Ing. Miloš F I D R M U C prom. mat., Jaroslav K A S E prom.mat.

Vladimír V A L O U C H prom mat.

Výzkumný ústav matematických strojů Praha

Programovací jazyk pro simulaci SIMSCRIPT

Simulace je experimentální technika operačního výzkumu. Spočívá v tom, že zkoumaný systém modelujeme pomocí počítače a s modelem provádíme experimenty, abychom pochopili, jak systém pracuje, předpověděli jeho chování v určitých situacích, ověřili, která z možných variant organisace systému je nejvhodnější, nebo abychom se naučili chování systému ovlivňovat či řídit. V mnoha případech je simulace vlastně jedinou metodou, jak nějaký systém zkoumat, nechceme-li se spokojit s pouhou intuicí. Jde o případy, že např.:

- systém nelze adekvátně matematicky popsat
- matematický popis systému by byl příliš složitý a vedl by k těžko řešitelným matematickým problémům / zejména když chování systému resp. vnější vlivy působící na systém mají stochastickou povahu/
- se systémem nelze přímo prakticky experimentovat buď proto, že je teprve navrhován /např. organisace dopravy a skladování surovin a výrobků v projektované továrně/, nebo protože by to bylo nebezpečné či příliš nákladné /provoz na letišti, regulace dopravy ve městě, ekonomický systém/

Vytvoření programu, modelujícího studovaný systém na počítači,

je spravidla velmi komplikované a je tím složitější, čím nižší je úroveň programovacího jazyka, který při vytváření modelu použijeme. Dokonce i obecné programovací jazyky, jako je třeba FORTRAN IV nebo ALGOL 60, se dají k vytváření modelů složitějších systémů použít jen s potížemi. Na druhé straně zase obecné programovací jazyky v některých situacích neumožňují napsat program dostatečně efektivně, což je nepříjemné zejména u modelů zahrnujících nějaké náhodné chování, u nichž je třeba při simulaci provádět často velké množství náhodných pokusů, takže nároky na strojový čas se stávají nežádoucími. Proto vznikla celá řada tzv. simulačních programovacích jazyků, které mají tuto práci usnadnit. Můžeme je rozdělit do dvou skupin. První skupinu tvoří jazyky pro tzv. spojitou simulaci, které umožňují - zhruba řečeno - modelovat nějaký systém na číslicovém počítači podobným způsobem, jako se to dělá na analogových počítačích. Druhou skupinu tvoří jazyky pro diskrétní simulaci. Diskrétní simulace je vhodná pro modelování systémů, v nichž probíhají děje, které si lze /při určité schematisaci/ představovat jako posloupnosti nějakých "událostí", které nastávají v diskrétních časových okamžicích. Tak např. při modelování provozu v továrně mohou takovými událostmi být spuštění stroje, porucha stroje, zahájení či ukončení nějaké výrobní operace, příjem objednávky, příjem zásilky materiálu, expedice výrobku a pod.

Simulační jazyky usnadňují vytváření diskrétních modelů obvykle v několika směrech:

- poskytují programátorovi aparát k zachycení skutečnosti, že v systému probíhá paralelně v čase více procesů, které se mohou vzájemně ovlivňovat
- umožňují snadno modelovat chování náhodných veličin a statisticky sledovat chování zkoumaného systému
- umožňují vytvářet v paměti počítače složité struktury dat odpovídající strukturám v reálném světě /např. různě organisované množiny/
- poskytují tvůrci modelu soustavu pojmů, které mu pomáhají modelovaný systém myšlenkově zvládnout a popsat

Programovacích jazyků pro diskrétní simulaci vznikla již celá řada. Některé z nich byly úzce zaměřeny jen na určitou oblast aplikací, některé již zastaraly. V současné době se zdá, že se v zahraničí nejvíce používají jazyky SIMSCRIPT /Rand Corporation/, GPSS /firma IBM/ a SIMULA /Dahl,Nygaard - Norské výpočetní centrum/. Simulační techniky se zatím nejvíce používají v USA.

