ALGPSS, een programma voor discrete simulatie van dynamisch stochastische processen

Dietz, J.L.G.

Award date:
1972

4 C P C 3

Technische Hogeschool Eindhoven
Afdeling Electrotechniek,
groep Meten en Regelen.

algpss, een programma voor discrete simulatie van dynamisch stochastische processen.

afstudeerverslag van J.L.G. Dietz.

Eindhoven, 19 november 1970.

Hoogleraar: Prof. dr. ir.
P. Eijkhoff
Coaches : Ir. J.A.G.M. Kerbosch
Ir. A.A. van Rede.

Summery.

GPSS is a program of IBM, with which stochastic dynamic processes easily can be simulated on a digital computer. It has however some disadvantages, e.g. the user does not know how exactly the simulation is executed and with which accuracy calculations are made. We have attempted to overcome these disadvantages by writing a simulation program in algol-60 with the features of GPSS.

Inhoudsopgave blz.

VOORWOORD

Veel bedrijfskundige problemen hebben betrekking op het gedrag van discrete, dynamische, stochastische systemen. Bij de bestudering van deze systemen is simulatie met behulp van een digitale computer een belangrijk hulpmiddel. Het eenvoudigst is deze simulatie indien het model van het te onderzoeken systeem gegeven kan worden als een procesbeschrijving. Een taal hiervoor door IBM ontwikkeld is GPSS/360. Hoewel het mogelijk is binnen de Technische Hogeschool Eindhoven GPSS/360 te gebruiken, namelijk op de IBM/360, leek het de groep applicatie-software van de afdeling Bedrijfskunde i.o. wenselijk te beschikken over een applicatieprogramma, geschreven in algol-60, waarmee discrete simulatie via een procesbeschrijving van het te simuleren systeem zou kunnen worden uitgevoerd. Een dergelijk programma is te verkiezen boven GPSS/360 om de volgende redenen:

1. Over enige jaren zal de Technische Hogeschool Eindhoven beschikken over een computer met een zeer goede en snelle algolcompiler, terwijl het niet zeker is of er ook een GPSS-compiler aanwezig zal zijn.

2. Tot die tijd kan gesimuleerd worden met behulp van de EL/X-8.

3. De werking van het applicatieprogramma ligt vast in de algoltekst. Dit betekent dat men gemakkelijk kan nagaan op welke wijze het aangeboden probleem wordt opgelost en wat de graad van nauwkeurigheid is waarmee de berekeningen worden uitgevoerd.

4. Het programma kan vrij eenvoudig worden gewijzigd, dit in tegenstelling tot GPSS/360 dat in de assembler language is geschreven.

Bovenstaande overwegingen hebben geleid tot de volgende, aan mij verstrekte opdracht:

1. Onderzoek of het mogelijk is het bedoelde applicatieprogramma te ontwerpen.

2. Indien dit mogelijk blijkt, voer het ontwerp dan uit. Neem hierbij als leidraad GPSS/360.

Dit verslag is een beschrijving van het applicatieprogramma, dat de naam "algpss" heeft gekregen.
Het is de bedoeling dat dit programma opgenomen wordt in het onderwijs aan de studenten in de afdeling der Bedrijfskunde i.o.
Bij het schrijven van dit verslag is uitgegaan van het standpunt dat de lezer al enigszins bekend is op het gebied van de discrete simulatie. De voorkennis, die wordt verondersteld is vervat in de syllabus "Inleiding Discrete Simulatie" van de hand van ir. J.A.G.M. Kerbosch en L.H. Kroep.

Eindhoven, 19 november 1970.

J.L.G. Dietz.

1. INLEIDING

1.1. Wachttijdproblemen.

Het simulatieprogramma "algpss" is ontworpen om als hulpmiddel te dienen bij de oplossing van een klasse problemen, die gewoonlijk wordt aangeduid met de naam "wachttijdproblemen" (queueing problems), waarvan hieronder twee voorbeelden volgen.

a. Dokterspraktijk.

