavu

Grafický zásobník se používá k uložení aktuálního grafického stavu: grafické cesty, aktuální transformační matice, cesty ořezu. Pro uložení grafického stavu slouží operátor **gsave**. Takto uložený grafický stav je pak možno kdykoli obnovit vykonáním operátoru **grestore**. Hloubka zásobníku je 31.

4. 6 Vykonávání spustitelných objektů

Jednotlivé objekty přicházejí do interpretu ze dvou zdrojů: prvním je zdrojový text, kde se znaky pomocí lexikálního analyzátoru sloučí do atomických formulí (tokens), které se následně převádí do objektů PostScriptu, a tyto objekty se vykonávají. Druhým zdrojem jsou procedurální pole objektů. Toto pole se vykonává tak, že se postupně vykonávají objekty v něm uložené.

Během vykonávání programu interpret přepíná mezi čtením zdrojového textu a vykonáváním procedurálních polí. Příklad: Interpret čte zdrojový text, právě přečtenou atomickou formulí bude jméno, které se bude vyhledávat ve slovnících. Objekt nalezený ve slovníku bude procedurální pole. Interpret pak potlačí čtení zdrojového textu, na procedurální zásobník si zaznamená údaje potřebné pro návrat ke čtení a začne vykonávat objekty uložené v procedurálním poli. Když dosáhne konce pole, pak se podle hodnot uložených na procedurálním zásobníku zpátky přepne na čtení dalších atomických formulí ze zdrojového textu. Na procedurálním zásobníku se ukládají informace o přerušených zdrojích (zdrojové texty, procedurální pole) a případně řídící informace cyklů.

Interpret PostScriptu pracuje s objekty dvěma rozdílnými přístupy. Interpret však přesně nečiní rozdíl mezi daty a programem. Některé objekty PostScriptu mohou vystupovat jak v roli dat, tak i v roli části programu. Objekty, které mají nastaven atribut *spustitelný objekt*, interpret přímo vykonává. Objekty nemající nastaven tento atribut se rozpoznávají jako data a ukládají se na zásobník operandů, a to nezávisle na typu objektu. Dokonce PostScriptové operátory jsou rozpoznávány jako data, pokud nemají nastaven atribut spustitelný objekt.

Dále je popsáno, jakým způsobem se vykonávají jednotlivé typy objektů s atributem spustitelný objekt:

Pro některé objekty má vykonání stejný efekt, jako kdyby se jednalo o data (číselné objekty (real, integer), pravdivostní hodnoty, slovníky, mark, save, font).

Procedurální pole se uloží na zásobník operandů, pokud je rozpoznáno interpretem jako data (například při definici procedurálního pole). Pokud je však vyvoláno nepřímo při vykonávání jiných objektů (typu jméno nebo operátor), je procedurální pole vykonáváno jako procedura.

Spustitelný string se uloží na procedurální zásobník. Interpret pak použije tento string namísto standardního vstupu. Po dosažení konce řetězce se tento řetězec vyzvedne z procedurálního zásobníku.

Spustitelný soubor je též uložen na procedurální zásobník a použije se namísto standardního vstupu.

Spustitelný objekt typu jméno spustí prohledávání ve slovnících na zásobníku slovníků. Pokud je nalezen jako klíčová hodnota v některém ze slovníků, pak se vykoná přiřazená hodnota k této klíčové hodnotě.

Spustitelný objekt typu operátor vyvolá zabudovanou rutinu v interpretu.

4. 7 Lexikální analyzátor

Při čtení zdrojového textu sdružuje interpret jednotlivé znaky do atomických formulí podle syntaktických pravidel PostScriptu. Po rozlišení jedné atomické formule provede její transformaci do objektu PostScriptu a nakonec tento objekt vykoná. Příklad: Když skupina znaků jsou číslice po nichž následuje mezera , pak interpret tuto skupinu čísel vyčlení jako atomickou formuli, na základě této atomické formule vytvoří objekt typu číslo. Nakonec tento objekt vykoná (uloží číselný objekt na zásobník).

Lexikální analyzátor se při rozpoznávání atomických formulí řídí následujícím diagramem (pozn. diagram je vytvořen na základě [2]):

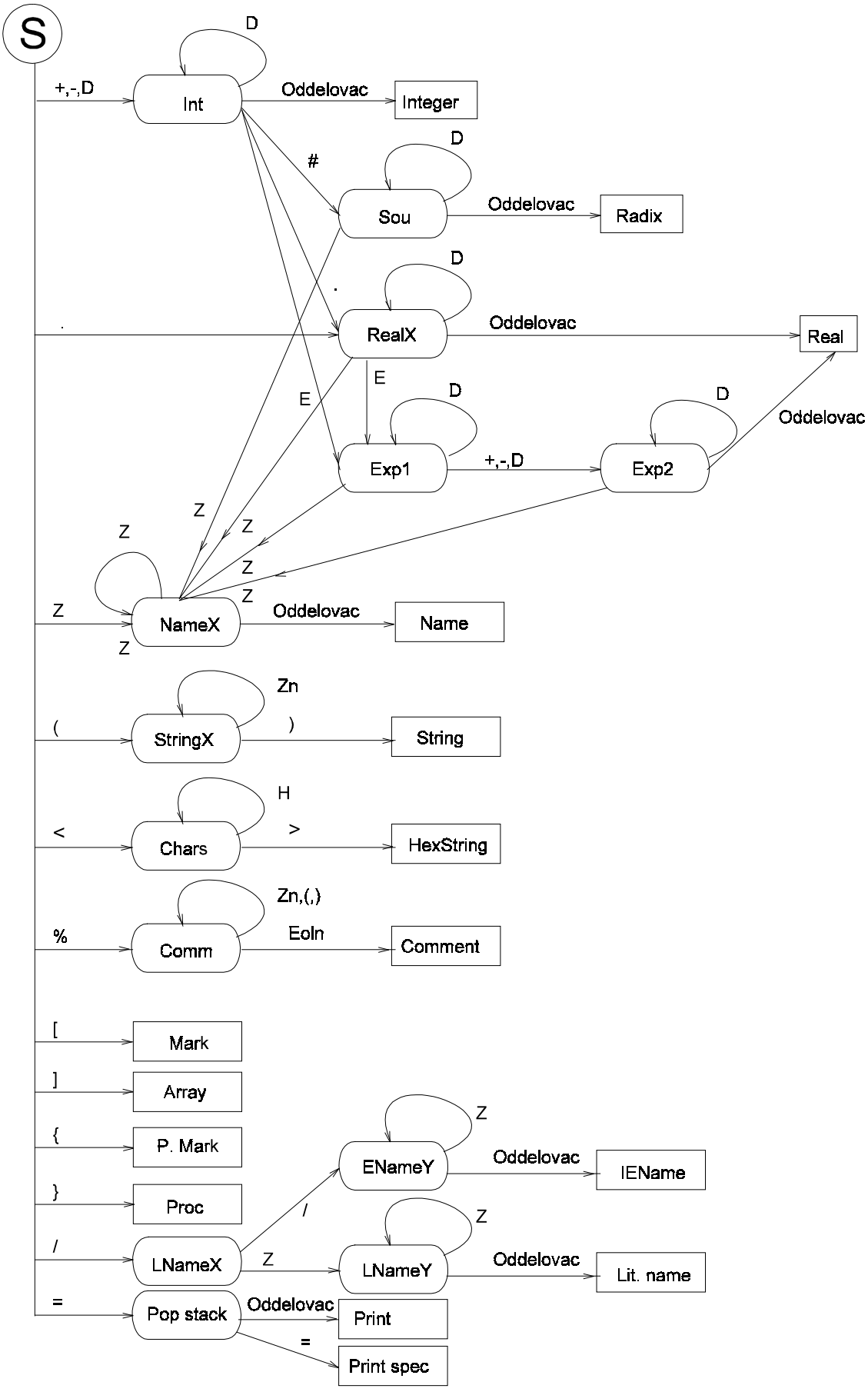
Terminály a neterminály:
Neterminální stavy jsou v oválech, koncové stavy lexikálního stavu zde představují obdélníky. Významy jednotlivých koncových stavů jsou popsány níže.

Zkratky ohodnocení hran:
D - desítková číslice
H - hexadecimální číslice
Zn - povolené znaky řetězce
Z - všechny ASCII znaky mimo znaků oddělovačů
Oddělovač - znak mezery, znak nové řádky, znak nové stránky, tabulátor, speciální znak
Eoln - konec řádku

Speciální znaky (vyznačené v grafu): (,),<,>,[,],{,},/,%,=

Koncové stavy:
Integer - celé číslo
Radix - celé číslo v explicitně definované soustavě
Real - reálné číslo
Name - řetězec znaků typu Jméno
String - řetězec znaků
HexString - řetězec hexadecimálních znaků, které se převedou na string
Comment - komentář (tato atomická formule se nepřevádí na objekt PostScriptu)
Mark - značka na zásobníku operandů (obyčejně se používá pro definování pole a je ekvivalentní operátoru **mark**
Array - ukončení definice pole
P.Mark - značka začátku definice procedurálního pole
Proc - značka konce definice procedurálního pole
Lit.name - jméno, které se bezprostředně nevykonává, ale pracuje se s ním jako se stringem
IEName - bezprostředně ohodnocené jméno
Print - vyzvednutí jednoho operandu ze zásobníku a vypsání jeho textové reprezentace na standardní výstup

Print spec - vyzvednutí jednoho operandu ze zásobníku a vypsání jeho syntaktické reprezentace na standardní výstup


S
+,-,D
Int
D
Oddelovac
Integer
#
Sou
D
Oddelovac
Radix
.
RealX
D
Oddelovac
Real
E
E
Exp1
D
+,-,D
Exp2
D
Oddelovac
Z
Z
Z
Z
Z
Z
NameX
Z
Oddelovac
Name
(
StringX
Zn
)
String
<
Chars
H
>
HexString
%
Comm
Zn,(,)
Eoln
Comment
[
Mark
]
Array
{
P. Mark
}
Proc
/
LNameX
=
Pop stack
/
Z
ENameY
Z
Oddelovac
IEName
LNameY
Z
Oddelovac
Lit. name
Oddelovac
Print
=
Print spec

4. 8 Řízení

4. 8 .1 Bezprostřední vykonávání objektů

Bezprostřední vykonávání je prováděno, pokud není prováděna definice procedurálního pole. To znamená, že ihned po rozpoznání objektu a jeho vytvoření se provede jeho vykonání. Příklad: Nechť *40 60 add 2 div* je segment PostScriptového programu. Interpret rozpozná číslo *40* a vytvoří číselný objekt, který vzápětí vykoná. Vykonání tohoto typu spočívá v jeho uložení na zásobník operandů. Podobně se zachová po rozpoznání čísla *60*. Nyní rozpozná objekt *add,* Toto jméno vzápětí vyhledá ve slovníku. Nalezne k němu přiřazený objekt typu operátor, vykonáním tohoto objektu se zavolá zabudovaná funkce sčítání, která vybere dva objekty ze zásobníku operandů, zkontroluje, zda jde o číselné objekty, a provede jejich součet, přičemž součtem vznikne nový číselný objekt s hodnotou *100*, který zabudovaná funkce uloží na zásobník operandů. Zbytek programu pracuje podobně. Interpret uloží na zásobník číselný objekt *2* a vyvolá zabudovanou funkci pro dělení. Funkce dělení vyzvedne dva operandy ze zásobníku, provede jejich vydělení a výsledek dělení opět uloží na zásobník operandů.

4. 8 .2 Odložené vykonávání objektů

Mějme příklad:
/average {add 2 div} def
40 60 average

Tento segment provede aritmetický průměr čísel *40* a *60*. Dává stejný výsledek jako příklad v předchozí kapitole. Interpret rozpozná nevýkonné jméno */average* (literal name) a uloží jej na zásobník. Dále rozpozná začátek definice procedury *{*. Začne sestavovat proceduru, která se skládá z objektů *add*, *2*, *div*. Tyto objekty však bezprostředně nevykonává, pouze je ukládá do definice procedury. Po rozpoznání konce definice procedurálního pole *}* vytvoří objekt procedura, který uloží na zásobník. Po rozpoznání jména *def* vyhledá odpovídající hodnotu ve slovnících uložených na slovníkovém zásobníku. Klíčové slovo *def* nalezne ve slovníku **SystemDict**, kde je mu přiřazena vestavěná funkce operátoru **def**. Operátor **def** vyzvedne objekt procedury a objekt nevýkonného jména ze zásobníku operandů a uloží tento pár do aktuálního slovníku (povětšině **UserDict**). Jinak řečeno první řádek příkladu vytvoří proceduru, která se bude volat jménem *average*. Druhá část příkladu pracuje následovně. Na zásobník se uloží čísla *60* a *40*, potom se rozpozná jméno *average*. Ve slovnících uložených na slovníkovém zásobníku vyhledá hodnotu přiřazenou k tomuto jménu. V tomto případě je to procedurální pole. Toto procedurální pole se vykoná: potlačí se čtení zdrojového textu a vykonají se postupně objekty uložené v tomto poli. V našem případě se nejprve vezme jméno *add,* ve slovníku se nalezne odkaz na vestavěnou funkci součtu. Tato funkce vyzvedne ze zásobníku operandů dva objekty, které musí být číselné, jinak vykonávání skončí chybou, provede jejich součet a výsledek uloží na zásobník jako číselný objekt (s hodnotou 100). Dále vykoná číselný objekt s hodnotou *2* (uloží 2 na zásobník operandů). Nakonec vykoná objekt typu jméno s hodnotou *div*. Ve slovnících nalezne odpovídající objekt, který je typu operátor, a zavolá vestavěnou funkci dělení. Tato funkce vyzvedne dva operandy ze zásobníku operandů (číselné objekty s hodnotami 100 a 2) a provede dělení. Výsledek (číselný objekt s hodnotou 50) uloží na zásobník operandů.

Odložení běhu spočívá v tom, že při definici procedurálního pole se objekty, které se jinak ihned vykonávají, ukládají na zásobník jako datové objekty a vykonávají se až při vykonávání procedurálního pole.

Procedurální pole se ihned po jeho definici ukládá jako datová struktura na zásobník operandů. Takto uloženou proceduru je možno vykonávat pomocí operátorů řízení běhu (operátory podmínky, cyklu...) nebo je možno ji uložit do slovníku pomocí operátoru **def**. Procedura se pak automaticky začne vykonávat po jejím nalezení ve slovníku pod patřičným jménem.

4. 8 .3 Bezprostředně ohodnocená jména

Jsou to jména, která mají ve zdrojovém textu tvar *//jméno*. Když interpret narazí na toto jméno, pak vyhledá objekt příslušející k tomuto jménu, avšak tento objekt se nevykoná, ale interpret s ním pracuje jako s datovou strukturou a jméno je substituováno příslušným objektem, právě tak, jak je možno na jméno aplikovat operátor **load**. Každý výskyt formule *//jméno* je vyměněn objektem asociovaným se jménem bez ohledu na to, jakého je tento objekt typu.

Příklad:
/a 3 def
/b {(test) show} def
potom
//a se substituuje číselným objektem s hodnotou 3
//b se substituce procedurálním polem {(test) show*}*
{//a //b a /b} nabyde po substituci tvar {3 {(test) show} a /b}.

Účel použití bezprostředně ohodnoceného jména je podobný použití operátoru **bind**: způsobuje to, že jména v proceduře jsou nahrazována přímo příslušnými objekty. Činnost operátoru velmi dobře ilustruje příklad v souboru BIND.PS v adresáři EXAMPLE na disketě s programem EMSCRIPT.

4. 9 Virtuální paměť

Virtuální paměť (VM) je paměťový prostor, kde se ukládají netriviální objekty. Tento prostor je využíván pro uložení řetězců, polí, procedurálních polí, slovníků. Rastrová grafická stránka, zásobníky a vestavěné funkce nejsou součástí VM. PostScript zahrnuje operátory **save** a **restore**, které umožňují běžícímu programu uložit a obnovit stav VM. Operátor **restore** vrátí všechny změny, které byly na hodnotách netriviálních objektů od zavolání korespondujícího operátoru **save**. Tyto operátory nemají vliv na obsah zásobníků, ani nevrací změny způsobené vedlejšími efekty, jako je zapisování do externích souborů nebo rastrové grafické stránky. Operátor **restore** zavře všechny soubory, které byly otevřeny po volání odpovídajícího **save**, ale neotevře soubory, které byly po **save** uzavřeny. Páry **save-restore** mohou být volány do hloubky 15 hladin.

4. 10 Chování systému při chybě

V průběhu vykonávání programu se může interpret dostat do chybového stavu. Tento stav může být způsoben sémantickou chybou v programu nebo chybou operačního systému (nedostatek paměti, neexistující soubor...). Když interpret detekuje chybu, spustí objekt typu jméno, kterým je popsána chyba. Podle tohoto jména vyhledá hodnotu ve speciálním slovníku **errordict.** Objekt přiřazený tomuto jménu je obyčejně procedura obhospodařující chybový stav. Implicitní procedura ve většině případů reaguje standardně: zaznamená informaci o chybě ve speciálním slovníku **$error** a potom vykoná operátor **stop**. Vykonáním operátoru **stop** se opustí nejvnitřnější procedura spouštěná pomocí operátoru **stopped** (pokud nastane uvnitř takového bloku), jinak interpret vyvolá vnější proceduru ošetření chyby. Tato procedura vykoná jmenný objekt **handleerror** ze slovníku **errordict**. Implicitní **handleerror** přijímá informace z **$error** slovníku a vydá zprávu o chybě podle konkrétního zařízení, na kterém běží. Někde **handleerror** jednoduše vypíše hlášení o chybě, v jiných systémech může provádět bližší diagnózu chyby.

Po výskytu chyby obsahuje slovník **$error** následující klíče:

1. newerror : boolean - nastavuje se na *true*, když je indikovaná chyba (**handleerror** tento atribut nastavuje na *false*)
2. errorname : name - identifikace chybového stavu, který nastal
3. command : cokoliv - operátor nebo jiný objekt, který se vykonával, když se objevila chyba
4. ostack : array - obsah zásobníku operandů, který byl uložen před vykonáváním objektu, kdy nastala chyba
5. estack: array - uložený procedurální zásobník
6. dstack : array - uložený zásobník slovníků

Program může změnit odezvu interpretu na chybu, například přidat některé další informace do chybového hlášení. Lze to provést dvěma způsoby: Předefinováním **handleerror** v **errordict** nebo přímým předefinováním objektů přiřazených ke klíčovým jménům chyb.

4. 11 Grafika

PostScript má mnoho prostředků pro tvoření grafických tvarů, rovinné transformace a převod grafického popisu do bitové mapy výstupního zařízení. PostScriptové operátory týkající se grafiky je možno rozdělit do 6 hlavních skupin:

1. **Operátory grafického stavu** - tato skupina zahrnuje operátory, které mění grafický stav systému. Grafický stav je seznam parametrů používaný pro vykreslování objektů, zahrnuje např. barvu, typ písma, tloušťku čáry apod. Hodnoty uložené v atributech grafického stavu jsou důležité pro ostatní skupiny grafických operátorů.

2. **Operátory souřadnicového systému** - operátory v této skupině manipulují s CTM (current transformation matrix), aby bylo možno dosáhnout požadovaných transformací z uživatelských souřadnic do souřadnic výstupního zařízení (viz 4. 11 .2 Souřadnicové systémy v PostScriptu). CTM je součástí grafického stavu.

3. **Operátory vytváření grafické cesty** - operátory v této skupině definují tvar vykreslovaného objektu. Tvar je uchováván v analytické podobě posloupnosti úseček a bézierových křivek. Tato posloupnost se nazývá ***grafická cesta***. Aktuální grafická cesta je součástí grafického stavu.

4. **Kreslící operátory** - tyto operátory převádí analytický popis grafických tvarů do rastrové grafické stránky, korespondující s výstupním zařízením.

5. **Operátory písma** - umožňují specifikovat a vytvářet fonty a vykreslovat text do rastrové grafické stránky.

6. **Operátory nastavení a výstupu na výstupní zařízení** - zajišťují převod grafické stránky na konkrétní zařízení. Když je grafická stránka kompletně připravena, spustí operátor **showpage** přenos na tiskárnu (fyzické výstupní zařízení).