Jazyk SIMSCRIPT II je koncipován jako obecný programovací jazyk. Vznikl podstatným zdokonalením a přepracováním jazyka SIMSCRIPT I, který vycházel z FORTRANu.

Program v SIMSCRIPTu II sestává z nepovinné preambule, hlavního programu a případně z podprogramů. V preambuli se deklarují globální proměnné a pole, specifikují se vlastnosti podprogramů a deklarují se další struktury dat - tzv. entity a množiny entit, které se využívají zejména při simulaci a jsou rovněž "globální", t.j. přístupné ze všech podprogramů. V hlavním programu a v podprogramech mohou být deklarovány lokální proměnné a lokální pole. Pole mohou mít dimensi až 255.

Možnost práce s entitami je jeden z rysů, kterými se SIMSCRIPT II výrazně liší od běžných programovacích jazyků. Entitou rozumíme n-tici jakýchsi proměnných, zvaných atributy této entity. Představujeme si obvykle, že entita znázorňuje nějaký objekt v reálném světě a že hodnoty atributů vyjadřují vlastnosti tohoto objektu. Atributy mají svá jména. V SIMSCRIPTu se deklarují vždy celé třídy entit, přičemž všechny entity dané třídy mají stejné atributy a vzájemně se liší jen jejich hodnotami. Entity mohou při výpočtu "vznikat" a "zanikat". To vede k nutnosti dynamické alokace paměti. Důležitou vlastností entit je, že je lze sdružovat do množin. Realisováno je to tak, že entity mají v takovém případě zvláštní atributy, jejichž hodnoty udávají např. další entitu v množině, předchozí entitu v množině a pod. Každá množina entit musí mít "vlastníka". Tímto vlastníkem může být jiná entita nebo celý modelovaný systém. Rozeznáváme dva druhy entit: temporérní entity a permanentní entity. Permanentní entity dané třídy se vzájemně odli-

šují indexy. Při výpočtu mohou vzniknout nebo zaniknout vždy jen všechny permanentní entity dané třídy najednou. Jejich výhodou je to, že jsou díky indexování vždy všechny snadno dostupné. Temporární entity mohou při výpočtu vznikat i zanikat jednotlivě. Pracuje se s nimi tak, že se uchovávají pointery na ně, nebo že se sdružují do množin. Atribut temporární resp. permanentní entity se adresuje pomocí svého jména a pointeru na entitu resp. indexu entity.

Dynamická alokace paměti umožňuje, že podprogramy lze v SIMSCRIPTu volat rekursivně a dále že pole jsou dynamická, t.j. že paměť se pro ně reservuje dynamicky při výpočtu a že paměť vyhrazenou pro pole, které už není dále potřebné, lze uvolnit a dát tak k dispozici pro vznik jiných struktur dat.

Simulace se v SIMSCRIPTu provádí tak, že se modelovaný systém popíše pomocí proměnných, polí, entit a množin, a typické změny, které mohou v systému nastávat při jeho činnosti, se popíšou pomocí speciálních podprogramů, zvaných události. Tyto události se pak vyvolávají v pořadí odpovídajícím časové následnosti změn stavu modelovaného systému. Plynutí času se přitom zobrazuje tím, že se mění hodnota speciální proměnné pojmenované TIME.V - tzv. systémového času.

Při spuštění programu se nejdříve model uvede do nějakého počátečního stavu a naplánuje se, při kterých hodnotách systémového času mají nastat některé události a s jakými parametry mají být vyvolány. Poté se příkazem START SIMULATION předá řízení tzv. aktivisační rutině, která posunuje systémový čas a spouští události. Systémový čas se posunuje "po skocích" od jedné události k druhé, takže období, kdy se v modelovaném systému nic neděje, se přeskakují. Každá událost přitom může způsobit naplánování jiných událostí nebo naopak může zabránit tomu, aby došlo k nějaké již naplánované události. Tím je umožněno, aby model pracoval v jistém smyslu autonomně; může v něm docházet ke změnám a situacím, které sice nějak vyplývají z počátečního stavu modelu, z jeho vnitřní logiky a ze vstupních dat, ale přitom je nesnadné je předem předvídat.