Een dokter met een drukke praktijk bemerkt dat soms patiënten, die bij hem op het spreekuur komen, weglopen zonder behandeld te zijn. De reden daarvoor is dat zij de wachttijd te lang vinden. Nu blijkt dat de behandeling van de patiënten vaak moet worden onderbroken omdat er een telefoongesprek binnenkomt. De dokter overweegt de aanschaf van een bandopname-installatie voor deze telefoongesprekken; hij hoopt daardoor het aantal patiënten, dat wegloopt zonder consult, sterk te verminderen. Vóór dat hij tot de aanschaf overgaat zou hij echter graag willen weten of het gebruik van zo'n installatie ook het gewenste effect zal hebben.

b. Jobshop.

In een werkplaats bevinden zich drie machines die we aanduiden met de nummers 1, 2 en 3. Er worden drie typen orders uitgevoerd, die als volgt zijn gespecificeerd:

type 1 : deze orders ondergaan twee bewerkingen; eerst op machine 1, daarna op machine 2 of 3, elk met een kans van 50%.

type 2 : deze orders ondergaan drie bewerkingen; 50% doorloopt de machine in de volgorde 2 - 1 - 3, de andere helft in de volgorde 3 - 1 - 2.

type 3 : deze orders ondergaan ook drie bewerkingen en wel in de volgorde 3 - 2 - 1.

Elk type order heeft voor de verschillende machines specifieke bewerkingstijden.

Indien een machine een order bewerkt heeft van type i en daarna een

order van type j ($i \neq j$) in behandeling neemt, moet de machine opnieuw worden ingesteld.
De bewerkingstijden zijn klein ten opzichte van de tussenaankomsttijden van de orders. De insteltijden van de machines echter zijn ten opzichte hiervan zo groot, dat een willekeurige verwerking van ordertypen door elkaar heen tot zeer lange wachtrijen zou leiden. Men wil graag dat de langste verblijftijd van een order in de werkplaats zo kort mogelijk is, desnoods ten koste van het gemiddelde en vraagt zich nu af welke discipline moet worden gevolgd bij de keuze van een ordertype uit de wachtrijen vóór de machines. Men zou bijvoorbeeld de volgende discipline kunnen volgen:
Werk eerst (op elke machine) alle orders van hetzelfde type af. Stel dan de machine in voor de order, die het langst staat te wachten en neem dat type in bewerking.

1.2. Analyse.

Ofschoon de twee voorbeelden twee verschillende gebieden bestrijken bezitten zij essentiële punten van overeenkomst.
Dit blijkt uit een nadere analyse, waarbij de volgende opmerkingen kunnen worden gemaakt:

1. Er zijn permanente componenten (dokter, machines) en tijdelijke componenten (orders, patiënten, telefoongesprekken).
2. De tijdelijke componenten vormen het "verkeer" door het systeem (de orders doorlopen de werkplaats terwijl ze van de ene machine naar de andere gaan). Vergelijk bijvoorbeeld ook de stroom van elektrische pulsen in een digitaal netwerk.
3. De tijdelijke componenten worden door de permanente componenten "geholpen" gedurende een bepaalde tijd, waarna ze zich verder bewegen in het systeem.
4. De tussenaankomsttijden en/of de helptijden van de tijdelijke componenten zijn stochastisch van aard. Het is bijvoorbeeld niet bekend of er op een bepaald moment een patiënt de wachtkamer binnenkomt, maar wel is er op elk tijdstip een zekere kans dat dit gebeurt.
5. Als een tijdelijke component geholpen wil worden maar de permanente component is reeds bezet, dan moet de eerste wachten. Zo ontstaan wachtrijen. De tijdelijke componenten staan steeds te

"dringen" om te worden geholpen.