4. 11 .1 Grafický stav

Interpret PostScriptu udržuje datovou strukturu nazývanou **grafický stav**, kde jsou uloženy aktuální grafické parametry. Tyto parametry určují, jakým způsobem bude vykreslována grafická cesta do grafické stránky. Např. operátor **show** použije atribut aktuálního fontu (current font) v grafickém stavu, operátor **fill** používá aktuálně nastavenou barvu (current color). Grafické stavy jsou udržovány na zásobníku. PostScriptový program může kopii aktuálního grafického stavu uložit na grafický zásobník pomocí operátoru **gsave**. Potom může libovolně měnit aktuální grafický stav. Po ukončení působnosti změn může program obnovit původní grafický stav operátorem **grestore**. Hloubka grafického zásobníku je 31.

Kompletní sada atributů grafického stavu je následující:

1. CTM : array - aktuální transformační matice. Tato matice převádí uživatelské souřadnice na souřadnice výstupního zařízení. Tato matice může být modifikována pomocí operátorů souřadnicového systému. Počáteční hodnota je určena podle konkrétního výstupního zařízení.

2. Color : interní - barva používaná během vykreslování grafické cesty do grafické stránky. Počáteční barva: černá

3. Position : 2 čísla - aktuální pozice v uživatelském prostoru (viz 4. 11 .2 Souřadnicové systémy v PostScriptu). Počáteční hodnota: pozice nedefinována.

4. Path : interní - aktuální grafická cesta. Počáteční hodnota: prázdná grafická cesta

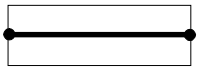
5. Clipping path : interní - cesta definující ořezávání při vykreslování do grafické stránky. Počáteční hodnota: celá tisková plocha výstupního zařízení

6. Font : dictionary - popisuje aktuální font používaný pro výstup textu. Počáteční hodnota je závislá na instalaci.

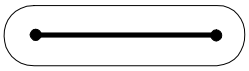
7. Line width : číslo - šířka čáry kreslené operátorem **stroke**.

8. Line cap : integer - určuje způsob zakončení čáry (silnou čarou je zobrazena definice v grafické cestě, slabá čára znázorňuje obrys čáry určité tloušťky vykreslované do grafické stránky)

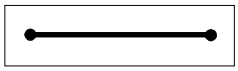
0 - butt cap



1 - round cap

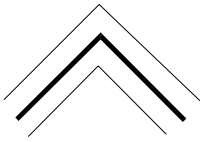


2 - projecting square cap

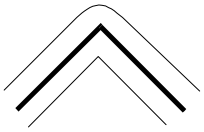


9. Line join : integer - určuje spojení dvou čar

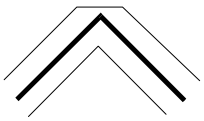
0 - mitter join



1 - round join



2 - bevel join



10. Halftone screen : interní - kolekce PostScriptových objektů, které definují aktuální vzor tónování pro barevný a černobílý výstup. Počáteční hodnota je závislá na instalaci.

11. Transfer : array - procedura, která převádí uživatelské stupně šedi na stupně šedi výstupního zařízení. Počáteční hodnota je závislá na instalaci.

12. Flatness : číslo - toto číslo udává přesnost aproximace bézierových křivek úsečkami. Počáteční hodnota je 1.

13. Mitter limit : číslo - maximální délka vybíhajícího růžku ze spojení dvou čar při Line join = Mitter join. Počáteční hodnota je 10.

14. Dash pattern : array - popis čerchování čáry kreslené operátorem **stroke**

15. Device : internal - interní datová struktura popisující výstupní zařízení. Počáteční hodnota je závislá na instalaci.

4. 11 .2 Souřadnicové systémy v PostScriptu

V souřadnicových systémech výstupních zařízení jsou veliké rozdíly (umístění počátku souřadnic, šířka a výška zobrazovacího okna). PostScript popisuje grafickou stránku, která by měla být nezávislá na výstupním zařízení, proto si zavádí vnitřní souřadnicový systém, tzv. uživatelský prostor (user space). Tento souřadnicový systém zaručuje, že vzhled grafické stránky bude mít stejné proporce na různých výstupních zařízeních a program v PostScriptu bude použitelný na různých výstupních zařízeních bez dodatečných zásahů do programu. Interpret automaticky transformuje polohu v uživatelském prostoru do prostoru aktuálně připojeného výstupního zařízení. Uživatelský prostor se inicializuje tak, že počátek souřadnic leží v levém dolním rohu, kladná souřadnice x je v horizontálním směru vpravo, kladná souřadnice y je ve vertikálním směru nahoru. Velikost jednotek podél osy x a y je 1/72 palce. Tento souřadnicový systém se nazývá standardní uživatelský prostor. Velikost jednotek 1/72 palce byla volena vzhledem k tomu, že to je zavedená jednotka v tiskařském průmyslu.

4. 11 .3 Transformace

V aktuálním grafickém stavu se nachází transformační matice CTM (current transform matrix) popisující transformaci z uživatelského prostoru do prostoru zařízení. PostScriptový program může měnit CTM. Je tak možno upravovat převod souřadnic z uživatelského prostoru do prostoru výstupního zařízení a tím provádět posun, rotaci a změnu měřítka vykreslovaných objektů na grafické stránce. Transformace se provádí již během definování grafické cesty a souřadnice uvedené v analytickém popisu grafické cesty jsou již v souřadnicích prostoru výstupního zařízení. Modifikace CTM mohou mít různé použití. Nejjednodušší je změna uživatelského souřadnicového systému pro celou stránku (v některých aplikacích potom může být uživatelský systém souřadnic vyjádřen v centimetrech, nebo může mít počátek jinde než v levém dolním rohu). Programově silnější je definování každého grafického elementu v jeho vlastním souřadnicovém systému, nezávislém na ostatních komponentech. To umožňuje popis tvaru nezávisle na jeho konkrétní velikosti a umístění ve výstupní grafické stránce.

Pro transformace jsou používány homogenní souřadnice (viz [2]) pro jejich velmi dobrou implementaci. Operátory **translate, scale, rotate** provádí násobení CTM určitou maticí transformace a tím modifikují způsob mapování:

1. Posun o DX ve směru osy x a o DY ve směru osy y

   DX DY **translate** Matice transformace:

1	0	DX
0	1	DY
0	0	1

2. Změna měřítka určeného koeficientem SX podél osy x a SY podél osy y

   SX SY **scale** Matice transformace

SX	0	0
0	SY	0
0	0	1

3. Otočení okolo počátku souřadnic proti směru hodinových ručiček o úhel α.

   α **rotate** Matice transformace

cos α	-sin α	0
sin α	cos α	0
0	0	1

4. 11 .4 Grafická cesta

Grafická cesta je definice tvarů a regionů všech druhů. Používá se pro definici obrysů, vyplňovaných ploch a regionů ořezu. Cesta je složena z úseček a bézierových křivek. Na tvary grafických objektů popisovaných grafickou cestou nejsou kladeny žádné zvláštní požadavky. Mohou být konvexní, konkávní, mohou obsahovat několik podskupin (reprezentující několik uzavřených ploch) , které se mohou libovolně překrývat.

Grafická cesta je reprezentována interní datovou strukturou, která není přímo přístupna jako objekt PostScriptu. Modifikace a použití grafické cesty je však možno za pomoci speciálních PostScriptových operátorů. Definici grafické cesty zajišťují *operátory vytváření grafické cesty* a vykreslení aktuální grafické cesty zajišťují *kreslící operátory*.

Aktuální grafická cesta je součástí grafického stavu. Tak, jako všechny jeho atributy, je možno uložit i aktuální cestu na grafický zásobník operátorem **gsave**.

Při definici cesty záleží na pořadí jednotlivých segmentů. Dvojice úseček je spojena pouze tehdy, když koncový bod první úsečky je zároveň počáteční bod úsečky druhé. Úsečky, které se protínají nebo dotýkají, ale nejsou definovány ihned po sobě, se nepovažují za spojené.

Podcesta je část grafické cesty, která se skládá ze segmentů (úseček, křivek) k sobě vzájemně spojených. Cesta je tvořena nejméně jednou podcestou. Existuje operátor **closepath**, který propojí úsečkou začátek a konec podcesty. Tímto aktuálně definovanou podcestu uzavře. Pokud se nepoužije tento operátor, zůstává cesta otevřená.

Definice cesty začíná operátorem **newpath**, který vyprázdní obsah stávající grafické cesty.

Grafická cesta se definuje pomocí uživatelských souřadnic. Každá souřadnice je transformována pomocí CTM do prostoru výstupního zařízení ve chvíli jejího záznamu do grafické cesty. Změna CTM tedy nemá vliv na již uložené souřadnice v grafické cestě.

Poslední bod v definici grafické cesty se zároveň stává tzv. aktuálním bodem (current point). Po operátoru **newpath** není aktuální bod definovaný. Většina operátorů, které přidávají segmenty do grafické cesty, požaduje, aby byl tento bod definovaný, jinak se generuje chyba **nocurrentpoint**.

Zde jsou hlavní operátory pro konstrukci grafické cesty:

1. **moveto** - definují nový aktuální bod bez vložení nového segmentu do grafické cesty. Tento operátor definuje začátek grafické podcesty. Používá se i pro první definici aktuálního bodu.
2. **lineto** - přidá úsečku do grafické cesty. Úsečka je připojena na aktuální bod.
3. **arc, arcn, arcto** - přidávají oblouk.
4. **curveto** - přidává bézierovu kubickou křivku.
5. **rmoveto, rlineto, rcurveto** - jsou podobné předchozím operátorům, ale pracují s relativními souřadnicemi.
6. **closepath** - přidá úsečku, která spojí začátek a konec grafické podcesty (uzavře podcestu).

Grafický stav též obsahuje cestu ořezu (clipping path) určující plochu, kam se může vykreslovat do rastrové grafické stránky. Operátor **clip** vytváří novou cestu ořezu z průsečíku aktuální grafické cesty s existující cestou ořezu.

4. 11 .5 Převod grafické cesty do rastrové grafické stránky

Operátory vykreslování převádí analytické vyjádření grafických tvarů do rastrové paměti, která je obrazem výstupního zařízení. Hlavními operátory tohoto převodu jsou **stroke** a **fill**. Operátory **image** a **show** též vykreslují do rastrové stránky, ale obcházejí aktuální grafickou cestu (oba jsou popsány dále).

4. 11 .5 .1 Operátor stroke

Operátor **stroke** kreslí čáry určité šířky. Výsledek vykreslení závisí na několika atributech grafického stavu:

1. barva (color)
2. šířka čáry (line width)
3. zakončení čáry (line cap)
4. spojení čar (line join)
5. hladkost aproximace křivky (flatness)
6. limit výběžku při spojení čar (mitter limit)
7. čerchování (dash)

Operátor **stroke** vykreslí jednotlivě všechny grafické podcesty. Způsob spojení dvou segmentů určuje atribut join line (mitter, round, bevel - nastavuje operátor **setlinejoin**). Pokud je vykreslovaná podcesta otevřená, pak se okrajové konce zakončí způsobem, který určuje atribut line cap (butt, round, projecting square - nastavuje operátor **setlinecap**). Linky mohou být plné nebo čerchované, to záleží na nastavení atributu dash. Barva čáry se řídí aktuálně nastavenou barvou (existují 3 operátory nastavení barvy: **setrgbcolor**, **sethsbcolor**, **setgray**). Operátor **setrgbcolor** nastavuje barvu určenou R, G, B hodnotami, **sethsbcolor** nastavuje barvu pomocí hodnot H, S, V a **setgray** nastaví barvu na šedou ze škály od bílé po černou pomocí hodnoty jasu. Atribut **flatness** určuje maximální odchylku aproximace bézierovy křivky pomocí úseček. Pro nastavení této hodnoty je možno použít operátor **setflat**.

4. 11 .5 .2 Operátor fill

Operátor **fill** vykresluje vnitřek oblasti, kterou popisuje aktuální grafická cesta. Pokud je cesta sestavena z několika podcest, pak **fill** při určování vnitřku tvaru uvažuje se všemi podcestami jako z celkem. Jakákoli otevřená podcesta bude před vyplněním uzavřena. PostScript jsou možné dva způsoby, jak definovat vnitřní oblast uzavřené grafické cesty:

Pravidlo *non-zero winding* určuje, zda bod náleží do vnitřní oblasti tvaru tak, že se z tohoto bodu sleduje myšlený libovolně nekonečně dlouhý nasměrovaný paprsek a sledují se průsečíky s vykreslovanou grafickou cestou. Inicializuje se čítač na 0 a při každém překročení grafické cesty podél myšleného paprsku se přičte nebo odečte 1. Přičítá se vždy, když grafická cesta křižuje paprsek ve směru hodinových ručiček, odečítá při křižování proti směru hodinových ručiček. Jestliže je po spočítání všech průsečíků výsledek rovný 0, pak není bod uvnitř tvaru, jinak se uvnitř tvaru nachází.

Druhé pravidlo *odd-even* počítá pouze počet průsečíků paprsku s grafickou cestou bez ohledu na její směr. Pokud je výsledek lichý, pak je bod uvnitř tvaru, je-li sudý, pak je vně tvaru.

Pravidlem *non-zero winding* se řídí operátory **fill** a **clip**, pravidlem *odd-even* se řídí operátory **eofill** a **eoclip**.

4. 11 .6 Bitové mapy (obrázky)

Bitová mapa je obdélníkové pole hodnot, v němž každá z nich reprezentuje barvu. PostScriptový obrázek je definován hodnotami stupňů šedi. Každá hodnota se sestává z 1, 2, 4 nebo 8 bitového čísla, které reprezentuje 2, 4, 16 nebo 256 stupňů šedi pro každý bod (pixel) obrázku.

Operátory, které nakreslí bitovou mapu do rastrové grafické stránky jsou **image** a **imagemask**.

Pro vykreslení grafické stránky potřebuje program znát tyto čtyři položky:

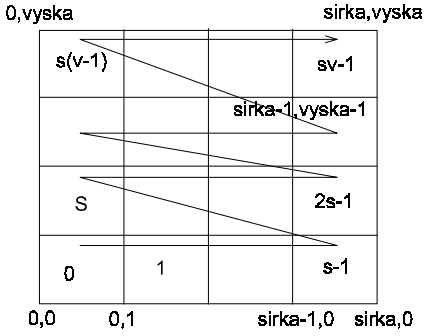
1. Formát zdrojové bitové mapy - počet sloupců (šířku), řádek (výšku) a počet bitů na jeden bod obrázku.
2. Samotný popis obrázku, sestávající z hodnot jednotlivých pixelů bitové mapy.
3. Vzájemnou korespondenci mezi uživatelským souřadnicovým systémem a souřadnicovým systémem bitové mapy.
4. Mapování stupňů šedi bitmapy a stupňů šedí výstupního zařízení.

4. 11 .6 .1 Reprezentace dat bitové mapy

Formát bitové mapy je určen hodnotami výšky, šířky a počtu bitů na pixel. Obrázek je reprezentován jako posloupnost znaků (celá čísla od 0 do 255). Každý znak obsahuje informaci o jednom nebo více pixelech, to závisí na formátu bitové mapy. Operátor **image** potřebuje jako jeden z operandů proceduru, kterou bude opakovaně vyvolávat. Tato procedura má za úkol vracet na zásobník řetězce určující bitmapu. Pokud tato procedura vrátí prázdný string, pak bude vykreslování bitové mapy předčasně ukončeno.

4. 11 .6 .2 Souřadnicový systém bitové mapy

Bitová mapa má svůj vlastní souřadnicový systém. Počátek souřadnic je v levém dolním rohu bitové mapy a velikost jednotek odpovídá vzdálenosti dvou sousedních pixelů. Bod o souřadnicích (0,0) odpovídá levému dolnímu rohu bitové mapy a na těchto souřadnicích leží levý dolní pixel., vedlejší pixel na stejné řádce má souřadnicích (1,0). Pixel v pravém horním rohu má souřadnice (šířka-1, výška-1). Souřadnice pravého horního rohu bitové mapy jsou však (šířka, výška).



Jedním z operandů operátorů **image** a **imagemask** je matice. Tato matice zajišťuje vzájemnou korespondenci mezi uživatelským prostorem a prostorem bitové mapy. Transformaci při vykreslování bitové mapy je možno rozdělit do dvou pomyslných kroků. Nejprve se převede jednotkový čtverec (0,0)-

(1,1) do hraničního obdélníku bitové mapy. Druhý krok mapuje jednotkový čtverec uživatelského prostoru do obdélníku v aktuálním uživatelském prostoru, který se použije pro přenos bitové mapy do grafické stránky. Tato konvence pak dovoluje definovat bitovou mapu nezávisle na tom, jak bude vykreslena. Např. pokud bude bitová mapa definována po řádcích (zleva-doprava) a zespoda nahoru, pak bude mít matice tvar [šířka 0 0 výška 0 0], a pokud bude matice definována po řádcích (zleva doprava), ale seshora dolu, bude mít matice tvar [šířka 0 0 -výška 0 výška]. S touto maticí je možno manipulovat jako s každou jinou maticí, čímž je možno bitovou mapou libovolně transformovat.

Vykreslování bitové mapy do rastrové stránky respektuje cestu ořezu. Je tudíž možno vyplňovat předem definované tvary.

4. 12 Fonty

Fonty jsou z hlediska PostScriptu slovníky. V každém slovníku fontů musí být obsaženy tyto informace:

1. FontMatrix : matrix - transformuje souřadnicový systém fontu do uživatelského souřadnicového systému. Vestavěné fonty PostScriptu jsou obyčejně definovány v 1000 bodových velikostech v souřadnicovém systému fontů. FontMatrix je pak inicializována na hodnotu [0.001 0 0 0.0001 0 0]. Pokud je velikost fontu měněna operátory **scalefont** nebo **makefont**, pak se nová matice transformace připojí ke stávající FontMatrix.

2. FontType : integer - určuje, jakým způsobem jsou fonty reprezentovány. Uživatelsky definovaný font má hodnotu FontType=3.

3. FontBBox : array - pole čtyř čísel, které určují (v souřadnicovém systému fontu) okrajový obdélník (bounding box) fontu. To je nejmenší obdélník, do kterého se vejde jakýkoli ze znaků fontu. Tato informace je používána pro cache a ořezávání.

4. Encoding : array - 256 párů, které mapují číselné kódy znaku (indexy pole 0..255) na jména jednotlivých znaků (hodnoty v poli typu string). Znaky diakritiky jsou popisovány celými jmény.

Slovník fontů je vytvořen jako PostScriptový objekt pomocí **definefont**. Operátor **definefont** vezme jméno fontu a slovník fontu, zkontroluje, zda slovník fontu odpovídá požadovaným konvencím, a vloží pár jméno-slovník fontu do globálního slovníku FontDirectory. Do slovníku fontu vloží tento operátor ještě jeden pár navíc. Tento pár má jméno FID a hodnotu FontID a je používán pro popis interních údajů, které PostScript potřebuje pro práci s fonty (například údaje o vyrovnávací paměti (cache)). Z tohoto důvodu je třeba při tvorbě vlastního slovníku fontů pamatovat na to, že slovník musí mít svou kapacitu takovou, aby se do něj vešel ještě tento pár.

Vestavěné fonty obsahují ještě tyto další informace:

1. FontName : name - název fontu (pouze pro informaci, není interpretem používán)

2. PrintType : integer - určuje způsob, jakým se budou fonty vykreslovat:
   0 - Popis znaků je určen pro plnou plochu (fill)
   1 - Popis znaků je určen pro konturu (stroke)
   2 - Popis znaků je určen pro fill, ale vykreslování bude probíhat jako obtah (outline)
   3 - Popis znaků je určen pro oba typy fill nebo stroke nebo jejich kombinace

3. Metrics : dictionary - informace o šířce a mezerách. Ve vestavěných fontech se obyčejně nepoužívá.

4. StrokeWidth : číslo - šířka kontury (v souřadnicovém systému fontu) pro outline fonty PrintType=2. Tento údaj není přítomen ve fill fontech, je nutno ho přidat, pokud z fill fontu tvoříte outline font.

5. FontInfo : dictionary - doplňující informace pro PostScriptový program (viz. dále).

6. UniqueID : integer - hodnota v rozsahu 0 - 16777215, která identifikuje font. Není úplně nezbytné, aby tato hodnota byla ve fontu přítomna, ale umožňuje používat cache pro rychlejší zobrazování fontu. Pokud tato hodnota ve fontu existuje, pak by každý font měl mít svou unikátní hodnotu (rozdílné hodnoty by měli mít i rozdílné verze toho samého fontu bez ohledu na to, jak

malé jsou mezi nimi rozdíly). Každý FontType má svůj vlastní nezávislý prostor UniqueID hodnot.