SIMSCRIPT též umožňuje, aby některé události nastávaly na základě dat vstupujících ze specifikovaných vstupních zařízení /tzv. vnější události/. Takovým zařízením může být také např. dálnopis nebo jiný terminál, pomocí něhož může experimentátor přímo zasahovat do chování modelu. Tímto způsobem lze do činnosti modelu zapojit i více lidí, takže simulace může získat podobu jakési hry mezi více lidmi a počítačem.

Plánování událostí a jejich spouštění je realisováno pomocí speciálních temporárních entit zvaných zápisy událostí. Pro každý druh událostí existuje odpovídající třída zápisů událostí. Zápisy událostí mají jako jeden svůj atribut hodnotu systémového času, ve kterém má dojít k příslušné události. Zařazují se do standardní množiny EV.S, v níž jsou uspořádány do podmnožin podle druhů událostí a v každé podmnožině pak vzestupně podle hodnot systémového času. Aktivisační rutina pracuje tak, že vyhledá v množině EV.S vždy zápis události s nejnižším /t.j. nejbližším/ systémovým časem, tento zápis události vyřadí z množiny EV.S, posune systémový čas na hodnotu určenou zápisem události a spustí příslušnou událost s parametry odpovídajícími případným dalším atributům zmíněného zápisu události.

K usnadnění modelování náhodných jevů je v SIMSCRIPTu k disposici jako standardní funkce generátor náhodných čísel, který může produkovat 10 nezávislých proudů pseudonáhodných čísel s rovnoměrným rozložením v intervalu (0 , 1). Na tento generátor navazuje řada dalších standardních funkcí, které umožňují generovat náhodné čísla s různými často se vyskytujícími teoretickými rozloženími nebo náhodná čísla s empirickými rozloženími zadanými tabelovanými hodnotami distribučních funkcí.

Dalším specifickým rysem SIMSCRIPTu je možnost tzv. monitorace globálních proměnných, polí a atributů entit. Tyto objekty lze v preambuli programu deklarovat jako monitorované vpravo resp. vlevo, což způsobí, že se při každém použití daného objektu pro výpočet hodnoty aritmetického výrazu, resp. pokaždé, když se objektu předává hodnota, vyvolá stejnojmenná prevostran-

ná resp. levostranná monitorovací rutina, kterou programátor sám napíše. Může v ní pracovat jak s hodnotou, která je přenášena, tak s pointerem resp. indexy určujícími adresu monitorovaného objektu. Monitorace lze výhodně používat jednak k ladění programů, jednak k modifikování činnosti modelu.

Pomocí monitorace je umožněno také statistická sledování hodnot různých parametrů modelu během simulace. Programátor může v preambuli programu deklarovat některé globálně dostupné objekty jako statisticky sledované a uvést, jaké jejich charakteristiky chce sledovat /např. průměr, rozptyl, maximální hodnotu/. Překladač na základě této deklarace vyhradí pro dané objekty zvláštní proměnné - střadače a vygeneruje monitorovací rutiny, které při změně hodnot sledovaných objektů opravují hodnoty střadačů a v případě potřeby počítají z hodnot střadačů požadované statistické charakteristiky.

Prostředky pro vstup a výstup dat jsou v SIMSCRIPTu navrženy s maximálním ohledem na pohodlí programátora. Existují jednak možnosti bezformátového vstupu i výstupu, jednak lze používat formátů obdobně jako ve FORTRANu. Parametry formátů mohou být aritmetické výrazy, takže lze snadno naprogramovat např. tisk grafů. Kromě toho jsou v jazyce ještě speciální prostředky pro usnadnění tisku tabulek, tisku hlaviček na stránkách a pod.