6. Sommige tijdelijke componenten kunnen prioriteit hebben op andere. Zo kunnen in voorbeeld (a) telefoongesprekken de behandeling van patiënten onderbreken en kan in voorbeeld (b) bij de gesuggereerde keuzediscipline een order in bewerking worden genomen vóór een order van een ander type, ofschoon de laatste langer in de wachtrij staat.
7. De toestand van het systeem verandert schoksgewijs, namelijk steeds als de toestand van één van de componenten zich wijzigt. Voorbeeld: de toestand van het systeem jobshop verandert als er een nieuwe order binnenkomt en als er een order op een machine is afgewerkt. Deze gebeurtenissen, die een verandering van de toestand van het systeem ten gevolge hebben, heten events.

1.3. Modelbouw.

Om het te onderzoeken systeem geschikt te maken voor simulatie wordt van het systeem een model gebouwd. Dit model is een abstractie en tevens een vereenvoudiging van de vaak gecompliceerde werkelijkheid , het bevat namelijk alleen die aspecten van het systeem, waarin de onderzoeker is geinteresseerd.
Een veel gebruikte methode bij het opstellen van modellen is de beschrijving van de structuur van het systeem m.b.v. een blokdiagram. Ieder blok stelt een bepaalde relevante activiteit van het systeem voor. De blokken zijn onderling verbonden door lijnen, die de weg aangeven, waarlangs de tijdelijke componenten zich door het model bewegen. Als voorbeeld van een blokdiagram is op blz. 6 een model van de dokterspraktijk geschetst.

MODEL VAN DOKTERSPRAKTIJK.