7. CharStrings : dictionary - přiřazuje jména znaků k popisům jejich tvarů.
8. Private : dictionary - obsahuje chráněné informace o fontu.

FontInfo obsahuje následující informace, které jsou určeny pro použití v programu. Nejsou využívány vnitřním zpracováním fontu v interpretu:

1. Notice : string - ochranná známka nebo copyright.
2. FullName : string - plné jméno fontu.
3. FamilyName : string - jméno skupiny, ke které font přináleží.
4. Weight : string - šířka fontu.
   1 - Bold
   2 - Medium
   3 - Light
   4 - Ultra
   5 - Heavy
5. Version : string - číslo verze fontu.
6. ItalicAngle : number - úhel odklonu od svislé osy ve stupních v případě kurzívy (italic).
7. IsFixedPitch : boolean - indikuje neproporcionální písmo (stejná šířka písmen).
8. Underlineposition : number - vzdálenost podtržení od základní čáry (basic line) znaku. Toto číslo je udáváno v jednotkách souřadnicového systému fontu.
9. UnderlineThickness : number - šířka podtržení (v souřadnicovém systému fontu).

4. 12 .1 Kódování znaků

Standardní fonty používají flexibilní kódové schéma, podle kterého se příslušnému kódu znaku vybere popis jeho tvaru. Toto kódování není přímo součástí fontu, ale je určeno v odděleném kódovém vektoru. Uživatelsky definované fonty musí vytvořit kódovací vektor sestavující se z kódů znaků v intervalu 0..255 a k nim přiřazených pojmenování. Program PostScriptu může měnit vektor kódování. To umožňuje tisk v rozdílných kódových stránkách.

V CharString ve slovníku fontu jsou klíči názvy znaku. Pro písmena je popis jednoduchý: „A“ má název ‘A’. U ostatních znaků jsou jména ve tvaru popisných názvů: ‘three’, ‘ampersand’, ’parenleft’.

Kódovací vektor je definován jako pole o 256 elementech, kde indexy 0..255 jsou kódy znaků a elementy pole jsou jména popisující jednotlivé znaky. Nalezení vhodného popisu tvaru znaku při vykreslování písma (např. operátorem **show**) se provádí v těchto krocích: Pro jednotlivé znaky v řetězci textu se vezme kód a v kódovacím vektoru se nalezne odpovídající jméno. Podle tohoto jména se v CharString vyhledá popis tvaru znaku.

Změnou existujícího kódovacího vektoru je možno jednoduchým způsobem vytvořit nový font s jiným vnitřním kódováním, tím že se pouze zamění hodnota v kódovacím vektoru (Encoding).

Toto flexibilní kódování znaků má použití hned ze dvou důvodů:

1. Umožňuje nastavit tisk textu v jiné znakové sadě než ASCII
2. Umožňuje aplikacím specifikovat a zahrnovat do tisku znaky, které se ve standardní znakové sadě nevyskytují. Některé fonty obsahují více než 256 znaků. V rámci PostScriptu není žádný standard, jak kódovat tyto znaky. PostScript zde nechává prostor programátorům.

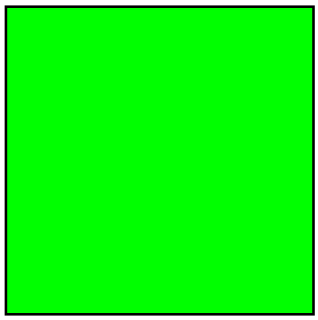
Kódovací vektor užitý ve většině standardních fontů je přiřazen ke jménu StandartEncoding v SystemDict. Všechny neobsazené pozice v kódovacím vektoru jsou vyplněny jménem '.notdef'. Výsledkem tisku těchto neobsazených znaků je malá mezera v textu.

5. Příklady PostScriptových programů

Pro představu, jakým způsobem se interpretuje zdrojový text PostScriptu jsou na následujících stránkách uvedeny příklady s komentáři. Tyto příklady lze též nalézt na přiložené disketě.

5. 1 .1 Příklad 1 (SQUARE.PS)

Tento program nakreslí čtverec se zelenou výplní a černým okrajem (zmenšeno):

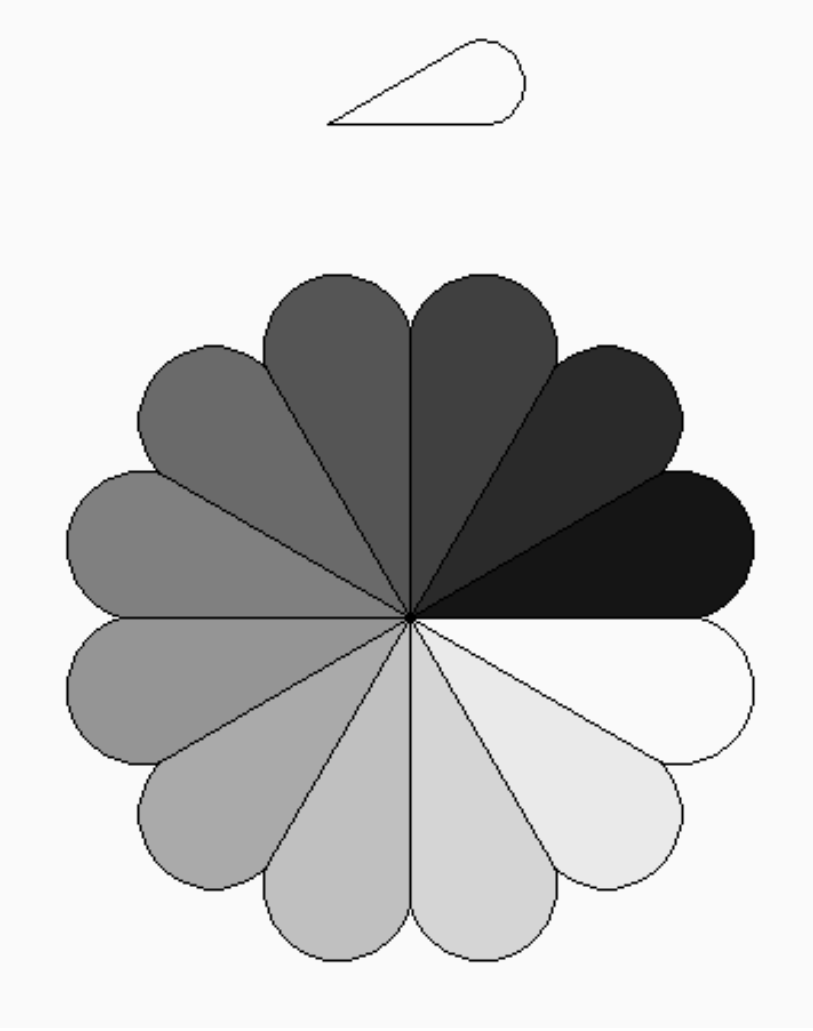


```
% Začátek souboru SQUARE.PS

/inch {72 mul} def              % Definice procedury inch, která násobí
                                % číslem 72. Po inicializaci interpretu
                                % jednotka v uživatelském prostoru odpovídá
                                % 1/72 palce. Pokud tedy chcete používat
                                % jako jednotky v uživatelském prostoru
                                % palce, je vhodné nadefinovat si tuto
                                % proceduru inch.

/square                         % Definice procedury square, tato procedura
                                % nakreslí čtverec o velikosti 1 palce, kde
                                % levý dolní roh leží v aktuálním bodě.
{ 1 inch 0 rlineto              % Vložení dolní vodorovné hrany čtverce do
                                % aktuální grafické cesty.
  0 1 inch rlineto              % Vložení pravé svislé hrany čtverce.
  1 inch 0 rlineto              % Vložení horní vodorovné hrany.
  closepath }                   % Uzavření aktuální grafické cesty - provede
                                % se spojení s výchozím bodem čtverce,
                                % který je dále uvažován jako uzavřená
                                % cesta.
def                             % Vložení procedury do slovníku.

gsave                           % Uložení aktuálního grafického stavu. To je
                                % dobré provádět vždy, když se budou měnit
                                % atributy grafického stavu.
newpath                         % Inicializace aktuální grafické cesty.
1 inch 1 inch moveto            % Definice aktuálního bodu se souřadnicemi
                                % [1;1] palce.
5 5 scale                       % Pětinásobné zvětšení měřítka.
square                          % Definice grafické cesty jako čtverce o hraně 5 palců..
0 1 0 setrgbcolor               % Nastavení barvy vykreslování na sytě
                                % zelenou.
```

```
gsave fill grestore                 % Vykreslení vnitřku čtverce, po fill se
                                    % automaticky inicializuje grafická cesta, je
                                    % proto třeba před vykreslením uložit
                                    % grafický stav.
0 0 0 setrgbcolor                   % Nastavení barvy vykreslování na černou.
stroke                              % Vykreslení kontury čtverce do grafické
                                    % stránky.

grestore                            % Obnovení původního grafického stavu
showpage                            % Vytištění grafické stránky na výstupním
                                    % zařízení.

% Konec souboru SQUARE.PS
```

5. 1 .2 Příklad 2 (WEDGE.PS)

Následující příklad je převzat z [12], kde se nachází několik dalších komentovaných příkladů. Výsledkem interpretace uvedeného příkladu bude následující obrázek (zmenšeno):



```
% Začátek souboru WEDGE.PS

/inch {72 mul} def                  % Definice procedury inch, která
                                    % provede násobení číslem 72, a
                                    % její uložení do slovníku.

/wedge                              % Následuje definice procedury
                                    % wedge, která provede nakreslení
                                    % jednoho okvětního lístku.
{newpath                            % nová grafická cesta.
 0 0 moveto                         % Nastavení aktuálního bodu na [0;0].
 1 0 translate                      % Přesunutí počátku souřadnic
                                    % do [1;0].
 15 rotate                          % Rotace souřadnicového systému
                                    % o 15° okolo počátku.
 0 15 sin translate                 % Posun o sin(15°) ve směru osy y.
 0 0 15 sin -90 90 arc              % Vložení do aktuální grafické
                                    % jedné hrany lístku a oblouku se
                                    % středem v [0,0] s poloměrem
                                    % sin(15°) od -90° do +90° proti
                                    % směru hodinových ručiček.
 closepath                          % Vložení do aktuální grafické
                                    % cesty druhé hrany okvětního
                                    % lístku. Tím je okvětní lístek
                                    % definován jako uzavřená cesta,
                                    % kterou je možno vyplnit.
} def                               % Zavedení definované procedury
                                    % do slovníku.

% Nakreslení jednoho okvětního lístku.

gsave                               % Uložení aktuálního grafického
                                    % stavu pro pozdější obnovení.
 3.75 inch 7.25 inch translate      % Přesunutí počátku do souřadnic
                                    % [3.75;7.25] palce.
 1 inch 1 inch scale                % Provedení změny měřítka tak, že
                                    % jednotka v uživatelském prostoru
                                    % nyní odpovídá jednomu palci
                                    % v prostoru výstupního zařízení.
wedge 0.02 setlinewidth stroke      % Procedura wedge vytvoří grafickou
                                    % cestu, tloušťka obvodové čáry se
                                    % nastaví na 0.02 jednotky a
                                    % vykreslí se obtah okvětního
                                    % lístku operátorem stroke. Vnitřek
                                    % okvětního lístku se nevyplní.
grestore                            % Návrat ke grafickému stavu před
                                    % začátkem kreslení okvětního
                                    % lístku.

% Nakreslení květu složeného z okvětních lístk.ů

gsave                               % Uložení aktuálního grafického
                                    % stavu pro pozdější obnovení
 4.25 inch 4.25 inch translate      % Přesunutí počátku do souřadnic
                                    % [3.75;7.25] palce.
 1.75 inch 1.75 inch scale          % Provedení změny měřítka na
                                    % 1.75 palce. Jednotka
                                    % v uživatelském prostoru nyní
                                    % odpovídá 1.75 palce v prostoru
                                    % výstupního zařízení.
```

```
0.02 setlinewidth                % Nastavení šířky čáry na 0.02
                                 % jednotek.
1 1 12                           % Uložení řídících parametrů pro
                                 % for cyklus: od 1 do 12
                                 % s přírůstkem 1.
                                 % Následuje definice těla for cyklu
{ 12 div setgray                 % každým průchodem for cyklu se
                                 % nastaví barva, kterou se bude
                                 % vykreslovat vnitřek okvětního
                                 % lístku na stupeň šedi tak, aby
                                 % lístky byly postupně vybarvovány
                                 % od černé až po bílou v proti
                                 % směru hodinových ručiček.
  gsave                          % Uložení grafického stavu.
   wedge                         % Definování grafické cesty
                                 % ve tvaru okvětního lístku.
   gsave fill grestore           % Vyplnění vnitřku okvětního
                                 % lístku. Operátor fill automaticky
                                 % vymaže obsah aktuální grafické
                                 % cesty po nakreslení výplně. Pokud
                                 % tedy chceme aktuální grafickou
                                 % cestu použít též pro jiný účel,
                                 % musíme před vykreslováním použít
                                 % operátor gsave.
   0 setgray stroke              % Nastavení barvy vykreslování
                                 % na černou a vykreslení obtahu
                                 % okolo zakreslené výplně.
  grestore                       % Obnova grafického stavu.
  30 rotate                      % Rotace uživatelského prostoru
                                 % o 30°. Zajistí se tím, že další
                                 % okvětní lístek se bude kreslit
                                 % orotovaný o 30°. Tato operace
                                 % způsobí, že se budou lístky
                                 % vykreslovat proti směru
                                 % hodinových ručiček. Pokud bychom
                                 % chtěli lístky vykreslovat
                                 % po směru hodinových ručiček musel
                                 % by tento příkaz být -30 rotate.
} for                            % Spuštění vykonávání těla
                                 % for cyklu.
grestore                         % Obnova grafického stavu.

showpage                         % Vykreslení grafické stránky
                                 % na výstupní zařízení.

% Konec souboru WEDGE.PS
```

6. Účel programu EMSCRIPT

Program EMSCRIPT jsem vytvořil pro uživatele, kteří se chtějí seznámit se základními funkcemi PostScriptového interpretu. V předcházející části byly zhruba popsány mechanizmy fungující v tomto interpretu, tak jak je definovala firma Adobe. EMSCRIPT není v některých případech přesnou kopií tohoto standardu, ale snaží se mu přiblížit tak, aby posloužil k prvnímu seznámení uživatele s činností takovýchto interpretů. Program není určen pro tisk rozsáhlých PostScriptových dokumentů ve vysoké kvalitě, je koncipován pro malé programy, které budou sloužit k objasnění základních principů. Grafická stránka je přístupná v jednom z náhledových oken, jehož obsah je možno vytisknout na tiskárně instalované ve WINDOWS™. Program též zahrnuje pouze základní podmnožinu operátorů. Tato podmnožina byla volena tak, aby bylo možno demonstrovat nejpoužívanější mechanizmy interpretace PostScriptového programu. Oproti komerčním produktům, určeným pro kvalitní tisk PostScriptových dokumentů, má EMSCRIPT zabudován systém monitorování vnitřních datových struktur interpretu. K programu je přiloženo několik ukázkových programů v PostScriptu, které je možno interpretovat pomocí programu EMSCRIPT a sledovat prováděné akce na zásobníku operandů a procedurálním zásobníku, obsah virtuální paměti, tvar aktuální grafické cesty, cesty ořezu a výslednou grafickou stránku.

7. Instalace a ovládání

7. 1 Systémové požadavky

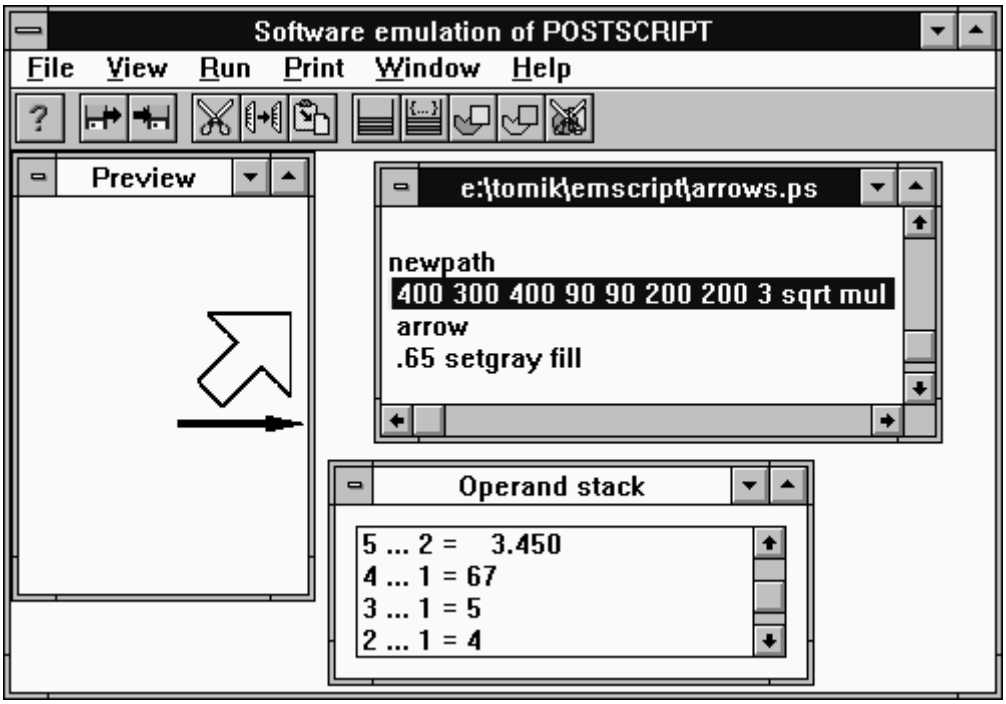
Pro správný běh interpretu je potřeba počítač IBM PC 386 nebo vyšší, minimálně 4 MB RAM, MS WINDOWS™.

7. 2 Instalace

Do libovolného adresáře nakopírujte soubory EMSCRIPT.EXE, BASICNUM.DTA a SYSDICT.DTA. Pokud chcete používat vzorové programy v jazyku PostScript, zkopírujte též soubory v adresáři EXAMPLE.

7. 3 Používání programu

EMSCRIPT dovoluje nahlížet do vnitřních struktur interpretu PostScriptu a tím lépe porozumět jeho činnosti. Tento rys usnadňuje výuku programovacího jazyka PostScript svou názorností a možností kontroly stavu interpretu po vykonání jedné řádky zdrojového textu. Zdrojový text načtený do okna je možno pročítat, upravovat a spouštět buď plynule volbou **Fast** v menu **Run** nebo po jednotlivých řádcích volbou **Step by step** v menu **Run** nebo klávesou ***F8***. V kterémkoli okamžiku je možno nastavit EMSCRIPT do počátečního stavu volbou **Reset interpret** v menu **Run**. Tato volba zajistí nastavení všech vnitřních struktur interpretu tak, aby se program mohl vykonávat znovu od začátku. Zdrojový text je možno načíst do okna, editovat a ukládat na disk. K tomuto účelu slouží volby **Open** a **Save** v menu **File**. Otevřená okna na pracovní ploše je možno uspořádat volbami **Tile** a **Cascade** v menu **Window**. Obsah okna výstupní grafické stránky je možno vytisknout, a to volbou **Graph page** v menu **Print**. Okna sloužící k nahlížení na vnitřní stav interpretu jsou popsána v dalším textu.



Ukázka hlavního okna programu EMSCRIPT

7. 4 Okno zásobníku

Okno náhledu na zásobník operandů je možno otevřít volbou **Operand stack** v menu **View**. Každý řádek v zásobníku představuje jeden uložený operand. První číslo v řádku je pořadí operandů v zásobníku, přičemž na vrcholu zásobníku je operand s nejvyšším pořadovým číslem. Druhé číslo označuje typ uloženého operandu:

Null (operand bez významu)	= 0
Integer (celé číslo)	= 1
Real (desetinné číslo)	= 2
Name (jméno)	= 3
Mark (značka)	= 5
Start def proc (začátek definice procedury)	= 7
Literal Name (nevýkonné jméno)	= 9
Logic (logická hodnota)	= 15
Dictionary (slovník)	= 20
File (soubor s textem)	= 21
String (řetězec)	= 41
SubString (odkaz na řetězec)	= 42
Array (pole)	= 51
SubArray (odkaz na pole)	= 52
Matrix (matice)	= 53
Font (písmo)	= 54
DefProc (začátek definice procedury)	= 60
Proc (procedura)	= 61
Operátor	= 70

Třetí hodnota v řádku je závislá na typu operandu.