----oooOOOooo----

Na závěr našeho článku předvedeme použití SIMSCRIPTu na příkladu.

Představme si, že nějaká opravárenská dílna má formou servisu udržovat v chodu jistý počet nějakých strojů. O těchto strojích je známo, že se náhodně porouchávají. Časový interval mezi ukončením opravy a další poruchou u každého stroje je náhodná veličina s exponenciálním rozdělením s průměrem 100 hodin. Nastane-li porucha, je doba T potřebná k opravě, za předpokladu, že opravu bude provádět jediný mechanik, opět náhodná veličina, tentokrát se stejnoměrným rozdělením v intervalu

2 - 10 hodin. Nepracuje-li stroj, dochází za každou hodinu prostoje k ekonomické ztrátě 10 000 Kčs. Bude-li na opravě stroje pracovat více mechaniků, zkrátí se doba potřebná k opravě. Toto zkrácení můžeme vyjádřit tak, že dobu T vynásobíme nějakým faktorem FAC(K) , který je funkcí počtu mechaniků K . Volba FAC (K) = $^{1}/_{K}$ není ovšem realistická, protože při větších K si budou mechanici pravděpodobně vzájemně překážet a dobu opravy nebude možno zvyšováním počtu mechaniků dále zmenšit. Nebudeme se zde snažit funkci FAC(K) blíže určit a spokojíme se raději s konstatováním, že FAC (1) = 1 a že pro K> 1 bude FAC (K) nejdříve klesat, až dosáhne nějaké minimální /kladné/ hodnoty a pak bude opět stoupat, nebo aspoň nebude dále klesat. V konkrétním případě by se muselo vycházet z empirické zkušenosti. Předpokládejme dále, že mechanici mají pevný hodinový plat, ať pracují nebo ne.

Úkolem simulace je zjistit, kolik mechaniků má být v dílně zaměstnáno a v jak velkých skupinách mají pracovat, aby průměrná celková ztráta, sestávající ze ztráty způsobené prostoji strojů a z platu mechaniků, byla minimální.

Budeme nyní uvažovat o tom, jak problém namodelovat v SIMSCRIPTu a současně se budeme odkazovat na program uvedený na konci tohoto článku.

Protože počet strojů se během simulace nemění, zobrazíme je permanentními entitami /příkazy 2,3/. Atributy stroje budou "čas poruchy" /t.j. čas, kdy se stroj naposled porouchal/ a "počet zaměstnaných mechaniků" v případě, že stroj je právě opravován /příkaz 3/. Počet zaměstnaných mechaniků deklarujeme jako celočíselnou proměnnou, proměnná "čas poruchy" bude automaticky reálná /přík.4/.

V modelu může docházet k těmto událostem:

- porucha stroje
- zahájení opravy stroje
- konec opravy stroje

Oprava stroje může být zahájena buď bezprostředně poté, co se stroj porouchá, nebo poté, co skončí oprava jiného stroje

a mechanici, kteří byli opravou zaměstnáni, mohou přejít na opravu čekajícího porouchaného stroje. Protože tedy zahájení opravy stroje časově spadá buď s poruchou stroje nebo s koncem opravy jiného stroje, není třeba považovat je za zvláštní událost. Naproti tomu však je vhodné zavést událost "konec simulace", která ukončí simulaci v nějakém předem stanoveném systémovém čase /viz příkaz 5/. Parametrem událostí "porucha stroje" a "konec opravy stroje" musí být číslo stroje, kterého se událost týká. Proto nadeklarujeme "číslo stroje" jako atribut zápisů událostí pro oba tyto druhy událostí /příkazy 6,7,8/.

Jestliže se stroj porouchá a nejsou právě k dispozici mechanici, aby ho začali opravovat, musí stroj čekat na opravu. Na opravu může současně čekat více strojů. Budou pak tvořit frontu - množinu porouchaných strojů /příkazy 3,9/, jejímž vlastníkem bude celý modelovaný systém.