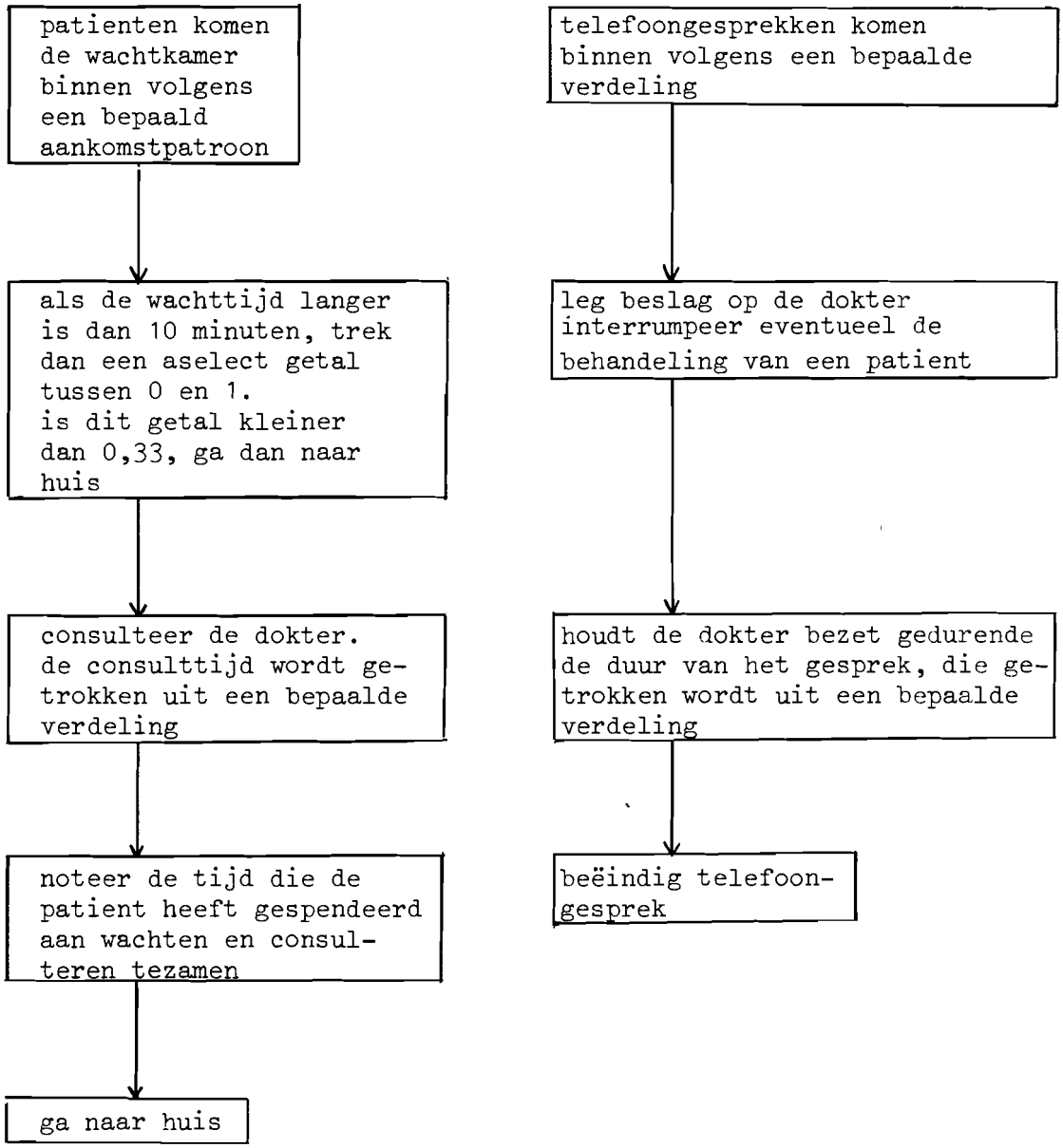


fig. 1.1.

2. GRONDSLAGEN VAN ALGPSS. CLOCKROUTINE.

In het voorgaande is gebleken dat discrete dynamische, stochastische systemen onderling een grote gelijkenis vertonen. Deze eigenschap maakt het mogelijk van de systemen modellen te bouwen, die zijn samengesteld uit een beperkte set entiteiten.
Figuur 2.1. (blz. 9) geeft een overzicht van de bouwstenen van algpss.
De aantallen entiteiten die voor een bepaald simulatieprogramma nodig zijn, worden voorgesteld door de waarden, die de gebruiker m.b.v. de initialisatieprocedure setalgpss aan de volgende variabelen van het type integer geeft.

ugntrans	: het aantal transactions, waarvoor wordt gesimuleerd
ugnfac	: het aantal facilities
ugnstor	: het aantal storages
ugnqueue	: het aantal queues
ugntable	: het aantal tables
ugnblock	: het aantal blokken

De componenten worden symbolisch voorgesteld door een aantal variabelen. De waarden van de variabelen bepalen de toestand van de component. Sommige variabelen zijn strikt intern, anderen staan ook ter beschikking van de gebruiker. Deze variabelen heten standaardattributen. Zij zijn vaak niet essentieel voor de werking van het programma, maar geven extra informatie over het verloop van het proces.