První v seznamu je operand, který je na vrcholu zásobníku. Poslední prvek je operand na dně zásobníku. Okno se automaticky aktualizuje po vykonání každého řádku zdrojového textu.

7. 5 Okno procedurálního zásobníku

Okno náhledu na procedurální zásobník je možno otevřít volbou **Executable stack** v menu **View**. První v seznamu je operand, který je na vrcholu zásobníku. Poslední prvek je operand na dně zásobníku. Okno se automaticky aktualizuje po vykonání každého řádku zdrojového textu.

7. 6 Okno virtuální paměti

Okno náhledu na kompletní obsah virtuální paměti (viz 4. 9 Virtuální paměť) je možno otevřít volbou **Virtual memory** v menu **View**. Okno se automaticky aktualizuje po vykonání každého řádku zdrojového textu. Jeden řádek v okně virtuální paměti znázorňuje jeden netriviální objekt, který je obsažen ve virtuální paměti interpretu. První číslo v řádku je pořadové číslo. To odpovídá odkazu do virtuální paměti MemDescr, pomocí kterého virtuální paměť poskytuje uložené objekty (viz 8. 4 Virtuální paměť). Druhé číslo vyjadřuje typ uloženého objektu:

1 Textový řetězec
3 Pole
5 Matice
6 Slovník
7 Grafický stav
8 Grafická cesta
9 Font

Typ objektu je pak ještě slovně vypsán ve čtvrtém sloupci, kde jsou případně další informace o objektu. Tyto informace závisí na konkrétním typu objektu. Třetí sloupec vyjadřuje počet sdílení tohoto objektu. Pokud tento objekt sdílí tři datové struktury v PostScriptu, pak je zde číslo 3. Není zde možno objevit objekty, které by měli ve třetím sloupci 0. Tyto objekty nejsou již v interpretu využívány a jsou automaticky uvolňovány z paměti. Namísto těchto objektů se zobrazuje slovo NIL.

7. 7 Okno aktuální grafické cesty

Okno aktuální grafické cesty je možno otevřít volbou **Graph path** v menu **View**. V tomto okně se konturami zobrazí tvar, který zobrazuje aktuální grafickou cestu (viz 4. 11 .4 Grafická cesta).

7. 8 Okno aktuální grafické cesty ořezu

Okno aktuální grafické cesty ořezu je možno otevřít volbou **Clip path** v menu **View**. V tomto okně je vidět černou plochu, která určuje region, do kterého je povoleno kreslit. Tento region je určen aktuální cestou ořezu (4. 11 .4 Grafická cesta). Zelenými konturami je zobrazena aktuální grafická cesta. Po vykonání operátorů pro převod do grafické stránky (**stroke** nebo **fill**) bude do grafické stránky vykreslena ta část grafické cesty, která leží v černé oblasti cesty ořezu.

7. 9 Okno výstupní grafické stránky

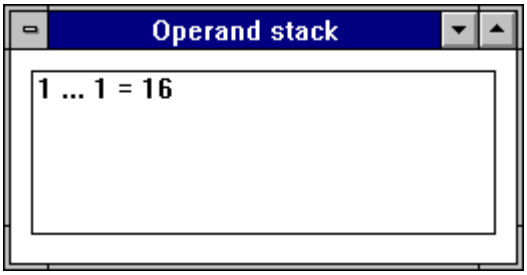
Okno je automaticky otevřeno při spuštění EMSCRIPTu. Obsahuje kompletní obraz rastrové grafické stránky připravované k výstupu. Okno se aktualizuje po každé instrukci kreslení do rastrové grafické stránky (stroke, fill, show). Pokud není okno na pracovní ploše, je možno otevřít volbou **Output** v menu **View**. V tomto okně se zobrazuje výstup 1:1. Pro rychlou orientaci na grafické stránce pak slouží náhledové okno, které lze otevřít volbou **Output preview** v menu **View**.

7. 10 Jak používat program EMSCRIPT

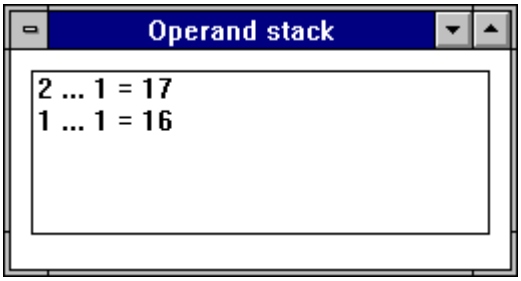
Na několika krátkých a jednoduchých příkladech zde ukáži, jakým způsobem je možno pracovat s programem EMSCRIPT. V záhlaví příkladu vždy bude název souboru, pod kterým jej můžete nalézt na přiložené disketě. Interpretaci zdrojového textu krokujte pomocí klávesy **F8** a držte se pokynů v komentářích na jednotlivých řádcích.

Příklad 1 (DEMO1.PS)

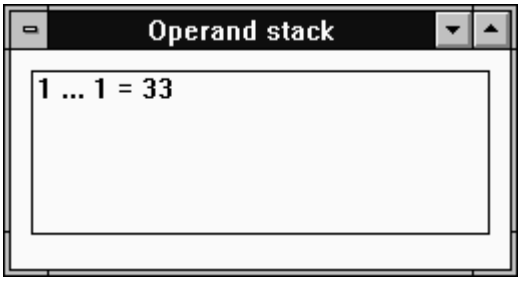
```
% Součet dvou čísel

16              % Otevřete okno zásobníku operandů. Na zásobníku je nyní jeden objekt typu číslo.
```




```
17              % Do zásobníku operandů se vloží číslo 17.
```

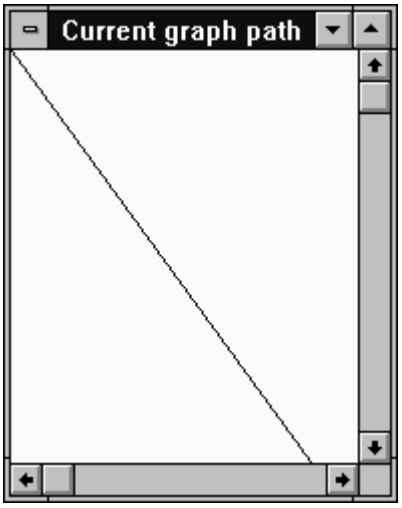



```
add             % Provede se součet dvou čísel na vrcholu zásobníku
                % a uloží se na zásobník operandů.
```



Příklad 2 (DEMO2.PS)

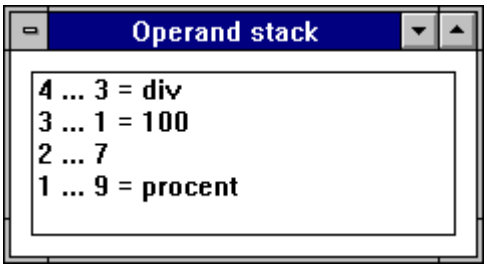
```
% Nakreslení úsečky

0 792 moveto

576 0 lineto        % Otevřete okno grafické cesty. Černou čarou se zde
                    % zobrazuje grafická cesta. V tomto
                    % případě vidíte úsečku s krajními body [0;792]-[576;0].
```




```
stroke              % Úsečka se vykreslí do okna grafické stránky.
```

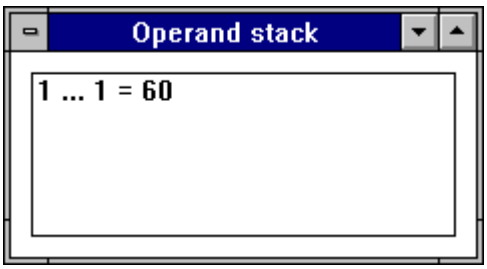
Příklad 3 (DEMO3.PS)

```
% Definice procedury

/procent            % V okně zásobníku se objeví jmenný objekt "procent".

{                   % Na zásobník se uloží značka počátku definice procedury.
                    % Otevřete okno procedurálního zásobníku. Na tomto zásobníku je nyní uložen
                    % příznak definice procedury.

100                 % Na zásobník operandů se uloží číslo 100.

div                 % Zde je vidět rozdíl mezi definicí procedury a přímým během programu.
                    % Jmenný objekt se nevykoná, ale uloží se na zásobník operandů.
```

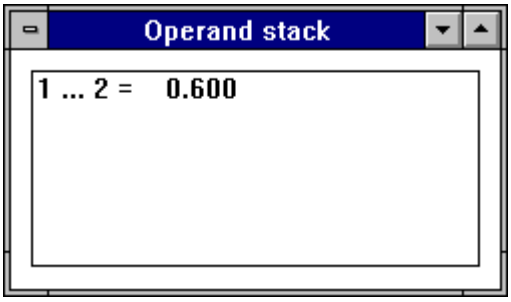



```
}                   % Po vykonání tohoto řádku se místo definice procedury objeví objekt typu
procedura.
                    % Z procedurálního zásobníku se vybere příznak definice procedury.

def                 % Procedura se uloží do slovníku. Zásobník operandů se vyprázdní.
```

60 % Na zásobník operandů se uloží číslo 60.

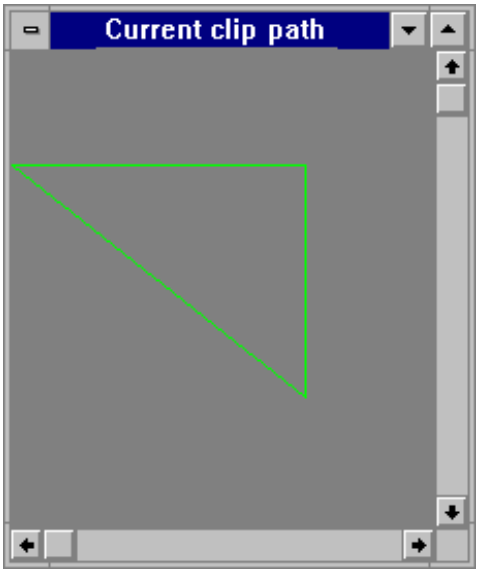


procent % Vykoná se procedura *procent*, která uloží
% na zásobník operandů číslo 0.6 (60/100).

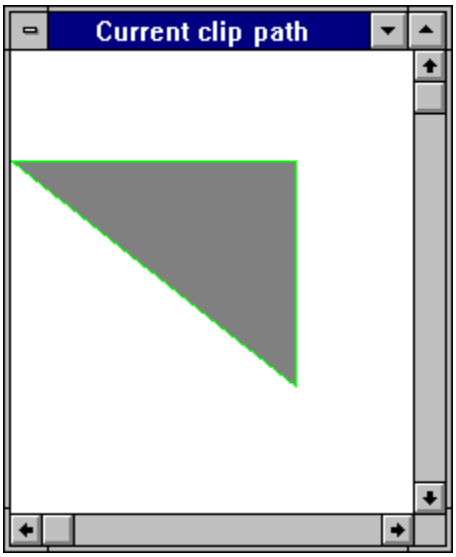


Příklad 4 (DEMO4.PS)

```
% Grafická cesta ořezu

/cm {72 mul 2.54 div}      % Definice procedury, která umožní používat jako jednotky centimetry.

newpath                    % Inicializace grafické cesty.

0 26 cm moveto

5 cm 26 cm lineto

5 cm 22 cm lineto

closepath

% Otevřete okno grafické cesty ořezu. V tomto okně je černou plochou znázorněn region,
% do kterého je možno zakreslovat a zelenou linkou je zobrazena aktuální grafická cesta.
```



clip % Po vykonání tohoto řádku se změní tvar černé plochy,je tak znázorněna
% nová cesta ořezu.



newpath

1 cm 25 cm moveto

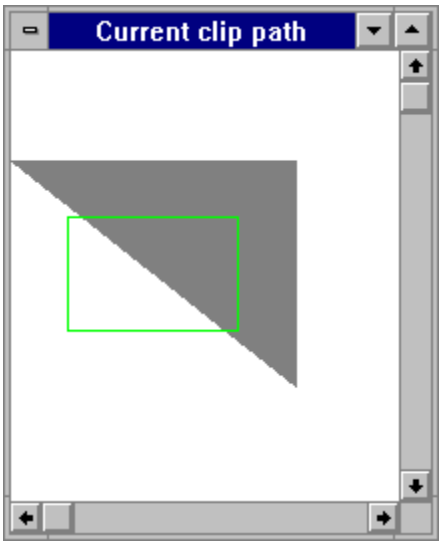
4 cm 25 cm lineto

4 cm 23 cm lineto

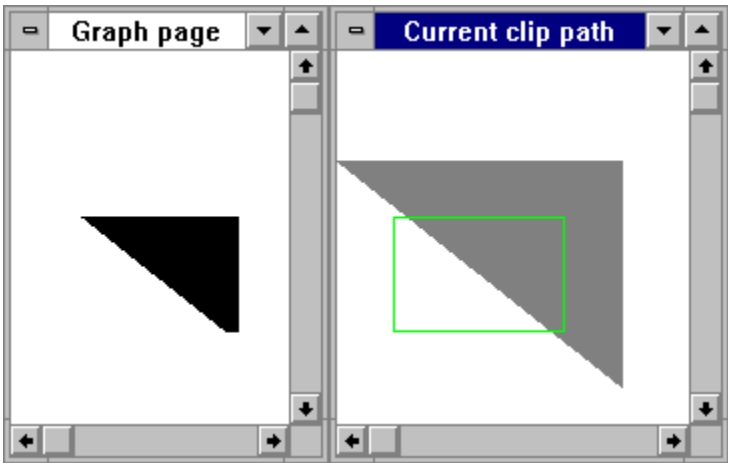
1 cm 23 cm lineto

closepath

% Grafická cesta má nyní tvar čtverce, který je v okně grafické cesty ořezu znázorněn zelenou linkou.



```
fill                    % Vykonáním tohoto řádku se přenese grafická cesta do grafické stránky.
                        % Porovnejte obsahy okna grafické cesty ořezu a okna grafické stránky.
```



7. 11 Vestavěné operátory v interpretu

Současná verze programu EMSCRIPT nepodporuje celou sadu operátorů podle předpisu firmy Adobe Systems Incorporated, ale pouze podmnožinu nefrekventovaněji používaných operátorů. Tato množina dovoluje ilustrovat interpretaci jednodušších příkladů programů v jazyku PostScript. Seznam vestavěných operátorů můžete nalézt v příloze A.

8. Vnitřní struktura programu EMSCRIPT

Program EMSCRIPT je napsán v jazyku Pascal, přičemž bylo použito vývojové prostředí firmy Borland - Borland Pascal 7.0. Program je psán speciálně pro platformu WINDOWS™ a používá pascalské moduly OWL. Při vytváření byla použita metoda objektového a událostmi řízeného programování. Zdrojové texty programu jsou obsaženy v těchto souborech:

MainMDI.pas	obsahuje objekt hlavního okna aplikace a modul řízení interpretace
objkomp.pas	obsahuje modul virtuální paměti a netriviálních objektů
konverze.pas	převod atomických formulí do PostScript objektů
lexikal.pas	lexikální analýza a modul vstupních souborů
grafika.pas	grafický modul
slovniky.pas	modul slovníků
chyby.pas	zpracování chyb
zasobnik.pas	zásobníky
fonty.pas	fonty
device.pas	grafická stránka
about.pas	modul pro dialog O programu
mfilec.pas	konstanty
toolbar.pas	unit pro horní lištu programu
globinf.pas	konstanty
hodnoty.pas	konstanty

basicnum.pas	program pro generování prvočísel, vytvoří soubor *basicnum.dta*
genop.pas	program pro generování seznamu vestavěných operátorů do souboru *sysdict.dta*

8. 1 Struktura interpretu

Interpret je možno rozdělit do několika relativně nezávislých modulů, které spolu navzájem komunikují:

Modul řízení interpetace - zajišťuje kooperaci ostatních modulů, udržuje chod interpretu a zpracovává chybové stavy (viz Modul řízení interpretace).

Modul analýza zdrojového textu - zajišťuje interpretaci zdrojového textu a vytváření odpovídajících objektů v systému (viz Lexikální analýza zdrojového textu).

Modul virtuální paměti - spravuje evidenci o sdílení netriviálních objektů a zajišťuje jejich správné uvolňování z paměti (viz Virtuální paměť).

Modul správy zásobníků - obhospodařuje zásobník operandů a procedurální zásobník. Kontroluje přetečení a vybírání z prázdného zásobníku.

Modul slovníků - zajišťuje vytváření nových objektů typu slovník, ukládání hodnot do slovníku, obhospodařuje slovníkový zásobník a vyhledává konkrétní hodnoty v určitém slovníku nebo ve všech slovnících uložených na slovníkovém zásobníku.

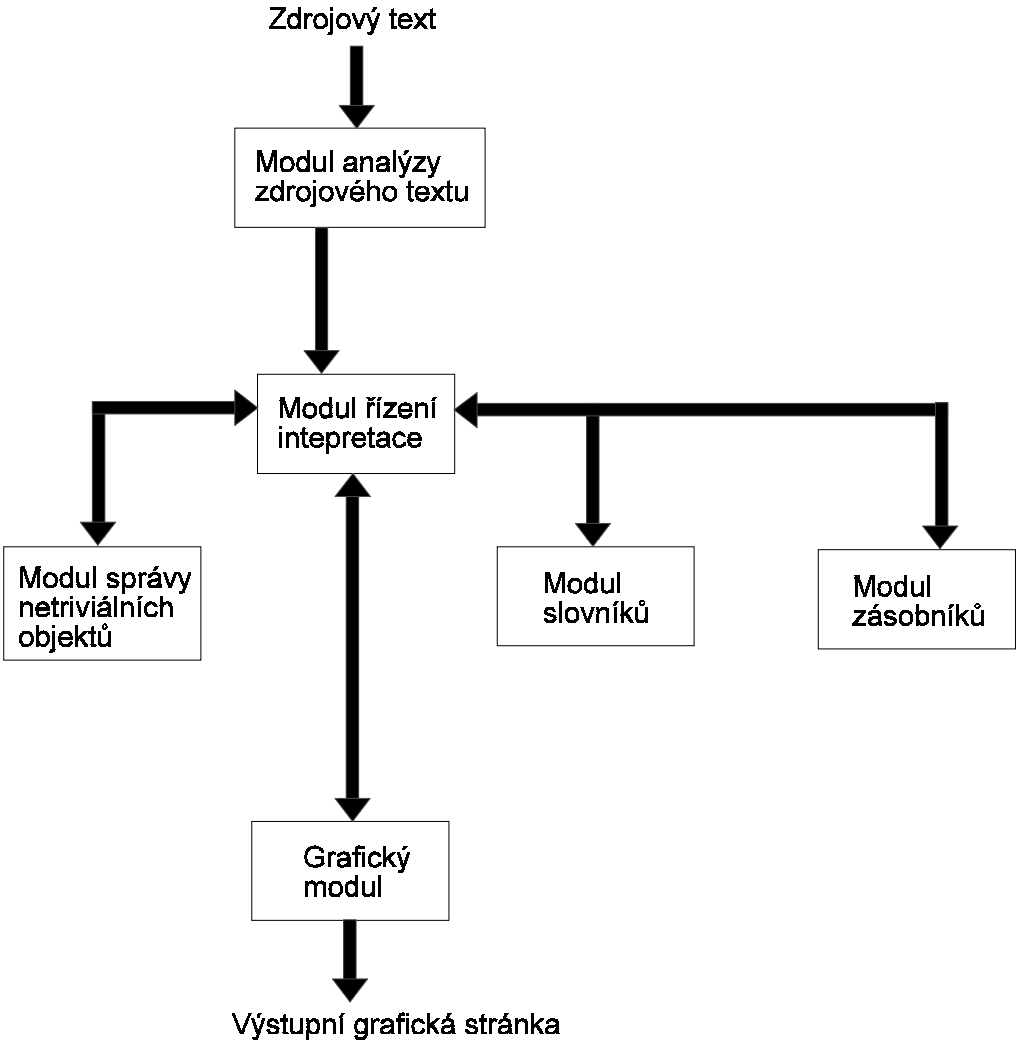
Grafický modul - spravuje grafický stav a grafickou cestu, přidává grafické objekty do grafické cesty, vykresluje grafickou cestu do rastrové výstupní stránky, definuje cestu ořezu a mění atributy v grafickém stavu interpretu.