Dále nadeklarujeme funkci pro výpočet faktoru FAC a několik globálních proměnných, jejichž smysl vyplyne z dalšího výkladu /příkazy 10-13/.

Příkazem 14 se způsobí, že všechny lokální proměnné deklarované v hlavním programu a v podprogramech budou celočíselné, nebude-li v konkrétních případech stanoveno jinak.

Obratme se nyní k událostem popisujícím chování modelu. Dojde-li k poruše stroje /přík.24/, přiřadí se současná hodnota systémového času atributu "čas poruchy" stroje, který se porouchal /přík.25/. Globální proměnná POC.VOL.MECH udává počet mechaniků, kteří právě nejsou zaměstnáni žádnou opravou. Jsou-li takoví mechanici k dispozici, zahájí se oprava stroje ihned. Za tím účelem se vyvolá podprogram "zařiď opravu stroje" s parametrem udávajícím číslo porouchaného stroje /přík.26a/. Tím událost skončí /přík.26b/. Není-li k dispozici žádný mechanik, zařadí se porouchaný stroj do množiny porouchaných strojů čekajících na opravu /přík.27/ a událost rovněž skončí.

Podprogram "zařiď opravu stroje" je tvořen příkazy 30 - 38. Příkazem 31 se deklaruje reálná lokální proměnná DOBA. Lokální proměnná I použitá negř. v příkazu 33 se deklaruje automaticky

jako celočíselná /viz příkaz 14/. V podprogramu se nejdříve vygeneruje náhodné číslo DOBA, udávající, jak dlouho by trvala oprava porouchaného stroje, kdyby ji prováděl jediný mechanik /přík.32/. Následující příkazy vyjadřují strategii používanou při přidělování mechaniků na opravy. V našem případě bude přiděleno Y mechaniků, kde Y může maximálně dosáhnout hodnoty, dané globální proměnnou POC.VE.SKUP /"počet ve skupině" - viz přík.33/. Nemůže být ovšem přiděleno více mechaniků, než kolik je jich k dispozici. Počet volných mechaniků se příslušně zmenší /přík.34/ a počet přidělených mechaniků se stane hodnotou atributu POCET.ZAM.MECH porouchaného stroje /přík.35/. Poté se naplánuje událost "konec opravy" porouchaného stroje v závislosti na závažnosti poruchy a počtu přidělených mechaniků /přík.36/ a tím podprogram skončí /přík.37/.

Změnou podprogramu "zařiď opravu stroje" bychom mohli vyzkoušet též jiné strategie v přidělování mechaniků, např. přidělovat mechaniky s ohledem na závažnost poruchy /pokud by to ovšem odpovídalo reálné situaci, t.j. pokud vedoucí servisu je schopen ze zprávy o poruše posoudit její závažnost/. Bylo by též možné napsat tento podprogram tak, aby se na nějaký terminál vypsaly potřebné údaje a člověk sedící u terminálu by se mohl cvičit v operativním rozhodování a sám určovat, kolik mechaniků na opravu přidělí.

Při ukončení opravy stroje /přík.39/ se zjistí ztráta způsobená prostojem a přičte se k hodnotě globální proměnné ZTRATA /přík.40/; mechanici, kteří na opravě pracovali, se opět zahrnou mezi volné mechaniky /přík.41/. Poté se naplánuje další porucha zmíněného stroje /přík.42/. Proměnná STREAM přitom udává číslo proudu náhodných čísel, použitého při generaci náhodného čísla s exponenciálním rozdělením. Protože se ukončením opravy uvolnili nějací mechanici, může být v případě, že nějaký stroj čeká na opravu / t.j. že množina porouchaných strojů je neprázdná/, zahájena další oprava /přík.43/.