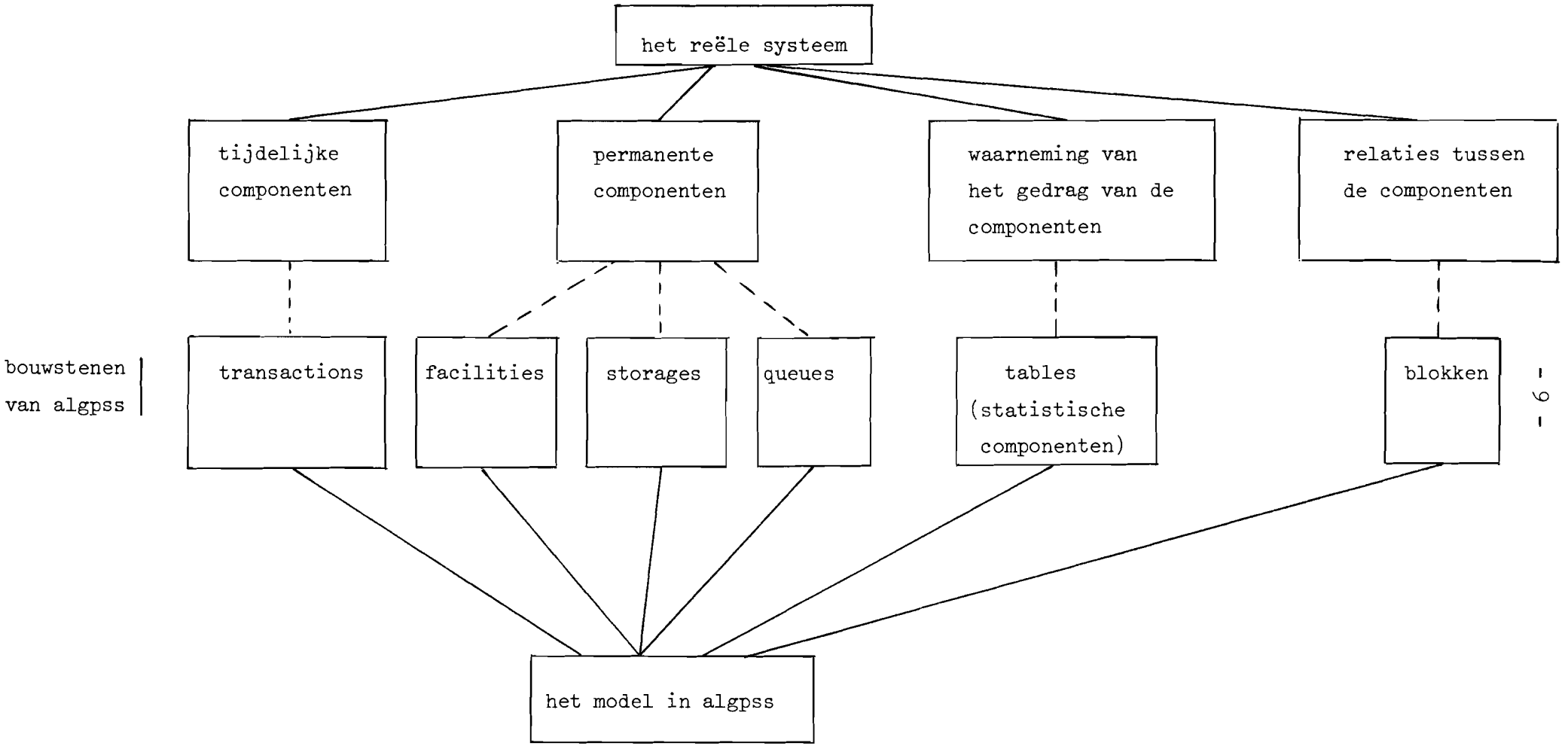
De transactions bewegen zich door een samenstel van permanente componenten. De weg , die een transaction volgt, wordt aangeduid door blokker De instructies in een blok hebben steeds betrekking op één transaction, de active transaction geheten (zie hfdst. 3.1.).
Hoewel de gebruiker het model opbouwt als een procesbeschrijving, wordt de simulatie uitgevoerd als ware het een eventbeschrijving. Er is in algpss een eventchain en een clockroutine. Deze zijn echter niet "zichtbaar" voor de gebruiker. Wel "zichtbaar" is het model

dat is samengesteld uit blokken, elk blok voorzien van een label (zie hoofdstuk 8).
Deze labels behoren tot de switch s, die aan het begin van het simulatieprogramma wordt gedeclareerd.
In de eventchain bevinden zich de transactions, die in de toekomst actief zullen worden. In elke transaction wordt bewaard het tijdstip waarop hij actief wordt en het nummer van het eerstvolgende blok, dat hij zal doorlopen.