8. 2 Hlavní okno aplikace

Aplikace interpretu spravuje objekt hlavního okna (MainMDI.pas), který definuje uživatelské okno programu, zpracovává hlášení operačního systému WINDOWS™ , koordinuje činnost a monitoruje aktuální stav interpretu. Významnou metodou je procedura **CMOperace**, která zajišťuje vykonávání jednotlivých příkazů zdrojového textu. Metoda **VykonBlok** zajišťuje provádění implementovaných operátorů, **ZpracovaniChyby** zpracovává chybová hlášení. Další skupina metod zajišťuje ukládání a obnovování aktuálního stavu systému a nastavení systému do počátečního stavu. V tomto objektu jsou též metody, které reagují na požadavky uživatele (volby v menu, horké klávesy), nastavují interpret podle přání uživatele a otevírají náhledová okna na vnitřní struktury interpretu.

8. 3 Modul řízení interpretace

Tento modul je součástí objektu hlavního okna aplikace a tvoří jej metody **CMOperace**, **VykonBlok** a **Zpracování chyby**. Tento modul se na základě stavu procedurálního zásobníku odkazuje buď na modul analýzy zdrojového textu nebo na procedurální zásobník (v případě odloženého vykonávání objektů viz Odložené vykonávání objektů). Zde se vykonávají jednotlivé objekty a ve spolupráci s ostatními moduly se tvoří grafická cesta. Modul vstupní analýzy textu slouží pro vstup programu pro interpretaci. Moduly slovníků, zásobníků a správy netriviálních objektů jsou moduly, které udržují objekty vytvořené na základě interpretace zdrojového textu. Grafický modul zajišťuje tvorbu grafické cesty a její výstup do grafické stránky. Modul řízení funguje jako koordinátor činností všech modulů podle vstupního programu PostScriptu. Tento modul též zahrnuje vykonávání vestavěných operátorů.



8. 4 Virtuální paměť

Všechny netriviální objekty jsou alokovány pomocí vnitřního správce tzv. virtuální paměti: VirtMemory. Tento správce provádí veškerou agendu o počtu sdílení a možnosti uvolnění objektu z paměti. Netriviální objekty jsou všechny objekty, jejichž velikost nedovoluje uložení v běžné datové položce TPolozka používané v zásobníku, slovníku, poli a procedurálním poli. V datová struktura TPolozka obsahuje atribut Typ nesoucí informaci o typu uloženého objektu. V případě netriviálního objektu je v TPolozka uložen index buňky virtuální paměti, kde se nachází paměťový ukazatel na instanci netriviálního objektu.

Všechny netriviální objekty jsou odvozeny z abstraktního objektu TCompObject, který obsahuje čítač sdílení s příslušnými metodami inkrementace a dekrementace. V čítači sdílení je uložen počet odkazů (z jiného objektu, např. pole, slovníku, nebo ze zásobníku operandů), které se na tento objekt odkazují. Stav čítače je pak rozhodující při uvolňování tohoto objektu z paměti. Interpret pracuje tak, že informace o všech netriviálních objektech jsou obsaženy v aktuálním objektu virtuální paměti. Na konci každého cyklu interpretu provede virtuální paměť rozbor netriviálních objektů a objekty, které vykazují počet sdílení 0, uvolní z paměti. S čítačem v netriviálním objektu se manipuluje v případě následujících operací:

Objekt	Metoda	Čítač sdílení
Zásobník operandů	Vlož Vyber	+ -
Procedurální zásobník	Vlož Vyber	+ -
Slovníkový zásobník	StoreDirect Vlož Vyber	+ **nový element** - **původní element** + -
Slovník	PutPair Done	+ **nový element** - **původní element** -
Pole, procedurální pole	Vlož PlusPrvek Done	+ **nový element** - **původní element** + **nový element** -
SubPole, SubŘetězec	Init Done	+ -

8. 4 .1 Objekt TVirtMemory

Objekt TVirtMemory je konkrétní objekt vytvořený pro správu netriviálních objektů.

8. 4 .1 .1 Atributy

VM : PCollection - pole ukazatelů na instance netriviálních objektů. MemDescr hodnoty, které vystupují jako virtuální ukazatele na objekty, jsou indexy tohoto pole.

8. 4 .1 .2 Metody

function Alloc(Obj : PCompObject; var MemDescr : TMemDescr):bool - zařadí netriviální objekt do evidence virtuální paměti. Pro správnou činnost interpretu se toto zařazení provádí pro všechny vytvořené netriviální objekty. ***Pozámka:*** Typ **bool** je 16 bitový typ analogický boolean.

function GetPointer(MemDescr : TMemDescr):PCompObject - podle hodnoty MemDescr vrací adresu na instanci netriviálního objektu.

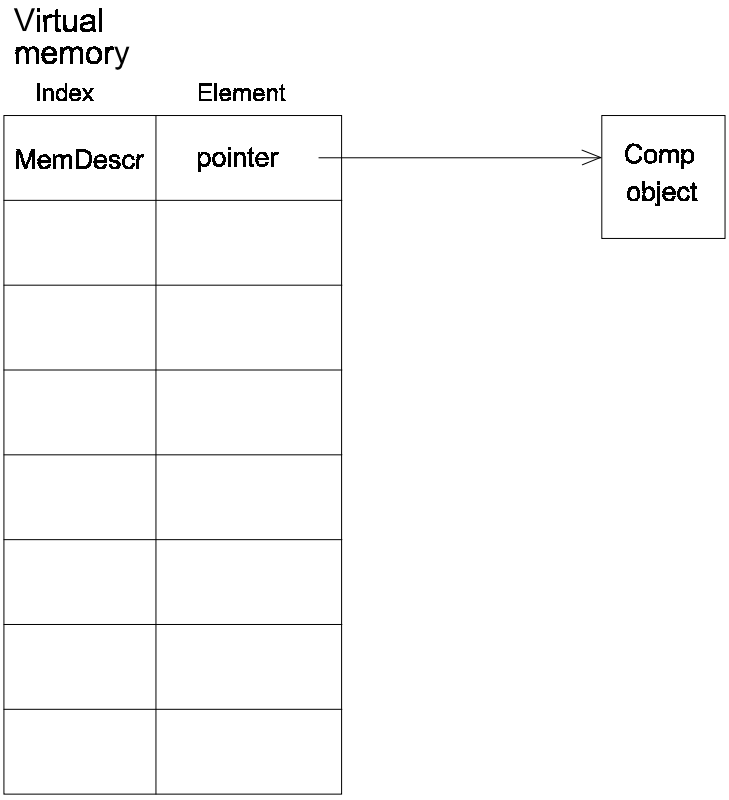
function GetMemDescr(APoint : PCompObject):TMemDescr - podle adresy instance netriviální adresy nalezne odpovídající MemDescr.

procedure Free - uvolní všechny instance netriviálních objektů, které mají čítač sdílení roven 0.

function Copy - tato metoda umožňuje vytvořit záložní kopii celé virtuální paměti. Vytvoří se nejen kopie instance objektu virtuální paměti, ale i všech netriviálních objektů, které jsou zařazeny do správy VM. Je určena pro použití operátoru **save.**

procedure Inc(MemDescr : TMemDescr) - zvyšuje počet sdílení v netriviálním objektu, na který odkazuje MemDescr.

procedure Dec(MemDescr : TMemDescr) - snižuje počet sdílení v netriviálním objektu, na který odkazuje MemDescr.



8. 4 .2 Abstraktní objekt TCompObject

Tento objekt zavádí manipulaci s čítačem sdílení. TCompObject je předkem všech netriviálních objektů.

Každý netriviální objekt má metodu **Copy**, která vytvoří kopii instance netriviálního objektu. Tato metoda se používá k vytváření záložní kopie celého obsahu virtuální paměti (viz TVirtMemory.Copy). Metoda je virtuální, to znamená, že metoda Copy každého objektu se postará o svůj konkrétní typ objektu a zařídí vše potřebné pro jeho úspěšné zkopírování a zkopírování všech objektů, na které se odkazuje. Pokud není u zděděných objektů uvedeno jinak, pak mají standardní metody: constructor Init, virtuální destructor Done() a metodu kopírování function Copy(VMemory : PVirtMemory):PCompObjekt.

8. 4 .2 .1 Atributy

COTyp : word - v tomto atributu se uchovává informace o typu zděděného objektu:
sco_NotIdentified = 0, netriviální objekt bez identifikace
sco_String = 1, řetězec
sco_Substring = 2, podřetězec
sco_Array = 3, pole
sco_Subarray = 4, podpole
sco_Matrix = 5, matice
sco_Dictionary = 6, slovník
sco_GraphState = 7, grafický stav
sco_GraphPath = 8, grafická cesta
sco_Font = 9, font

Kolik : word - čítač sdílení, počáteční hodnota je 0. Podle manipulací s objektem, uvedených v předchozí tabulce, se tento čítač nastavuje a nese informaci o počtu sdílení tohoto objektu.

VMem : Pobject - ukazatel objekt virtuální paměti, která obhospodařuje tento objekt.

8. 4 .2 .2 Metody

procedure Pridej - zvýší hodnotu čítače sdílení

procedure Uber - sníží hodnotu čítače sdílení

8. 5 Lexikální analýza zdrojového textu

Emulátor nepracuje v klasickém terminálovém režimu. Pro vstup příkazů se používá externích souborů uložených na disku. Vstupní soubory mohou být vykonávány ve dvou módech: mód bez náhledu na zdrojový text. Tento mód je určen pro zpracování zdrojových textů příkazem **run**. Tento mód není v této verzi programu využit. Druhý mód umožňuje nahlížet na interpretované zdrojové texty pomocí náhledového okna. Je tím možno sledovat stav systému po vykonání každé řádky zdrojového textu. Zdrojové texty je možno v náhledovém okně editovat i v průběhu vykonávání programu.

8. 5 .1 Object TFile (unit LEXIKAL.PAS)

Vstupní soubor zdrojového textu bez možnosti jeho monitorování.

8. 5 .1 .1 Atributy:

EDF : word - identifikace Tfile a TEditFile.

Jmeno : array [0..40] of char - jméno souboru se zdrojovým textem.

TextBuff : array [0..255] of char - vstupní buffer (soubor čte tento objekt po řádcích a jednotlivé znaky přenáší přes tento buffer).

Konec : boolean - indikace konce souboru.

FileDescr : TDOSStream - objekt obhospodařující systém čtení souboru (součást API WINDOWS™).

8. 5 .1 .2 Metody:

function CtiZnak(var Znak : char):char - čte jeden znak ze vstupního souboru

procedure VratZnak(var Znak : char) - vrací znak do vstupního souboru, tato procedura se používá, pokud je přečtený znak současně oddělovačem čtené atomické formule a začátkem následující atomické formule.

procedure VstupOp(var Typ : byte; var Atom : PChar) - přečte atomickou formuli ze vstupního textu, v Typ vrací typ objektu, jaký tato formule popisuje. Text atomické formule je uložen v Atom.

8. 5 .2 Object TEditFile (unit LEXIKAL.PAS)

Vstupní soubor zdrojového textu s možností jeho monitorování. Oproti objektu TFile umožňuje navíc prohlížet, editovat a krokovat zdrojový text

8. 5 .2 .1 Atributy:

EdF : word - identifikace Tfile a TEditFile.

Jmeno : array [0..40] of char - jméno souboru se zdrojovým textem.

TextBuff : array [0..255] of char - vstupní buffer (soubor čte tento objekt po řádcích a jednotlivé znaky přenáší přes tento buffer).

Konec : boolean - indikace konce souboru.

FileDescr : TDOSStream - objekt obhospodařující systém čtení souboru (součást API WINDOWS™).

TextWindowFile : PFileWindow - ukazatel na objekt obhospodařující editační okno.

DebugSign, FirstF8, IsNewLine - atributy zajišťující krokování zdrojového textu.

Line : integer - číslo právě provedené řádky při krokování.

8. 5 .2 .2 Metody:

function CtiZnak(var Znak : char):char - čte jeden znak ze vstupního souboru

procedure VratZnak(var Znak : char) - vrací znak do vstupního souboru, tato procedura se používá, pokud je přečtený znak současně oddělovačem čtené atomické formule a začátkem následující atomické formule.

procedure VstupOp(var Typ : byte; var Atom : PChar) - přečte atomickou formuli ze vstupního textu, v Typ vrací typ objektu, jaký tato formule popisuje. Text atomické formule je uložen v Atom.

procedure MakeTextWindow - vytvoří náhledové editační okno, které zobrazuje zdrojový text.

procedure SelectLine(Line : integer) - v editačním okně označí řádku ve zdrojovém textu.

8. 6 Objekty Zasobnik a BehZasobnik (unit ZASOBNIK.PAS)

Tyto objekty jsou určeny pro správu zásobníku operandů a procedurálního zásobníku. Do zásobníku je možno vkládat a opětně z něj vybírat (LIFO - *last in first out*). Zásobník má automatickou kontrolu přetečení nebo podtečení. Při těchto chybových stavech se vyvolává metoda indikace chyby v objektu **ErrorHandle**. Ukládání a vybírání automaticky zvětšuje a zmenšuje čítač sdílení vkládaných a vybíraných netriviálních objektů.

8. 6 .1 Atributy:

VMem : PVirtMemory - odkaz na virtuální paměť, kam přísluší uložené netriviální objekty.

AktPoz : word - ukazuje na aktuální pozici vrcholu zásobníku.

POBox : array [0..500] of TPolZas *pro Zasobnik* a array [0..250] of TPolBehZas *pro BehZasobnik* - pole pro ukládání vkládaných prvků.

8. 6 .2 Metody:

function Vloz(Pol : TPolZas):bool - uloží na zásobník operand Pol, v případě úspěšného uložení vrací ***true.***

function Vyber(var Pol : TPolZas):bool - vybere z vrcholu zásobníku operand a vrací jej v parametru Pol. V případě bezchybného vybrání ze zásobníku vrací tato metoda ***true***.

8. 7 Objekt Slovník (unit SLOVNIKY.PAS)

Objekt slovník zajišťuje ukládání a vyhledávání asociovaných dvojic PostScriptových objektů.

8. 7 .1 Atributy:

Kapacita : word maximální povolený počet párů v tabulce slovníku.

Pocet : word aktuální počet párů v tabulce.

Veltab : word skutečná velikost tabulky (velikost tabulky je prvočíslo přibližně o 10% větší než je kapacita. Je to z důvodu použití vyhledávací metody hashování v tabulce slovníku).

Ukaz : pointer ukazatel na obsah tabulky. Tabulka se skládá z ukazatelů na instance párů.

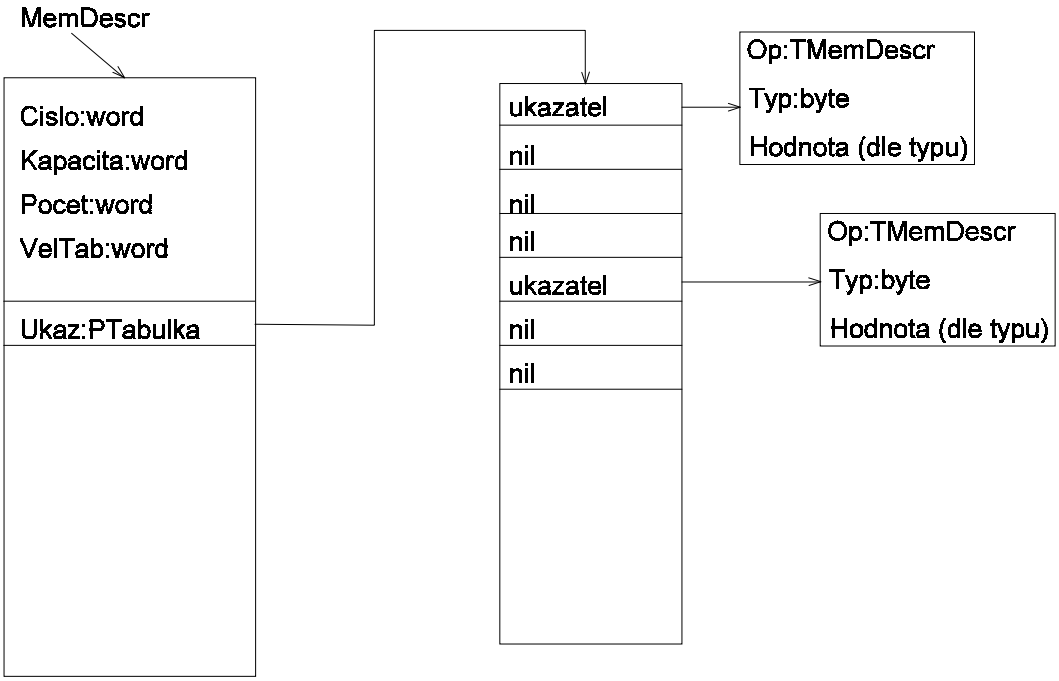
Instance páru je typu Tpolozka=record
Op : TMemDescr; ukazuje na objekt jména klíče.
Typ : byte; určuje typ uloženého objektu.
Hodnota : THodnota; obsahuje hodnotu uloženého objektu.
end

Pro vkládání a získávání objektů v tabulce slovníku se používá metod PutPair a GetPair.

8. 7 .2 Metody:

function PutPair(var Polozka : TPolozka):bool vloží pár do slovníku. V parametru Polozka musí být vyplněny všechny hodnoty páru. Metoda provede vyhledání ve slovníkové tabulce. Pokud již klíč v tabulce existuje, pak se pouze zamění přiřazená hodnota. Jinak se vytvoří nová instance Tpolozka, do té se vloží obsah parametru Polozka a do tabulky se na příslušné místo zapíše ukazatel na tuto instanci. Metoda vrací false, když je slovník plný, jinak vrací true.

function GetPair(var Polozka : TPolozka; var Index : word):bool nalezne pár uložený v tabulce. Do Polozka.Op je třeba vložit klíč, podle kterého se bude ve slovníku vyhledávat. Pokud hledaný klíč v tabulce existuje, pak metoda vrací true a naplní hodnoty v Polozka a Index (Index je skutečný index v tabulce slovníku). Jestliže klíč nebyl nalezen, pak se parametry metody nevyplňují a metoda vrací false.



8. 7 .3 Vyhledávání v tabulce slovníku

V tabulce slovníku je pro vyhledání klíče zvolena metoda hashování. Pro rovnoměrné rozmisťování klíčů je velikost tabulky prvočíslo. Pro snížení možného počtu kolizí je velikost tabulky o 10% větší než je kapacita párů v tabulce. První index vyhledání se získá součtem jednotlivých znaků v řetězci klíče (funkce H0 v unit Slovniky.pas). Ošetření kolize je prováděno metodou kvadratických pokusů (podle [11]):

$h_0=H(K)$
$h_i=(h_0+i^2) \bmod N \ (i>0)$

Pokud se použijí rekurentní vztahy pro $hi=i^2$ a $di=2i+1$, tak pro výpočet dalšího indexu není třeba použít operaci umocnění:

$h_{i+1}=h_i+d_i$
$d_{i+1}=d_i+2 \ (i>0)$

SystemDict má pevně danou strukturu. To umožňuje tuto tabulku vygenerovat pouze jednou a její strukturu uložit v parametrickém souboru. Program pro generování parametrického souboru se jmenuje GenOp a parametrický soubor má jméno Sysdict.DTA.

UserDict má limit 200, velikost tabulky je tedy vypočtena na 223.

Velikost tabulky se určí podle vygenerovaného seznamu prvočísel (program BasicNum) uloženého v parametrickém souboru Basicnum.DTA. Soubor obsahuje vzestupně setříděná prvočísla menší než 16384. V tomto souboru se vhodné prvočíslo vyhledá metodou půlení intervalu (funkce Velikost v unit Slovniky.pas) maximálně po 14 krocích.

8. 8 Objekt SlovZas (unit SLOVNIKY.PAS)

Slovníky, ve kterých bude interpret vyhledávat, jsou udržovány na slovníkovém zásobníku. Tento zásobník má hloubku 20. Na rozdíl od zásobníku operandů je tento zásobník poněkud složitější, neboť mimo ukládání a vybírání musí zajišťovat prohledávání všech slovníků na zásobníku. Na dně se nachází systémový slovník **systemdict** a nad ním uživatelský slovník **userdict**. Ani jeden z těchto slovníků nelze ze zásobníku vybrat. V **systemdict** jsou uložena všechna jména vestavěných operátorů a k nim příslušné objekty. Zásobník kontroluje přetečení a podtečení. Při vkládání a vybírání slovníku se automaticky zvyšuje a snižuje čítač sdílení vkládaných slovníků. Při vkládání netriviálních objektů do slovníku uloženém na zásobníku se zvyšuje čítač sdílení těchto objektů.