Celý experiment zorganisujeme tak, že při jednom spuštění programu provedeme pro daný počet strojů a daný maximální po-

čet mechaniků ve skupině několik simulací pro různý celkový počet mechaniků. Na začátku hlavního programu tedy přečteme počet strojů, hodinový plat mechanika, meze, v nichž se má měnit počet mechaniků, krok, s nímž se má počet mechaniků měnit, počet mechaniků ve skupině, dobu simulace a číslo proudu náhodných čísel pro generaci hodnot náhodných veličin během simulace /příkaz 17/. Globální proměnné N.STROJ se deklaruje automaticky při deklaraci třídy permanentních entit STROJ a udává celkový počet strojů. Příkaz 17 ukazuje možnost bezformátového čtení v SIMSCRIPTu. Příkazem 18 se pro kontrolu vypíšou hodnoty proměnných STREAM a POC.VE.SKUP . Příkaz 18 ukazuje možnost výstupu pomocí formátů. Poté se na tiskárně vynechají tři řádky příkazem 19. Nyní se příkazem 20 vytvoří všechny stroje /jejich počet je dán hodnotou N.STROJ/. Pak se bude pro různé počty mechaniků /cyklus 21/ spouštět model. Naplánuje se vždy pro každý stroj jeho první porucha /21b/, inicialisuje se proměnná "počet volných mechaniků" /21c/, naplánuje se konec simulace /21d/ a spustí se vlastní simulace /21e/.

Na konci každé simulace /přík.45/ se vytiskne počet mechaniků a celková ekonomická ztráta vztažená na 1 hodinu /přík.46/. Poté se model připraví pro další spuštění /příkazy 47-50/.

Ukážeme ještě, jak by šlo např. zjistit průměrný počet nečinných mechaniků vážený podle doby, po kterou nebyli využiti. Stačilo by v preambuli programu kdekoli za příkazem 11 uvést příkaz

ACCUMULATE PRUM.VOL.MECH AS AVERAGE OF POC.VOL.MECH

čímž by se proměnná POC.VOL.MECH stala statisticky sledovanou. V události "konec simulace" bychom pak mohli spočtený průměr vytisknout třeba příkazem bezformátového psaní

LIST PRUM.VOL.MECH

Aby se nulovaly statistická střadače před zahájením další simulace, bylo by poté třeba ještě napsat příkaz

RESET TOTALS OF POC.VOL.MECH

----oooOOOooo----

V uvedeném příkladu jsme předvedli jen malou část prostředků, které SIMSCRIPT poskytuje programátorovi. Silná stránka SIMSCRIPTu je právě v jeho obecnosti. Domníváme se, že by mohl být výhodně použit i k programování jiných úloh než simulačních.