In hoofdlijnen verloopt het simulatieproces als volgt:

1. De clock-routine haalt de eerste transaction uit de eventchain.
2. De tijd wordt aangepast: ugt:= eventtijdstip.
3. Het nummer van het eerstvolgende blok van de transaction wordt vastgesteld. Stel dit nummer is k.
4. De transaction wordt geactiveerd. Het programma gaat verder bij blok k (goto s[k]).
5. De blok-instructies worden uitgevoerd totdat de statement goto clock wordt bereikt.
   Dit gebeurt bijvoorbeeld als een transaction de toegang tot een facility wordt geweigerd.
   Het programma gaat terug naar de clock-routine.
6. De clockroutine haalt de volgende transaction van de eventchain en gaat verder bij (2).

De uitvoering van de simulatie geschiedt dus als een eventbeschrijving, waarbij de eventbeschrijving van een bepaald type event, afhankelijk van dat type, een zeker gedeelte van het model is.


het reële systeem
tijdelijke componenten
permanente componenten
waarneming van het gedrag van de componenten
relaties tussen de componenten
bouwstenen van algpss
transactions
facilities
storages
queues
tables (statistische componenten)
blokken
het model in algpss

Figuur 2.1.

3. COMPONENTEN

3.1. Transactions

Een transaction wordt voorgesteld door een reeks subscripted variables van het real array ugcc [-5: ugubcc] (chain case). Zo'n reeks heet een schakel van het array ugcc. Een schakel wordt aangeduid met zijn adres ugla (link address) en bestaat uit minimaal 8 geheugenplaatsen. (ugla-7 tot en met ugla).
In figuur 3.1.1. op blz. 12 is een overzicht gegeven van deze plaatsen en de betekenis van hun inhoud.
De gebruiker kan met behulp van de procedure setalgpss aan de transactions een aantal parameters meegeven. Aan de variabele ugnpar wordt dan een waarde groter dan 0 toegekend. Is bijvoorbeeld ugnpar = 2 dan behoren ook de plaatsen ugcc[ugla + 1] en ugcc[ugla + 2] tot de schakel. In deze twee plaatsen kan de gebruiker naar believen informatie bewaren. Het toekennen van een waarde aan een parameter geschiedt met behulp van de procedure assign. Het standaardattribuut par(i) levert de waarde van parameter i af, d.w.z. de waarde van ugcc[ugla + i].

Definities.

Losse transaction.
De schakels van het array ugcc kunnen gerangschikt zijn tot een chain. Als dit het geval is, heeft ugcc[ugla - 3] de waarde van het nummer van de chain, waartoe de transaction met schakeladres ugla behoort. Als een transaction niet tot een chain behoort heet hij "los" (ugcc [ugla - 3] = ugnill)*.

Active transaction.
Het simulatieprogramma opereert op elk moment op één transaction, namelijk die transaction, die het schakeladres heeft, dat momenteel gelijk is aan de waarde van de variabele ugla. Deze transaction heet

* ugnill is een interne poetsconstante met de waarde: -110244.

de "active transaction". Alle transactions in een systeem zijn een of meer keren "active" maar niet tegelijkertijd. Als er zich op een bepaald moment n transactions in een systeem bevinden, is één van die n de active transaction. Alle andere n-1 transactions heten niet-actief.

De algemene regel luidt dat een niet-actieve transaction tot een chain behoort. De active transaction is meestal los.

Het totaal aantal chains is 1 + ugnqueue. Zij zijn als volgt aan de clock-routine en de queues toegewezen:

chain nummer	betekenis
0	eventchain (zie clock-routine)
1	chain van queue 1
2	chain van queue 2
.	
.	
.	
.	
ugnqueue	chain van queue ugnqueue

De indeling van het array ugcc is weergegeven in figuur 3.1.2. op bladzijde 13 en 14.

De volgende procedures opereren op transactions:

a. 3 interne procedures: fromf, tof (la), tosort (la,j,sp);
b. 4 simple-blok-procedures: generate (t, label),
terminate (m),
advance (nat),
transfer (r);
c. 1 hulpprocedure: assign (i,t);
d. 1 outputprocedure: printtransaction.