8. 8 .1 Atributy:

VMem : PVirtMemory - odkaz na virtuální paměť, která přísluší k uloženým slovníkům.

SlovZas : array [1..20] of TMemDescr - pole pro uložení odkazů na slovníky.

SlovZasTop : byte - ukazatel na vrchol zásobníku.

SlovZasAkt : byte - ukazatel používaný pro procházení slovníků v zásobníku při vyhledávání klíče.

SlovZasSystem : byte - ukazatel do zásobníku na místo, kde leží systemdict.

SlovZasUser : byte - ukazatel do zásobníku na místo, kde leží userdict.

8. 8 .2 Metody:

function Vloz(Slovnik : TMemDescr):bool - vloží na vrchol zásobníku nový slovník (ukazatel Slovnik). V případě, že proběhla v pořádku, vrací metoda hodnotu ***true***.

function Vyber : bool - vybere slovník na vrcholu zásobníku. Pokud metoda proběhne v pořádku, pak vrací ***true***.

function Put(SlovPor : byte; Polozka : TPolozka):bool - vyvolá metodu **PutPair** slovníku, který se nachází na pozici SlovPor v zásobníku. Pokud vložení do slovníku proběhne v pořádku vrací metoda ***true***.

function Get(SlovPor : byte; var Polozka : TPolozka; var PolIndex : word):bool - v konkrétním slovníku vybere uloženou hodnotu podle klíče. Pole Op záznamu Polozka naplníte řetězcem, obsahujícím klíč, podle kterého hledáte, do SlovPor pak pořadí konkrétního slovníku na zásobníku. V případě, že je klíč nalezen, pak metoda vrací v druhé části parametru Polozka hodnotu příslušející tomuto klíči a v parametru PolIndex vrací index nalezeného páru ve slovníku. Metoda vrací ***true***, pokud byl klíč nalezen.

function Find(var Polozka : TPolozka; var SlovPor : word; var PolINdex : word):bool - metoda prohledá postupně od vrcholu zásobníky dokud nenajde odpovídající klíč. Pole Op parametru Polozka naplňte hledaným klíčem. V případě, že v některém ze slovníků, uložených na zásobníku, bude klíč nalezen, pak se metoda vrátí v druhé části parametru Polozka nalezenou hodnotu, v SlovPor pořadí odpovídajícího slovníku v zásobníku a v PolIndex index uloženého páru uvnitř slovníku. Metoda při úspěšném nalezení klíče vrací ***true***.

function Where(var Polozka : TPolozka; var Slovnik : TMemDescr; var PolIndex : Word):bool - pracuje stejně jako metoda **Find**, ale vrací místo pořadí v zásobníku přímo odkaz na konkrétní slovník.

function StoreDirect(SlovID : word; PolIndex : word; Polozka : TPolozka):bool - vloží obsah parametru Polozka na místo s indexem PolIndex do slovníku s identifikačním číslem SlovID. Při svém vzniku dostane každý slovník jedinečné číslo, podle kterého lze slovník jednoznačně identifikovat, toho se potom využívá v této metodě. V případě korektního uložení hodnoty vrací metoda ***true***.

8. 9 Správa řetězců, polí a procedurálních polí

8. 9 .1 Objekt Ret (unit OBJKOMP.PAS)

Tento objekt slouží pro uložení textového řetězce.

8. 9 .1 .1 Atributy:

Count : word - velikost uloženého řetězce.

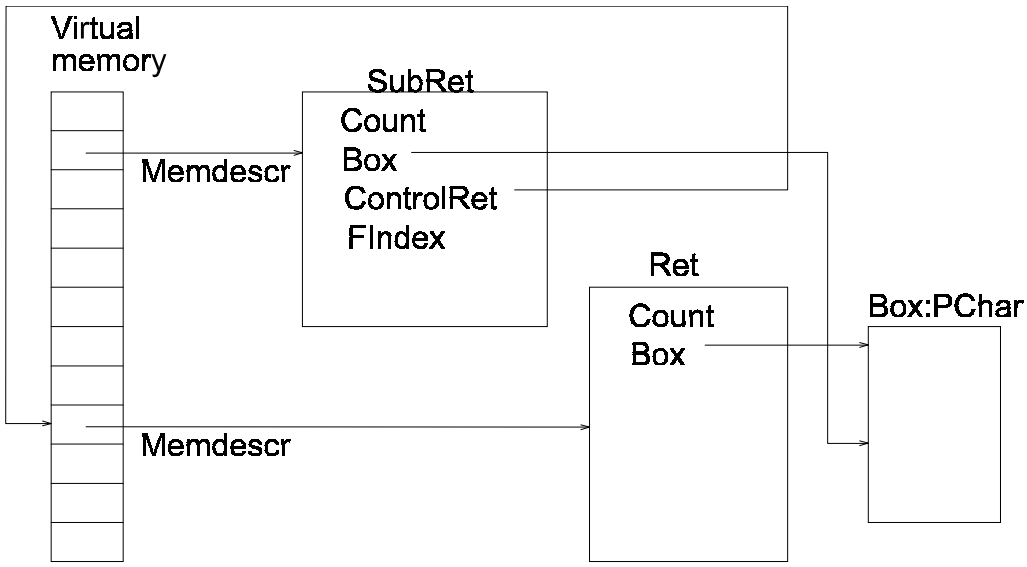
Box : PChar - alokovaná paměť, kde je uložen textový řetězec ukončený 0.

8. 9 .1 .2 Metody:

constructor Init(VMemory : PVirtMemory; XRet : PChar; RCount : word) - naplní atribut Box textovým řetězcem XRet o délce RCount.

8. 9 .2 Objekt SubRet (unit OBJKOMP.PAS)

Hodnoty komplikovaných objektů jsou sdílené, tzn. že můžete nad objektem řetězce vytvořit novou kopii řetězce nebo část řetězce. Tato kopie se tváří jako samostatný objekt, ale samotný textový řetězec je sdílený. Pokud tedy změníte obsah jednoho objektu, pak se automaticky změní obsah všech objektů, které byly vytvořeny jako kopie. Z toho důvodu se zavádí interní pomocný objekt SubRet, který se navenek tváří jako objekt typu Ret, ale pracuje na sdíleném řetězci.



8. 9 .2 .1 Atributy:

Count : word - počet znaků v řetězci.

Box : PChar - ukazatel do sdíleného řetězce.

ControlRet : TMemDescr - odkaz na originální objekt řetězce. Podle tohoto odkazu se zvyšuje a snižuje čítač sdílení u originálního objektu řetězce. Při změně čítače sdílení objektu SubRet se automaticky mění i čítač sdílení originálního objektu Ret.

FIndex : word - index prvního znaku ve sdíleném řetězci.

8. 9 .2 .2 Metody:

constructor Init(VMemory : PVirtMemory; BasicRet : TMemDescr; FirstIndex, CountSub : word) - nastaví Box na konkrétní adresu v originálním objektu podle FirstIndex.

function StartPoint : pointer - vrací adresu začátku sdíleného podřetězce.

8. 9 .3 Objekt Proc (OBJKOMP.PAS)

Tento objekt je společný pro pole a pro procedurální pole.

8. 9 .3 .1 Atributy:

Count : longint - velikost pole (podle této hodnoty je určena velikost paměťového bloku, na který ukazuje ProcBox.

ProcBox : PProcBox - ukazatel na pole TProcBox = array [0..500] of TPolProc, kde jsou uloženy jednotlivé prvky pole Typ TProcBox je abstraktní a velikost skutečně alokovaného pole je odvozena z hodnoty Count. Pro procedurální pole má zvláštní význam první položka pole longint(ProcBox^[0].Uk), která určuje jak je pole „naplněné“, tzn. že pole nemusí být zcela vyplněno až do počtu Count. Podle této hodnoty se řídí výkonný blok při vykonávání procedury.

8. 9 .3 .2 Metody:

constructor Init(VMemory : PVirtMemory; Pocet : longint) - VMemory je ukazatel na objekt virtuální paměti, ke kterému se vztahují všechny uložené objekty. Pocet určuje velikost pole

procedure PlusPrvek(XPrvek : TPolProc) - na konec pole přidá další objekt, přičemž zvyšuje hodnotu ProcBox^[0].Uk a tím dynamicky rozšiřuje pole (maximálně do hodnoty Count). Pokud je vkládaný objekt netriviální, pak zvyšuje jeho čítač sdílení.

procedure VratPrvek(var Index : longint; var Prvek : TPolProc) - do Prvek dosadí objekt, který se v poli nachází v buňce s indexem Index. Index se automaticky posunuje k následující buňce. Tato metoda se používá při vykonávání procedurálního pole.

procedure Vloz(Index : longint; Prvek : TPolProc) - do buňky s indexem Index vloží objekt Prvek. Pokud je již uložený objekt netriviální, pak sníží hodnotu jeho čítače sdílení (tento objekt bude v poli nahrazen vkládaným). Pokud je netriviální vkládaný objekt, pak zvýší hodnotu jeho čítače sdílení. Pozn.: Při uvolnění objektu pole se automaticky sníží čítač sdílení všech v něm uložených netriviálních objektů.

procedure Element(Index : longint; var Prvek : TPolProc) - do Prvek uloží objekt z buňky Index. Tato metoda se používá v obyčejném poli.

function Size : longint - vrací velikost pole (podle longint(ProcBox^[0].Uk))

function KonecProcedury(Index : longint):bool - v procedurálním poli vrací hodnotu ***true***, když jsou vykonány všechny procedury. Tato metoda se používá v kombinace s metodou **VratPrvek**, která krokuje přes procedurální pole.

8. 9 .4 Objekt SubArray (unit OBJKOMP.PAS)

Tento objekt je určen pouze pro obyčejné pole. Je zděděný z **Proc**. Obsluha tohoto objektu a princip sdílení hodnot s originálním polem je analogický objektu **SubRet**.

8. 9 .4 .1 Atributy:

Count : longint - kapacita paměťového bloku určeného pro sdílené pole, na který ukazuje ProcBox. Pokud sílená oblast nezačíná na začátku původního paměťového bloku je velikost o to menší.

ProcBox : PProcBox - ukazatel do sdíleného paměťového bloku TProcBox = array [0..500] of TPolProc, kde jsou uloženy jednotlivé prvky pole.

BasArray : TMemDescr - ukazatel na originální objekt **Proc**, který vytvořil sdílený paměťový blok.

FIndex : word - první index ve sdíleném poli. Toto pole se používá pro určení začátku sdílené oblasti v paměťovém bloku příslušejícímu BasArray.

8. 9 .4 .2 Metody:

constructor Init(VMemory : PVirtMemory; BasicArray : TMemDescr; FirstIndex, CountSub : Word) - BasicArray je odkaz na originální objekt, který vytvořil sdílený paměťový blok a který spravuje jeho alokaci i uvolnění v paměti. FirstIndex určuje posunutí začátku sdíleného pole vůči začátku paměťového bloku. CountSub je počet prvků sdíleného pole. (Tak je možno sdílet pouze určitý interval původního pole.) **Init** automaticky zvyšuje čítač sdílení objektu BasArray a tím si zajistí, že originální objekt nebude uvolněn z paměti, dokud se na něj bude odkazovat.

destructor Done - snižuje hodnotu čítače sdílení objektu BasArray. Tím naznačuje, že se na originální objekt pole již dále nebude odkazovat.

function StartPoint : pointer - vrací adresu začátku sdílené oblasti určené podle FIndex.

8. 10 Objekt Matrix (unit OBJKOMP.PAS)

Objekt Matrix se používá pro transformační matice, jedná se o pole 6 hodnot XX, XY, XT, YX, YY, YT, které jednoznačně určují rovinnou transformační matici v homogenních souřadnicích. Tento objekt je používán pro všechny druhy implementovaných transformací.

8. 10 .1 Atributy:

XY, XT, YX, YY, YT : TReal - určují obsah transformační matice:

XX	XY	XT
YX	YY	YT
0	0	1

8. 10 .2 Metody:

procedure NewValue(AXX, AXY, AXT, AYX, AYY, AYT : TReal) - při vytvoření matice nastaví constructor Init hodnoty tak, aby se jednalo o jednotkovou matice. Hodnoty v matici lze měnit touto metodou (metoda dovoluje modifikaci buněk, které jsou používány pro rotaci, posun a změnu měřítka).

8. 10 .3 Použití:

V unit GRAFIKA.PAS se nacházejí rutiny, které modifikují transformační matice

procedure Rotate(CTM : PMatrix; Angle : TReal)

procedure Translate(CTM : PMatrix; X, Y : TReal)

procedure Scale(CTM : PMatrix; SX, SY : TReal)

které převádějí uživatelské souřadnice na souřadnice zařízení

procedure TransCoord(CTM : PMatrix; XUser, YUser : TReal; var XDevice, YDevice : TReal)

a naopak

procedure InvertTransCoord(CTM : PMatrix; XDevice, YDevice : TReal; var XUser, YUser : TReal)

8. 11 Objekt GraphState (unit GRAFIKA.PAS)

Objekt GraphState zajišťuje správu aktuálního grafického stavu.

8. 11 .1 Atributy:

CTM : TMemDescr - odkaz na objekt aktuální transformační matici.

Color : TColorRef - aktuální barva vykreslování. Pro určení barvy se používá popis používaný ve Windows™.

PosX, PosY : TReal - aktuální pozice pomyslného kurzoru.

Path : TMemDescr - odkaz na objekt aktuální grafické cesty.

ClipPath : TMemDescr - odkaz na objekt aktuální cesty ořezu.

ClipRgn : HRgn - pro vykreslování a ořezávání se používají standardní služby poskytované GDI ve Windows™, tento atribut se používá k uložení handle na region ořezu vytvořený pomocí GDI služeb.

LWitdh : longint - šířka čáry pro **stroke.**

Lcap : integer - způsob zakončení čáry při **stroke**. Pro vykreslování čáry je použito standardních služeb GDI, kde je pouze jeden typ zakončení (round cap). Tento atribut není v současném systému použit.

Ljoin : integer - způsob spojení čar. Pro vykreslování čáry je použito standardních služeb GDI, kde je pouze jeden typ spojení (round join). Tento atribut není v současném systému použit.

HalfScreen : pointer -způsob tónování. Pro vykreslování je použito standardních služeb GDI, kde je způsob tónování určen ovladačí výstupních zařízení. Tento atribut není třeba používat.

Transfer : PProc - procedura pro převod uživatelského tónování do tónování výstupního zařízení. Tento atribut není třeba. Převod obstarávají ovladače Windows™.

Flatness : TReal - přesnost aproximace bézierových křivek úsečkami.

Mitter limit : TReal - limit pro spojení čar typem mitter join. Současný systém používá GDI služeb s typem spojení round join, proto není tento atribut v současném systému používán.

Dash pattern : pointer - způsob čerchování. Není v současném systému podporován.

Device : pointer - odkaz na objekt výstupního zařízení.

8. 11 .2 Metody:

Kromě standardních metod objektu TCompObject je definována metoda DefClipPath, která převádí analytický popis grafické cestu ořezu na region ořezu ve Windows™. Výsledkem je region definovaný uvnitř Windows™, jehož handle je pro další použití uložen v atributu ClipRgn.

8. 12 Objekt GraphPath (unit GRAFIKA.PAS)

8. 12 .1 Atributy:

GraphState : TMemDescr - odkaz na objekt grafického stravu, který přísluší této grafické cestě.

MemBlock : pointer - ukazatel na začátek řetězce paměťových bloků, kam se ukládá analytický popis grafické cesty.

CurrMemBlock : pointer - pomocný ukazatel používaný při procházení řetězce paměťových bloků popisu.

CurrIndex : word - ukazatel v rámci jednoho paměťového bloku.

StartPoint : TGrPoint - počáteční bod grafické podcesty.

DefCurrPoint : boolean - indikátor definovaného aktuálního bodu grafické cesty (po **newpath** není aktuální bod definován).

CurrentPoint : TGrPoint - souřadnice aktuálního bodu.

ViewCurrPath : boolean - pro vykreslování grafické cesty do rastrové grafické stránky a pro zobrazování aktuální grafické cesty v OKNĚ AKTUÁLNÍ GRAFICKÉ CESTY se používají stejné procedury. V prvním případě je nutno respektovat aktuální cestu ořezu, kdežto v druhém případě je to nežádoucí. Hodnota ViewCurrPath indikuje zda se jedná o zobrazení v OKNĚ AKTUÁLNÍ GRAFICKÉ CESTY.

8. 12 .2 Metody pro tvorbu grafické cesty:

procedure InsertMove(X,Y:TReal) přidá do analytického popisu grafické cesty posun na novou souřadnici. Zároveň změní hodnotu CurrentPoint a StartPoint.

procedure InsertLine(X,Y:TReal) přidá úsečku z aktuálního bodu do bodu [X, Y] a změní souřadnice aktuálního bodu na X a Y. Do analytického popisu se zaznamenává pouze koncový bod, počáteční bod je při vykreslování dán hodnotou CurrentPoint.

procedure InsertClosePath přidá informaci o spojení do počátku aktuální grafické podcesty (propojení CurrentPoint do StartPoint).

procedure InsertCurve(X1, Y1, X2, Y2, X3, Y3 : TReal) přidá bézierovu křivku, která je pak definována CurrentPoint a body X1, Y1, X2, Y2, X3, Y3.

procedure AddArc(First : boolean; Radius, X0, Y0, X3, Y3, XT, YT : TReal) - přidá úsečku (pokud je definován aktuální bod) z aktuálního bodu do bodu [X0, Y0], nebo posun do bodu [X0, Y0], jestliže aktuální bod definován není, a bézierovu křivku, která představuje oblouk z bodu [X0, Y0] do [X3, Y3] s poloměrem Radius. Bod [XT, YT] je bod, kde se dotýká tečna rovnoběžná s úsečkou [X0, Y0]-[X3, Y3]. Z předchozích hodnot lze získat zbývající dva body bézierovy křivky určením F:

$$F = \frac{4}{3.(1+\sqrt{1+(XT-X0)^2+(YT-Y0)^2}).Radius^2}$$

Body definující požadovanou bézierovu křivku pak budou:

[X0, Y0]

[X0+(XT-X0).F,Y0+(YT-Y0).F]

[X3+(XT-X3).F,Y3+(YT-Y3).F]

[X3, Y3]

procedure InsertArc(Clockwise : boolean; XC, YC, Radius, Angle1, Angle2 : TReal - pomocí volání metody **AddArc** sestaví žádaný oblouk z jedné nebo několika bézierových křivek a vloží jej do analytického popisu grafické cesty. Hodnota Angle1 určuje úhel, kde oblouk na kružnici začíná, Angle2 určuje koncový úhel, Clockwise určuje směr vykreslování oblouku (po směru nebo protisměru hodinových ručiček). XC, YC jsou souřadnice středu, Radius poloměr oblouku. Pokud je úhel vykreslovaného oblouku větší než 90^{o}, pak se oblouk rozdělí na segmenty po 90^{o} a jeden zbytek menší než 90^{o}.