---oooOOOooo---

```
číslo
příkazu
 1      PREAMBLE
 2      PERMANENT ENTITIES
 3        EVERY STROJ HAS SOME CAS.PORUCHY, SOME POCET.ZAM.MECH
            AND MAY BELONG TO MN.POR.ST
 4        DEFINE POCET.ZAM.MECH AS INTEGER VARIABLE
 5      EVENT NOTICES INCLUDE PORUCHA, KONEC.OPRAVY,
                                              KONEC.SIMULACE
 6        EVERY PORUCHA HAS SOME CISLO.STROJE
 7        EVERY KONEC.OPRAVY HAS A CISLO.STROJE
 8        DEFINE CISLO.STROJE AS AN INTEGER VARIABLE
 9      THE SYSTEM OWNS A MN.POR.ST
10      DEFINE FAC AS REAL FUNCTION
11      DEFINE POC.MECH, PIAT.MECH, MIN.POC.MECH, MAX.POC.MECH,
            KROK, POC.VE.SKUP AND POC.VOL.MECH AS INTEGER
                                                   VARIABLES
12      DEFINE STREAM AS INTEGER VARIABLE
13      DEFINE ZTRATA AND DOBA.SIMULACE AS REAL VARIABLES
14      NORMALLY, MODE IS INTEGER
15      END
```

```
16      MAIN
17      READ N.STROJ, PIAT.MECH, MIN.POC.MECH, MAX.POC.MECH,
          KROK, POC.VE.SKUP, STREAM AND DOBA.SIMULACE
18      WRITE STREAM AND POC.VE.SKUP AS "STREAM:", I 6, S 5,
          "SKUPINY PO", I 6, " MECHANICICH" , /
19      SKIP 3 LINES
20      CREATE EACH STROJ
21      FOR POC.MECH = MIN.POC.MECH TO MAX.POC.MECH BY KROK
          DO
21a         FOR EACH STROJ
21b          SCHEDULE PORUCHA (STROJ)  IN
                    EXPONENTIAL.F  (1ØØ.Ø,STREAM)   UNITS
21c         LET POC.VOL.MECH = POC.MECH
21d         SCHEDULE KONEC.SIMULACE IN DOBA.SIMULACE UNITS
21e         START SIMULATION
21f       LOOP
22      STOP
23      END

24      EVENT PORUCHA GIVEN POROUCHANY.STROJ
25      LET CAS.PORUCHY (POROUCHANY.STROJ) = TIME.V
26      IF POC.VOL.MECH > Ø
26a         CALL ZARID.OPRAVU (POROUCHANY.STROJ)
26b         RETURN
26c     ELSE
27      FILE POROUCHANY.STROJ LAST IN MN.POR.ST
```

```
28      RETURN
29      END

30      ROUTINE ZARID.OPRAVU          (POR.ST)
31      DEFINE DOBA AS REAL VARIABLE
32      LET DOBA = UNIFORM.F (2.Ø, 1Ø.Ø, STREAM+1)
33      LET Y = MIN.F (POC.VOL.MECH, POC.VE.SKUP)
34      SUBTRACT Y FROM POC.VOL.MECH
35      LET POCET.ZAM.MECH (POR.ST) = Y
36      SCHEDULE KONEC.OPRAVY (POR.ST) IN DOBA * FAC (Y) UNITS
37      RETURN
38      END

39      EVENT KONEC.OPRAVY GIVEN POR.ST
40      ADD 1ØØØØ * (TIME.V - CAS.PORUCHY (POR.ST))  TO ZTRATA
41      ADD POCET.ZAM.MECH (POR.ST)  TO POC.VOL.MECH
42      SCHEDULE PORUCHA (POR.ST)  IN EXPONENTIAL.F (1ØØ.Ø,STREAM)
                                                          UNITS
43      IF MN.POR.ST IS NOT EMPTY
43a       REMOVE FIRST STROJ FROM MN.POR.ST
43b       CALL ZARID.OPRAVU (STROJ)
43c     ELSE RETURN
44      END

45      EVENT KONEC.SIMULACE
46      PRINT 3 LINES WITH POC.MECH  AND
```

```
        ZTRATA / DOBA.SIMULACE + POC.MECH * PLAT.MECH
        AS FOLLOWS
      POCET MECHANIKU :          ****
      CELK.ZTRATA ZA CAS.JEDNOTKU : ******.**

47    LET TIME.V = 0.0      LET ZTRATA = 0.0
48    UNTIL MN.POR.ST IS EMPTY
          REMOVE FIRST STROJ FROM MN.POR.ST
49    FOR I = 1 TO EVENTS.V  LET F.EV.S (I) = 0
50    RETURN
51    END

52    ROUTINE FOR FAC (K)
       .......
      RETURN WITH ....
      END
```

---oooOOOooo---

Literatura:

1. The SIMSCRIPT II Programming Language , P.J.Kiviat, R.Villanueva, and H.M.Markowitz , Prentice-Hall,Inc., Englewood Cliffs, New Jersey, 1968
2. The SIMSCRIPT II Programming Language: Reference Manual, P.J.Kiviat and R Villanueva, The Rand Corporation, Santa Monica, California ; Prentice-Hall, Inc., 1968