Figuur 3.1.1.

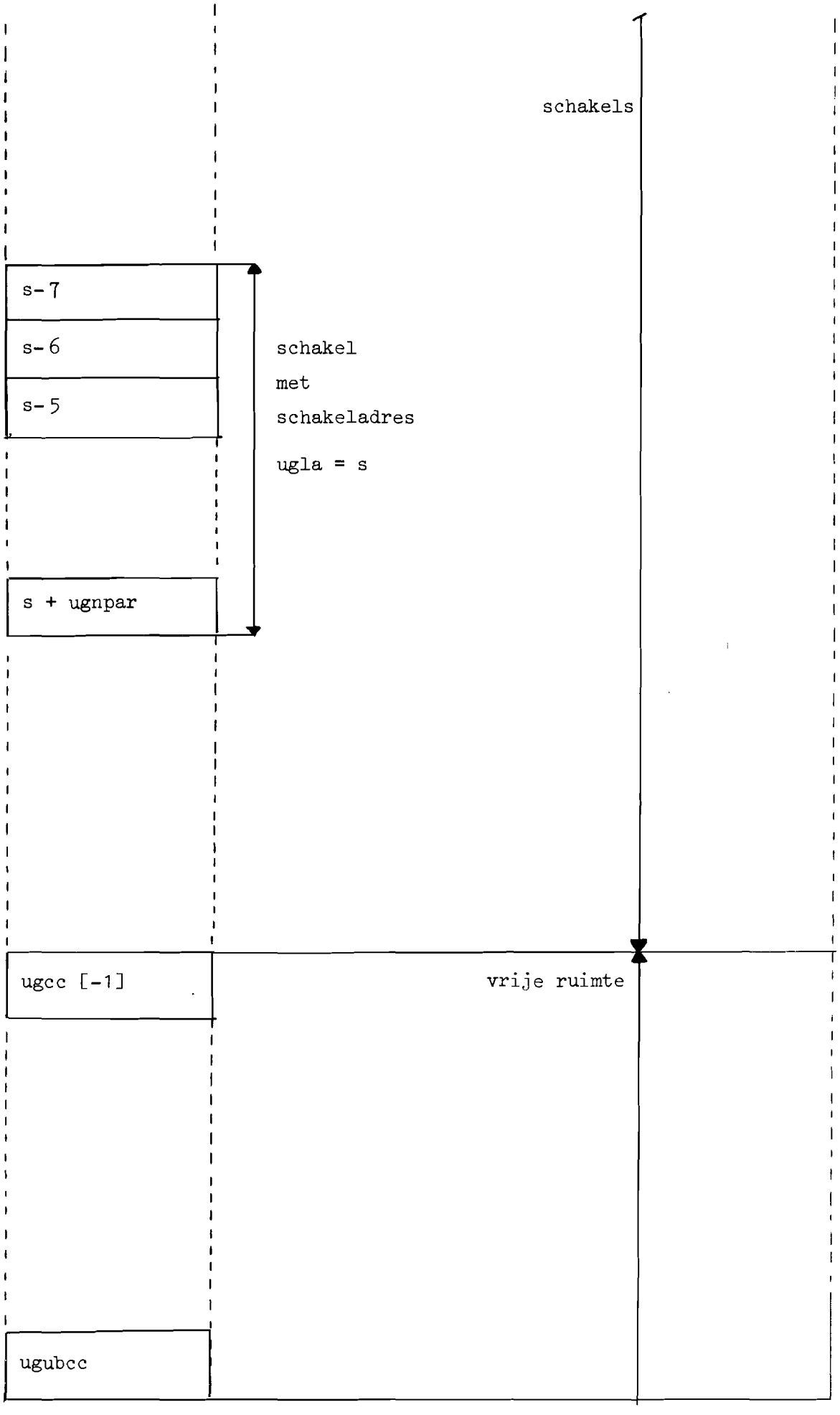
i ugcc [i]