8. 12 .3 Metody správy paměti pro ukládání popisu grafické cesty

V předchozím textu bylo popsáno jakým způsobem objekt **GraphPath** doplňuje popis grafické cesty. Následující metody jsou určeny pro alokování a ukládání jednotlivých segmentů do paměti. Segmenty grafické cesty jsou následujících typů: posun na nový bod, úsečka z aktuálního bodu do nového, bézierova křivka a uzavření aktuální grafické podcesty, každému typu přísluší záznam určité struktury:

8. 12 .3 .1 .1 .1 Posun a úsečka:

```
PGrLine=^TGrLine;
TGrLine=record
            Code : byte;      určení typu
            P1 : TGrPoint;  koncový bod posunu nebo úsečky
          end;
```

8. 12 .3 .1 .1 .2 Bézierova křivka:

```
PGrCurve=^TGrCurve;
TGrCurve=record
                Code : byte;      určení typu
                P1 : TGrPoint;  druhý bod určující křivku (prvním bodem je aktuální bod)
                P2 : TGrPoint;   třetí
                P3 : TGrPoint;    čtvrtý bod
            end;
```

8. 12 .3 .1 .1 .3 Uzavření grafické podcesty:

```
PGrClosePath=^TGrClosePath;
TGrClosePath=record
                  Code : byte;  určení typu
                  FirstSegment : pointer; ukazatel na strukturu, která je začátkem uzavírané podcesty
                end;
```

Tyto datové struktury jsou ukládány v paměťových blocích o velikosti 2kB. Tato velikost byla volena tak, aby se neprováděla alokace pro každý relativně malý segment a přitom se na druhé straně nealokoval příliš velký blok paměti pro cesty s malým počtem segmentů. (Do jednoho bloku lze umístit 56 segmentů bézierových křivek nebo 156 segmentů úseček). Každá sekvence záznamů uvnitř bloku končí zvláštním záznamem, který odkazuje na další paměťový blok (v případě, že jeden blok pro uložení popisu nestačí, pak se další bloky řetězí) nebo označuje konec popisu grafické cesty:

8. 12 .3 .1 .1 .4 Odkaz na další blok:

```
PGrNextBlock=^TGrNextBlock;
TGrNextBlock=record
                  Code : byte;
                  NextBlock : pointer;
                end;
```

8. 12 .3 .1 .1 .5 Konec popisu grafické cesty:

```
PGrEndOFSequence=^TGrEndOfSequence;
TGrEndOfSequence=record
                     Code : byte;
                   end;
```

Kódy jednotlivých segmentů v paměťovém bloku jsou následující:

```
Gr_Move           = 1 - posun do bodu
Gr_Line           = 2- úsečka
Gr_Curve          = 3 - křivka
Gr_ClosePath      = 4 - uzavření cesty
Gr_NextBlock      = 98 - odkaz na následující paměťový blok
Gr_EndOfSequence  = 99 - konec sekvence popisu
```

Na první paměťový blok popisu cesty ukazuje atribut MemBlock. I v případě, že je grafická cesta prázdná, je paměťový blok pro ukládání vytvořen a obsahuje záznam konce popisu.

procedure NewMemBlock(PointerToBlock : PGrNextBlock) - tato metoda vytvoří nový paměťový blok pro ukládání. V případě, že je parametr PointerToBLock = ***nil***, pak se ukazatel na nově vytvořený blok uloží do atributu MemBlok, jinak je parametr ukazatel na záznam o pokračování v dalším paměťovém bloku a do něj se též zaznamenává ukazatel na tento nově napojovaný blok.

procedure CopyAllBlocks(NewP : PGraphPath) - vytvoří kopii celého řetězce paměťových bloků do kopie grafické cesty, na kterou ukazuje NewP. (Kopie vlastního objektu grafické cesty se vytváří pomocí virtuální metody **Copy** společné pro všechny netriviální objekty).

procedure ReleaseAllBlocks - uvolní celý řetězec paměťových bloků z paměti.

procedure InsertNextBlock - tato metoda je volána při požadavku dalšího paměťového bloku popisu a automaticky zajistí vytvoření nového bloku metodou **NewMemBlock** a zapíše správné informace do

vnitřku bloků, tak aby bylo zajištěno zřetězení bloků. Požadavek na další paměťový blok se objevuje při uložení segmentu grafické cesty, který při uložení překročí určitý bezpečný limit zbývajícího prostoru v paměťovém bloku. Tento limit je odvozen od nejdelšího možného záznamu + záznamu odkazu na další blok.

8. 12 .4 Metody vykreslování grafické cesty do rastrové grafické stránky

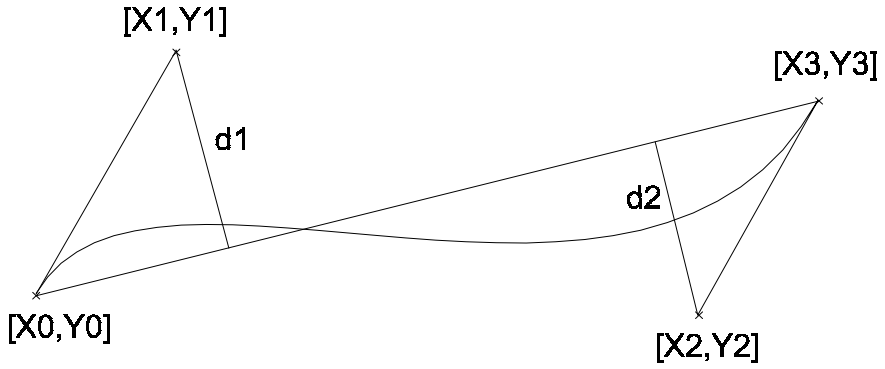
procedure DrawPath(DragDc : HDC) - provede vykreslení obrysu aktuální grafické cesty do výstupního kanálu DragDC (výstupní kanál může být okno, tiskárna nebo Windows metafile. V programu je využíváno vykreslování do okna výstupního zařízení. Nejprve se definuje se vytvoří pro dané okno region podle aktuální cesty ořezu. Prochází se aktuální cesta a vykresluje se grafický tvar. Bézierovy křivky se proloží úsečkami s přesností, kterou určuje atribut Flatness v objektu aktuálního grafického stavu. Tato metoda je používána pro vykreslování do bitmapy grafické stránky (funkce operátoru **stroke**) a aktuální cesty do OKNA AKTUÁLNÍ GRAFICKÉ CESTY.

8. 12 .4 .1 .1 .1 Proložení bézierovy křivky

Metoda proložení křivky vychází z vlastností křivky: bézierova křivka prochází oběma krajními body, dva vnitřní body určují zakřivení (krajními a vnitřními body jsou myšleny dva páry bodů určující bézierovu křivku). Vzdálenost libovolného bodu ležícího na křivce od přímé spojnice krajních bodů je vždy menší než vzdálenost nejméně jednoho vnitřního bodu od spojnice. Křivka se rekurzivně dělí na poloviny, do té doby, kdy vzdálenost obou vnitřních bodů od přímé spojnice krajních bodů je menší než limitní hodnota (atribut Flatness).

8. 12 .4 .1 .1 .2 Algoritmus dělení křivky [10]:

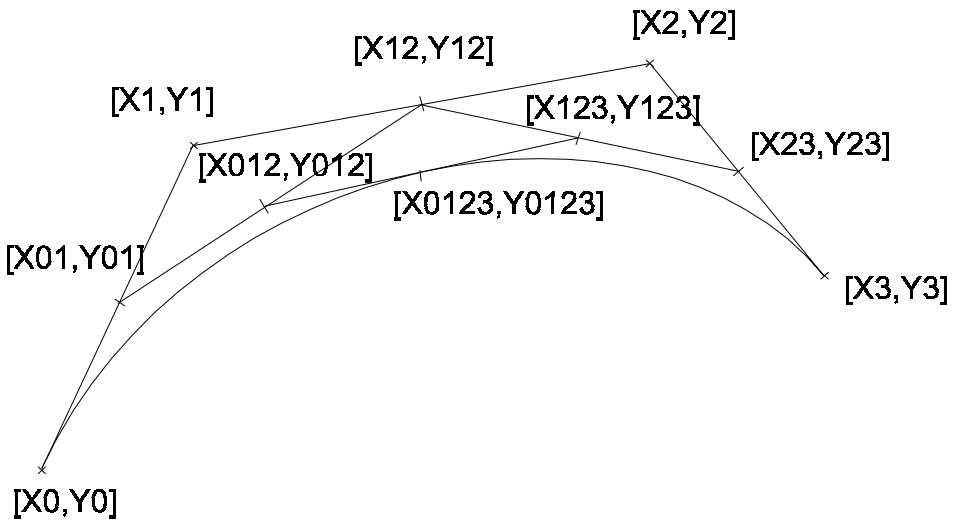
1. Určí se vzdálenost obou vnitřních bodů od úsečky [X0, Y0],[X3, Y3].



dx:=X3-X0
dy:=Y3-Y0
$d1 := \frac{dy*(X1-X0)-dx*(Y1-Y0)}{\sqrt{(dy*dy+dx*dx)}}$
Podobně i pro druhý bod

2. Pokud jsou obě vzdálenosti menší než limit, pak je možno nahradit křivku úsečkou

3. Pokud je větší, pak je třeba křivku rozdělit na dvě a pro každou novou křivku provést ten samý postup od bodu 1. Rozdělení křivky (viz [10]):



Bézierova křivka určená body [X0, Y0];[X1, Y1];[X2, Y2];[X3, Y3] se rozdělí na dvě určené body [X0, Y0];[X01, Y01];[X012, Y012];[X0123, Y0123] a [X0123, Y0123];[X123, Y123];[X23; Y23];[X3, Y3], kde
x01:=(x1+x0)/2
y01:=(y1+y0)/2
x12:=(x2+x1)/2
y12:=(y2+y1)/2
x23:=(x3+x2)/2
y23:=(y3+y2)/2
x012:=(x12+x01)/2
y012:=(y12+y01)/2
x123:=(x12+x23)/2
y123:=(y12+y23)/2
x0123:=(x012+x123)/2
y0123:=(y012+y123)/2

procedure FillPath(DragDC:HDC) - provede vykreslení plochy uvnitř grafické cesty. Postupuje se stejně jako v předchozím případě. Cesta je převedena na sekvenci úseček. Pro vykreslení plochy jsou použity procedury GDI **Polygon** a **PolyPolygon**. Nejprve se naplní buffer popisu polygonů po převedení aktuální grafické cesty se vyvolá procedura **Polygon** (pokud grafická cesta sestává z jedné podcesty) nebo **PolyPolygon** (pokud grafická cesta obsahuje více podcest). Tato metoda je používána pro vykreslování do bitmapy grafické stránky (funkce operátoru **fill**) a aktuální cesty ořezu do OKNA AKTUÁLNÍ CESTY OŘEZU.

8. 13 Objekt GraphStack (unit GRAFIKA.PAS)

Operátory **gsave** a **grestore** zajišťují uložení a obnovení aktuálního grafického stavu. Při uložení grafického stavu se ukládají tyto objekty: grafický stav, grafická cesta, cesta ořezu a aktuální transformační matice CTM. Vytvoří se tedy záložní kopie těchto objektů ve virtuální paměti a odkazy na tyto uložené objekty se vloží do záznamu typu TGraphStateRecord, který se uloží na grafický zásobník. Protože uložené objekty jsou netriviální, je důležité poznamenat, že vkládání a vybírání automaticky zvětšuje a zmenšuje hodnotu čítače sdílení těchto objektů.

8. 13 .1 Atributy:

VirtMemory : PVirtMemory - odkaz na virtuální paměť, kam přísluší uložené netriviální objekty.

POBox : array [0..30] of TGraphSaveRecord - pole pro ukládání.

Top, Bottom, Curr : word - vrchol, dno a aktuální položka zásobníku.

8. 13 .2 Metody:

function Push(Element : TGraphSaveRecord):bool - Tato metoda vytvoří kopie objektů, na které jsou odkazy v záznamu Element. Při vytváření těchto kopií obdrží tato metoda nové odkazy na kopie a ty uloží na zásobník v záznamu typu TGraphSaveRecord. V případě správného vykonání vrací metoda hodnotu ***true***.

function Pop(var Element : TGraphSaveRecord):bool - vybere ze zásobníku záznam typu TGraphSaveRecord a vrací jej v parametru Element. V případě správného vykonání metoda vrací ***true***.

Typ TGraphSaveRecord

```
type TGraphSaveRecord=record
                        XGraphState : TMemDescr - odkaz na objekt grafického stavu
                        XGraphPath : TMemDescr - odkaz na objekt grafické cesty
                        XClipPath : TMemDescr - odkaz na objekt cesty ořezu
                        XCTM : TMemDescr odkaz na aktuální transformační matici
                      end;
```

8. 14 Objekt Font (unit FONTY.PAS)

EMSCRIPT pracuje s fonty jednodušším způsobem než popisuje předpis PostScriptu. Je možno tento modul (FONTY) rozšířit, ale většině uživatelů pro první přiblížení struktury PostScriptového programu tento mechanizmus postačí. Vychází se z hotových písem True Type ve Windows™. Správu fontu zajišťuje objekt TFont v modulu Fonty.

8. 14 .1 Atributy:

Name : PChar jméno fontu.

Font : TLogFont struktura, která se používá pro aktivaci True Type fontu ve Windows™.

OutputDC : HDC připojení na výstupní zařízení.

OldTrueTypeFont : HFont předchozí font příslušející výstupnímu zařízení.

TrueTypeFont : HFont font vytvořený pomocí struktury TLogFont procedurou ***CreateFontIndirect*** (Borland Pascal 7.0).

XOffset : integer zde se kumuluje posuv souřadnice x při psaní slova. Hodnota se použije pro modifikaci grafické cesty po vykreslení textu.

Instance tohoto objektu se vytvoří PostScriptovým operátorem **findfont**. V konstruktoru se naplní struktura Font : TLogFont.

Operátor **scalefont** změní ve struktuře Font : TLogFont údaj o velikosti písma.

Operátor **setfont** označí příslušný objekt za aktuální tím, že odkaz na něj zapíše do atributu Font aktuálního grafického stavu.

Výstup písma do grafické stránky se provádí operátorem **show**. Uloží se aktuální grafická cesta. Text určen pro výstup se rozebere na jednotlivé znaky. Pro každý znak se provede procedura ***GetGlyphOutline***, která vrátí posloupnost úseček a kvadratických B-spline křivek. Posloupnost se přenese do grafické cesty PostScriptu, přičemž se provádí transformace podle CTM. Po převedení všech znaků do grafické cesty, dojde k vykreslení této grafické cesty do grafické stránky, obnoví se předchozí grafická cesta a přidá se segment move podle osy x o délku textu v daném fontu.

Poznámka: Vnitřní mechanizmus vykreslování fontů v programu EMSCRIPT nepoužívá tabulku *Encoding*, takže zde není možnost interpretovat programy, které používají tuto tabulku.

8. 14 .2 Popis funkce *GetGlyphOutLine*:

function GetGlyphOutline(A : HDC; B, C : word; var D : TGlyphMetrics; E : longint; F : PChar; var G : TMat2):longint;

A	určuje přiřazení k výstupnímu zařízení (používá se atribut OutDC v TFont)
B	obsahuje kód znaku, jehož tvar chceme obdržet
C	určuje formát, v jakém se bude vracet tvar znaku. Může nabývat hodnot: GGO_BITMAP - tvar ve formě bitmapy (není v EMSCRIPTu použito) GGO_NATIVE - vrací body určující úsečky a B-spline křivky Pokud je C=0, pak se plní pouze informace o velikosti v D (zde není použito)
D	TGlyphMetrics struktura, která popisuje uložení znaku ve znakové buňce
E	určuje velikost bufferu, ve kterém jsou uloženy informace o tvaru znaku. Pokud jsou tento parametr roven 0 a parametr C je buď GGO_BITMAP nebo GGO_NATIVE, vrací funkce požadovanou velikost tohoto bufferu
F	ukazatel na buffer, ve kterém budou uloženy informace o tvaru znaku. Jestliže je C=GGO_NATIVE, pak jsou tyto informace uloženy v posloupnosti struktur TTTPOLYGONHEADER a TTTPOLYCURVE
G	ukazuje na transformační matici, která se používá při C=GGO_BITMAP. Tento parametr však nesmí být nil ani v případě, že C=GGO_NATIVE.

V našem případě je tato funkce používána tak, že popis tvaru je vrácen jako posloupnost kontur. Každá kontura je popsána strukturou TTTPOLYGONHEADER, za kterou následuje příslušný počet TTTPOLYCURVE struktur. Všechny body určující křivky jsou typu TPOINTFX a reprezentují absolutní pozici na výstupním zařízení na výstupním zařízení. (Zde bylo nutno simulovat výpis písmene do souřadnic [0,0] pro lepší manipulaci se získaným tvarem znaku). Počáteční bod kontury je uložen v atributu pfxStart struktury TTTPOLYGONHEADER. TTTPOLYCURVE struktury mohou popisovat buď posloupnost úseček nebo kvadratickou B-spline křivku, které jsou používány pro definici True Type fontů.

Typ TTTPOLYGONHEADER=record

cb : longint	určuje počet bytů potřebných pro TTTPOLYGONHEADER a posloupnosti TTTPOLYCURVE
dwType : longint	je vždy TT_POLYGONTYPE
pfxStart : TPointFx	počáteční bod popisu kontury

Za každou strukturou TTTPOLYGONHEADER následuje nejméně jedna struktura TTTPOLYCURVE.

Typ TTTPOLYCURVE=record

wType : word	určuje typ křivky popisovaný touto strukturou: TT_PRIM_LINE nebo TT_PRIM_QSPLINE
cpfx : word	určuje počet elementů v následujícím poli
apfx : array of TPointFx	obsahuje body popisující tvar kontury

TT_PRIM_LINE	popisuje řadu bodů, které jsou spojeny úsečkami.
TT_PRIM_QSPLINE	je série bodů, které definují kvadratické spline funkce používané v True Type technologii pro popis křivek. Q-spline je popsána třemi body (A, B a C), kde bod A a C jsou leží na křivce a bod B mimo ni. Rovnice pro popis křivky je následující: $V(t)=(A-2B+C)t^2 + (2B - 2A)t + A$ Vlastni formát dat je takovýto: Bod A na Q-spline křivce je aktuální bod (buď se jedná o pfxStart v TTTPOLYGONHEADER nebo o koncový bod předchozí křivky. Bod B je aktuální bod v posloupnosti. Bod C může být definován dvojím způsobem: Pokud má posloupnost dva nebo více bodů, které následují po B, pak bod C je střední bod mezi bodem B a následujícím bodem v posloupnosti, jinak bod C je bod v posloupnosti hned za bodem B. Následující příklad demonstruje, jakým způsobem je možno body v posloupnosti číst: pfxA:=pfxStart {Počáteční bod pro polygon} for u:=0 to cpfx-1 do {Projde všechny body v posloupnosti}

```
begin
  pfxB:=apfx[u];
  if u<cpfx-2 then   { Za bodem B následuje více než jeden bod }
    begin
      pfxC.x:=(pfxB.x + apfx[u+1].x)/2; { Střední bod }
      pfxC.y:=(pfxB.y + apfx[u+1].y)/2;
    end          else
      pfxC:=apfx[u+1];
  KresliQ-spline(pfxA, pfxB, pfxC);
  pfxA:=pfxC;
end;
```

```
Typ TPointFx=record
        x: TFixed
        y: TFixed

Typ Tfixed=record
        fract : word          desetinná část čísla
        value : integer       celá část čísla
```

8. 15 Převod HSV do RGB

PostScript umožňuje definovat barvu v prostoru RGB nebo v prostoru HSV nazývaného též (HSB, HLS nebo HBS). Emulační program používá GDI funkcí, které pracují v RGB prostoru. Proto se definice barvy v HSB prostoru konvertují do prostoru RGB. Rutiny pro konverzi použité v emulátoru jsou RGBtoHSV a HSVtoRGB v unit Grafika.

8. 16 Obsluha chybového stavu

V unit CHYBY.PAS je deklarován objekt Err, jenž uchovává informace o chybě, která nastala. V tomto unitu je též definována globální proměnná ErrorHandle:PErr, která představuje indikaci chybového stavu pro celý interpret. V případě chybového stavu se kterákoliv část interpretu odkáže na tuto proměnnou. Od této chvíle je indikován chybový stav a interpret po nezbytném uklizení a ukončení právě rozpracovaných činností vydá chybové hlášení, které identifikuje typ chyby a kde tato chyba nastala. O chybové hlášení se postará metoda **ZpracovaniChyby** v hlavním okně aplikace (objekt **HlavniOkno** v MAINMDI.PAS).

Objekt Err (unit CHYBY.PAS). V případě výskytu chyby se v této verzi programu zablokuje vykonávání dalších příkazů interpretu a interpret se musí uvést do korektního stavu volbou **Reset interpret** v menu **Run**. Interpret se neuvádí do korektního stavu automaticky, aby uživatel mohl nahlížet na vnitřní datové struktury s hodnotami v době výskytu chyby.

8. 16 .1 Atributy:

Error:byte - typ chyby:

err_noerror	= 0 - není indikována žádná chyba
err_dictfull	= 1 - snaha o přidání nového páru do zaplněného slovníku
err_dictstackoverflow	= 2 - snaha o uložení slovníku do plného slovníkového zásobníku
err_dictstackunderflow	= 3 - snaha o uvolnění slovníku z prázdného zásobníku slovníků
err_execstackoverflow	= 4 - přeplnění procedurálního zásobníku
err_execstackunderflow	= 24 - vybírání z prázdného procedurálního zásobníku
err_handleerror	= 5 - není použito v EMSCRIPT
err_interupt	= 6 - není použito v EMSCRIPT
err_invalidacces	= 7 - není použito v EMSCRIPT
err_invalidfileacces	= 26 - chybný přístup k souboru
err_invalidexit	= 25 - není použito v EMSCRIPT
err_invalidfont	= 8 - není použito v EMSCRIPT
err_invalidrestore	= 9 - není použito v EMSCRIPT
err_ioerror	= 10 - chyba při vstupně výstupních operacích
err_limitcheck	= 11 - překročení limitu

err_nocurrentpoint	= 12 - není definován aktuální bod grafické cesty
err_rangecheck	= 13 - překročení rozsahu
err_stackoverflow	= 14 - přetečení zásobníku operandů
err_stackunderflow	= 15 - snaha vybírat z prázdného zásobníku operandů
err_timeout	= 16 - není použito v EMSCRIPT
err_typecheck	= 17 - neočekávaný typ
err_undefined	= 18 - není definován
err_undefinedfilename	= 19 - nedefinované jméno souboru
err_undefinedresult	= 20 - nedefinovatelný výsledek
err_unmatchedmark	= 21 - není použito v EMSCRIPT
err_unregistered	= 22 - není použito v EMSCRIPT
err_VMerror	= 23 - přeplnění virtuální paměti
err_syntaxerror	= 98 - chyba v syntaxi zdrojového textu
err_endoffile	= 99 - konec vstupního souboru
err_criticalerror	= 255 - kritická chyba aplikace

Where : array [0..30] of char - zde je text, který identifikuje modul nebo činnost, při které nastal chybový stav.

8. 16 .2 Metody:

procedure NoError - nastaví objekt do stavu Bez chyby

procedure NewValue(AWhere : PChar; Err : byte) - nastaví hodnoty objektu podle indikované chyby, zde záleží na jednotlivých modulech, jakým způsobem opíší chybový stav.

function Value : byte - vrací typ chyby

function Description : PChar - vrací text ,který blíže popisuje okolnosti chybového stavu.

9. Závěr

Výsledkem této diplomové práce je interpret, který interpretuje programy v jazyku PostScript pro vybranou množinu operátorů. Dovoluje uživateli jednoduchým a přehledným způsobem krokovat interpretaci, nahlížet na obsah zásobníku operandů, dovoluje pozorovat celou cestu vytváření grafické cesty a grafické cesty ořezu, až po vykreslení do grafické stránky. Tuto grafickou stránku pak může uživatel vytisknout na libovolné tiskárně. Program EMSCRIPT je tak vhodnou pomůckou pro výuku programovacího jazyka PostScript. V rámci vytváření programu jsem se seznámil s programovacím jazykem PostScript, s objektovým a událostmi řízeným programováním. Seznámil jsem se s praktickým využitím poznatků z počítačové grafiky a tvorby interpretu. Vyšší verze programu by mohla být rozšířena na celou sadu PostScriptových operátorů. Dále je možno přidat další náhledové prostředky detailních pohledů na PostScriptové objekty. Při testování programu se ukázalo, že by bylo velmi vhodné umožnit krokovat interpretaci též uvnitř procedury. To předpokládá, že jednotlivé prvky procedurálního pole budou mít odkaz na svůj původ. Implementaci tohoto nástroje jsem však musel odložit kvůli velkému rozsahu prací v jiných částech interpretu.

10. Příloha A - Implementované operátory

10. 1 Operátory pro manipulaci se zásobníkem operandů

pop	vyzvedne operand zásobníku
exch	zamění dva operandy na vrcholu zásobníku
dup	udělá duplikát operandu na vrcholu zásobníku
copy	provede kopii určeného počtu elementů na vrcholu zásobníku
index	duplikuje operand na určité pozici v zásobníku
roll	provede rotaci operandů na zásobníku
clear	vyprázdní zásobník operandů
count	počet operandů na zásobníku
mark	uloží na zásobník objekt typu **mark**
cleartomark	uvolní obsah zásobníku až po první objekt typu **mark**
counttomark	počet operandů až po první **mark**

10. 2 Aritmetické operátory

add	součet
div	dělení
idiv	celočíselné dělení
mod	zbytek po dělení
mul	násobení
sub	odečítání
abs	absolutní hodnota
neg	negace
ceiling	nejbližší vyšší celé číslo
floor	nebližší nižší celé číslo
round	zaokrouhlení na celé číslo
truncate	odstraní desetinnou část
sqrt	druhá odmocnina
atan	arcus tangens
cos	kosinus
sin	sinus
exp	exponent
ln	přirozený logaritmus
log	desítkový logaritmus
rand	náhodné číslo

srand	nastavuje inicializační číslo pro pseudonáhodné generování
rrand	čte inicializační číslo pro pseudonáhodné generování

10. 3 Operátory nad objekty typu pole

array	vytvoří pole zadané velikosti
[	začátek definice pole
]	konec definice pole
length	velikost pole
get	vrací prvek pole na zadaném objektu
put	vloží do pole prvek
aload	vloží všechny elementy pole na zásobník operandů
astore	přenese operandy ze zásobníku operandů do pole

10. 4 Operátory komprimovaného pole

Komprimované pole slouží pro úspornější uložení v paměti. Toto pole je možno nahradit polem nekomprimovaným, a proto tento typ ani opereátory nad ním nejsou v této verzi vestavěny.

10. 5 Operátory nad slovníky

dict	vytvoří slovník zadané kapacity
length	počet uložených párů ve slovníku
maxlength	kapacita slovníku
begin	vloží slovník na zásobník slovníků
end	uvolní slovník ze zásobníku slovníků
def	vloží pár do slovníku
load	hledá ve slovnících na zásobníku slovníků
store	přepíše nejvyšší výskyt definice klíče
get	vyhledá v určitém slovníku daný klíč
put	vloží pár do určitého slovníku
known	testuje, zda se klíč vyskytuje ve slovníku
where	vrací slovník, kde se nachází hledaný klíč
systemdict	uloží ***systemdict*** na zásobník operandů
userdict	uloží ***userdict*** na zásobník operandů
currentdict	uloží slovník na vrcholu zásobníku slovníků na zásobník operandů
countdictstack	počet slovníků v zásobníku slovníků

10. 6 Operátory řetězců

string	vytvoří řetězec zadané velikosti
length	velikost řetězce znaků

10. 7 Relační, logické a bitové operátory

eq	je rovno
ne	není rovno
ge	větší nebo rovno
gt	větší
le	menší nebo rovno
lt	menší
and	logické, bitové and
not	logické, bitové not
or	logické, bitové or
xor	logické, bitové xor
true	vloží na zásobník operandů true
false	vloží na zásobník operandů false
bitshift	bitový posun

10. 8 Řídící operátory

if	vykoná proceduru, pokud je vstupní hodnota *true*
ifelse	vykoná jednu z procedur podle hodnoty vstupního logického operandu
for	vykoná for cyklus
repeat	vykoná několikrát určitou proceduru
loop	cyklicky provádí danou proceduru

10. 9 Operátory typu, atributů a konverzí

cvx	nastavuje příznak spustitelný objekt
cvlit	odznačuje příznak spustitelný objekt
cvi	konvertuje číslo nebo řetězec znaků na celé číslo
cvn	konvertuje řetězec znaků na jméno
cvr	konvertuje číslo nebo řetězec znaků na desetinné číslo
cvs	konvertuje libovolný objekt na řetězec znaků

10. 10 Operátory nad externími soubory

Tato verze interpretu je zaměřena na výklad jednodušších příkladů bez použití externích souborů.

10. 11 Operátory virtuální paměti

Tato verze nepředpokládá programové konstrukce s operátory pro ovládání virtuální paměti.

10. 12 Různé operátory

bind	v proceduře zamění jména operátorů za objekty typu operátor

10. 13 Operátory grafického stavu

grestoreall	obnova nejspodněji uloženého grafického stavu
gsave	uložení aktuálního grafické stavu
grestore	obnovení grafického stavu
setlinewidth	nastavení tloušťky čáry
curentlinewidth	vrací aktuálně nastavenou tloušťku čáry
setlinecap	nastavení způsobu zakončení čáry
currentlinecap	vrací aktuální nastavení způsobu zakončení čáry
setlinejoin	nastavení způsobu napojení čar
currentlinejoin	vrací aktuální nastavení napojení čar
setmitterlimit	nastavení limitního úhlu spojení čar
currentmitterlimit	vrací aktuální nastavení limitního úhlu spojení čar
setflat	nastavení tolerance při proložení křivky úsečkami
currentflat	aktuální tolerance při proložení křivky úsečkami
setgray	nastavení barvy na určitý stupeň šedi
currentgray	vrací aktuální nastavení stupně šedi
sethsbcolor	nastavení barvy HSB hodnotami
currenthsbcolor	vrací aktuální nastavení barvy v HSB hodnotách
setrgbcolor	nastavení barvy v RGB hodnotách
currentrgbcolor	vrací aktuální nastavení barvy v RGB hodnotách

10. 14 Operátory matic a souřadných systémů

matrix	vytvoří matici
initmatrix	nastaví CTM (aktuální matici transformace) na implicitní nastavení výstupního zařízení
currentmatrix	naplní matici hodnotami CTM
setmatrix	nastaví hodnoty CTM podle zadané transformační matice
translate	provede operaci posunu na CTM nebo zadané matici
scale	provede operaci změny měřítka na CTM nebo zadané matici
rotate	provede operaci rotace podle počátku na CTM nebo zadané matici
identmatrix	nastaví matici na matici identity

defaultmatrix	nastaví matici na implicitní transformační matici výstupního zařízení
concat	provede vynásobení CTM a zadané matice, výsledek uloží do CTM
concatmatrix	provede vynásobení dvou matic
transform	provede transformaci souřadnic podle CTM nebo zadané matice
dtransform	provede transformaci diferencí souřadnic podle CTM nebo zadané matice
itransform	provede inverzní transformaci souřadnic podle CTM nebo zadané matice
idtransform	provede inverzní transformaci diferencí souřadnic podle CTM nebo zadané matice
invertmatrix	provede inverzi zadané matice

10. 15 Operátory konstrukce grafické cesty

newpath	inicializace aktuální grafické cesty
currentpoint	vrací souřadnice aktuálního bodu
moveto	nastavení aktuálního na zadané souřadnice
rmoveto	relativní moveto
lineto	úsečka z aktuálního bodu na zadané souřadnice
rlineto	relativní lineto
arc	oblouk protisměru hodinových ručiček
arcn	oblouk po směru hodinových ručiček
curveto	beziérova křivka
closepath	uzavření grafické podcesty
clippath	nastavení aktuální grafické cesty obsahem grafické cesty ořezu
charpath	přidání obtahu textu do aktuální cesty
initclip	inicializace grafické cesty ořezu
clip	nastavení cesty ořezu podle aktuální grafické cesty
eoclip	nastavení cesty ořezu podle aktuální grafické cesty

10. 16 Operátory vykreslování do grafické stránky

fill	vykreslení výplně grafické cesty
eofill	vykreslení výplně grafické cesty
stroke	vykreslení obtahu grafické cesty
erasepage	vykreslení celé plochy grafické stránky bílou barvou

10. 17 Operátory výstupu a nastavení výstupního zařízení

Interpret není zaměřen na speciální nastavení výstupního zařízení. Grafickou stráku je možno vytisknout pomoci ovladačů WINDOWS™, které lze nastavit přímo v tomto systému.

10. 18 Operátory písma a znaků

findfont	vrací font podle zadaného názvu
scalefont	nastaví velikost fontu
setfont	nastaví font na aktuální
currentfont	vrací aktuální font
show	vypíše text na grafickou stránku

10. 19 Operátory vyrovnávací paměti fontů

Mechanizmy výpisu textu nepoužívají vyrovnávací paměť.

11. Příloha B - Operátory, které nejsou implementovány

11. 1 Operátory pro manipulaci se zásobníkem operandů

Všechny operátory této skupiny jsou vestavěny.

11. 2 Aritmetické operátory

Všechny operátory této skupiny jsou vestavěny.

11. 3 Operátory nad objekty typu pole

getinterval	vytvoří objekt jako interval z daného pole
putinterval	vloží nové hodnoty z jednoho pole do druhého
copy	zkopíruje všechny elementy pole do druhého pole
forall	pro všechny elementy provede proceduru

11. 4 Operátory komprimovaného pole

packedarray	vytvoří komprimované pole
currentpacking	vrací mód komprimování
setpacking	nastaví aktuální mód komprimace
length	velikost komprimovaného pole
get	vyzvedne jeden prvek z komprimovaného pole
getinterval	uloží interval z komprimovaného pole do jiného nekomprimovaného pole
aload	uloží na zásobník všechny elementy komprimovaného pole
copy	zkopíruje elementy komprimovaného pole do jiného pole
forall	pro všechny elementy provede proceduru

11. 5 Operátory nad slovníky

copy	zkopíruje obsah slovníku
forall	pro každý element slovníku provede proceduru
errordict	uloží ***errordict*** na zásobník operandů
dictstack	zkopíruje obsah slovníku do pole

11. 6 Operátory řetězců

get	vrací znak určitého indexu
put	vloží znak do řetězce znaků
getinterval	vloží na zásobník podřetězec
putinterval	vloží do řetězce obsah podřetězce
copy	zkopíruje řetězec

forall	pro každý znak řetězce provede proceduru
anchorsearch	definuje zda je podřetězec odvozený od daného řetězce
search	vyhledá podřetězec v řetězci
token	rozpozná atomickou formuli na začátku řetězce

11. 7 Relační, logické a bitové operátory

Všechny operátory této skupiny jsou vestavěny.

11. 8 Řídící operátory

exec	vykoná libovolný objekt
stop	ukončí blok, který je vykonáván jako *stopped*
stopped	zavede blok, který je možno ukončit operátorem **stop**
countexecstack	počet prvků na procedurálním zásobníku
execstack	vloží elementy procedurálního zásobníku do pole
quit	ukončí interpret
start	vykoná se při spouštění interpretu

11. 9 Operátory typu, atributů a konverzí

type	vrací řetězec, kde je slovně popsán typ vstupujícího objektu
xcheck	testuje příznak spustitelný objekt
executeonly	redukuje přístup na *execute-only*
noacces	neumožňuje žádný přístup k objektu
readonly	dedukuje přístup pouze pro čtení
rcheck	testuje, zda je objekt určen pouze pro čtení
wcheck	testuje, zde je možno objekt měnit
cvrs	konvertuje číslo v libovolné soustavě na řetězec znaků

11. 10 Operátory nad externími soubory

file	otevře externí soubor
closefile	uzavře externí soubor
read	přečte jeden znak z externího souboru
write	zapíše jeden znak do externího souboru
readhexstring	přečte řetězec hexadecimálních hodnot ze souboru do řetězce znaků
writehexstring	zapíše řetězec hexadecimálních hodnot z řetězce znaků do souboru
readstring	přečte řetězec znaků ze souboru
writestring	zapíše řetězec znaků do souboru
readline	přečte řádku ze souboru do řetězce znaků

token	přečte atomickou formuli ze souboru
bytesavailable	počet bytů v souboru určených pro čtení
flush	pošle data ve vyrovnávací paměti na standardní výstup
flushfile	pošle data ve vyrovnávací paměti do souboru nebo čte do konce souboru
resetfile	vymaže obsah vyrovnávací paměti
status	vrací *true*, pokud je soubor otevřen
run	vykoná externí soubor zadaného jména
currentfile	vrací soubor, který se právě vykonává
print	vypíše řetězec znaků na standardní výstup
=	vypíše textovou reprezentaci libovolného objektu na standardní výstup
= =	vypíše syntaktickou reprezentaci na standardní výstup
stack	vypíše obsah zásobníku operandů na standardní výstup (textovou reprezentaci)
pstack	vypíše obsah zásobníku operandů na standardní výstup (syntaktickou reprezentaci)
prompt	vykoná se vždy, když je interpret připraven pro interaktivní vstup
echo	povoluje a potlačuje vypisování zpráv

11. 11 Operátory virtuální paměti

save	uložení aktuálního stavu virtuální paměti
restore	obnovení stavu virtuální paměti
vmstatus	zpráva o stavu virtuální paměti

11. 12 Různé operátory

null	uloží objekt bez významu na zásobník operandů
usertime	vrací čas v milisekundách
version	vrací verzi interpretu

11. 13 Operátory grafického stavu

initgraphics	inicializace grafického stavu
setdash	nastavení čerchování čáry
currentdash	vrací aktuální nastavení čerchování čáry
setscreen	nastaví půltónování
currentscreen	vrací aktuální půltónování
settrasfer	nastavení procedury půltónování
currenttransfer	vrací aktuální proceduru půltónování

11. 14 Operátory matic a souřadných systémů

Všechny operátory této skupiny jsou vestavěny.

11. 15 Operátory konstrukce grafické cesty

arcto	tangentový oblouk
rcurveto	relativní curveto
flattenpath	proložení křivek v grafické cestě úsečkami
reversepath	obrácení směru grafické cesty
strokepath	nahrazení aktuální grafické cesty jejím obrysem
pathbbox	vrací obdélník ohraničující aktuální grafickou cestu
pathforall	provede zadané procedury pro všechny prvky grafické cesty

11. 16 Operátory vykreslování do grafické stránky

image	vykreslení bitmapového obrázku do grafické stránky
imagemask	vykreslení bitmapové masky do grafické stránky

11. 17 Operátory výstupu a nastavení výstupního zařízení

showpage	výstup a resetování aktuální grafické stránky
copypage	výstup grafické stránky
banddevice	instalace výstupního zařízení
framedevice	instalace výstupního zařízení
nulldevice	instalace zařízení bez výstupu (černá díra)
renderbands	vykoná proceduru pro jednotlivé oblasti stránky na výstupním zařízení

11. 18 Operátory písma a znaků

definefont	registruje nový font
makefont	transformuje existující font podle matice a vytvoří nový font
ashow	vypíše text na grafickou stránku se zadanými mezerami mezi písmeny
widthshow	vypíše text na grafickou stránku se zadanými mezerami u zadaného písmene
awidthshow	vypíše text na grafickou stránku kombinuje **ashow** s **widthshow**
kshow	vypíše text na grafickou stránku a mezi písmeny vykoná zadanou proceduru
stringwidth	vrací velikost ohraničujícího obdélníku zadaného řetězce znaků v aktuálním fontu
FontDirectory	slovník slovníku fontů
StandardEncoding	kódovací vektor standardního fontu

11. 19 Operátory vyrovnávací paměti fontů

cachestatus	vrací nastavení vyrovnávací paměti

setcachedevice	deklaruje rozměry znaků procházející zpracováním ve vyrovnávací paměti
setcachwidth	deklaruje rozměry znaků, které neprocházejí zpracování ve vyrovnávací paměti
setcacheparam	nastavuje parametry vyrovnávací paměti
currentcacheparam	vrací aktuální nastavení parametrů vyrovnávací paměti

12. Literatura

[1] Adobe system incorporated; PostScript language, Reference manual; Addison-Wesley Publishing company; USA; 1990

[2] Skala V.; Počítačová grafika I,II,III;Ediční středisko VŠSE v Plzni; Plzeň; 1991

[3] Granát L., Sechovský H.; Počítačová grafika; SNTL; Praha; 1980

[4] Gries D.; Kompilátory číslicových počítačov; Alfa; Bratislava; 1981

[5] Borland; ObjectWindows™ Programing Guide; 1992

[6] Vanžura J.; Objektové programování v Turbo Pascalu pro Windows; KOPP; České Budějovice; 1992

[7] Fořt I.; Windows 3, techniky programování; Grada a.s.; Praha; 1992

[8] Schildt H., Pappas Ch. H., Murray W. H.; Osborne Windows programing series volume 1-3; Osborne McGraw-Hill; Berkeley; 1994

[9] Dror A.; Modifying Windows; Osorne McGraw-Hill; Berkeley; 1994

[10] Barsky B.A., DeRose D.A.; The beta2-split: A special case of the beta-spline curve and surface representation, IEEE 1985.

[11] Wirth N.; Algoritmy a štruktúry údajov Alfa; Bratislava; 1988

[12] Adobe system incorporated; PostScript language, Tutorial and cookbook; Addison-Wesley Publishing company; USA;