ugla-7	Hulpparameter t.b.v. de storage. Zie procedure enter.
ugla-6	Het nummer van de transaction. De transactions worden in volgorde van generatie genummerd, te beginnen met 1.
ugla-5	Het tijdstip waarop de transaction voor het eerst het model is binnengekomen.
ugla-4	Het labelnummer van het volgende blok dat de transaction moet doorlopen.
ugla-3	Als de transaction tot een chain behoort: het nummer van de chain, als de transaction los is: ugnill.
ugla-2	het adres van de voorgaande schakel in de chain waartoe de transaction behoort of het laatst behoord heeft. Als de transaction de eerste schakel in de chain is dan is de waarde: ugnill.
ugla-1	het adres van de volgende schakel in de chain waartoe de transaction behoort of het laatst behoord heeft. Als de transaction de laatste schakel in de chain is dan is de waarde: ugnill.
ugla	Sorteerparameter. Als een transaction in een chain wordt gesorteerd, wordt de waarde van de grootheid waarop men sorteert, bewaard in de sorteerparameter.

Figuur 3.1.2.

i	ugcc [i]
-5	ugubcc (upper bound of array cc)
-4	ugnpar + 8 = aantal plaatsen per schakel
-3	aantal chains
-2	startindex chain van vrijgekomen schakels (terminated transactions)
-1	startindex vrije ruimte
0	link address laatste schakel in chain 0
1	link address eerste schakel in chain 0
2	aantal schakels in chain 0
3 x j	link address laatste schakel in chain j
3 x j + 1	link address eerste schakel in chain j
3 x j + 2	aantal schakels in chain j
3 x ugcc [-3]	link address laatste schakel in chain ugcc [-3]
3 x ugcc [-3] +1	link address eerste schakel in chain ugcc [-3]
3 x ugcc [-3] +2	aantal schakels in chain ugcc [-3]

Figuur 3.1.2.


schakels
s-7
s-6
s-5
schakel
met
schakeladres
ugla = s
s + ugnpar
ugcc [-1]
vrije ruimte
ugubcc

3.2. Facilities.

Een facility is een entiteit, die op elk moment dienst kan verlenen aan één transaction. Men zegt dat een transaction een facility bezet. Zolang een facility bezet is, moeten andere transactions, die de facility ook willen bezetten, wachten totdat deze vrij is.
Met het gebruik van een facility zijn vier simple-blok-procedures gemoeid, t.w. seize (k, j, sp), release (k), preempt (k,j,sp) en return (k).
Zij stellen de volgende instructies aan een transaction voor:

seize (k,j,sp)	: bezet facility k, indien deze vrij is. Als de facility bezet is ga dan in queue j staan, gesorteerd op sp.
release (k)	: verlaat facility k en geef hem vrij. Het release-simple-blok is de pendant van het seize-simple blok.
preempt (k,j,sp)	: bezet facility k, indien deze tenminste niet reeds is bezet door een preempting transaction. Als dit het geval is, ga dan in queue j staan, gesorteerd op sp. Is de facility bezet door een seizing transaction onderbreek dan de behandeling daarvan.
return (k)	: verlaat facility k en geef hem weer terug aan zijn vorige bezigheid. Het return-simple-blok is de pendant van het preempt-simple-blok.

De facilities worden in algpss voorgesteld door subscripted variables van arrays met lengte nfac. De toestand van een facility wordt volledig bepaald door de waarden van twee variabelen:

ugsla [k] (seizing link address) en
ugpla [k] (preempting link address)

Er zijn vier mogelijke toestanden:

1. not seized, not preempted,
2. seized, not preempted,
3. not seized, preempted,
4. seized and preempted.

De waarde van ugsla [k] en ugpla [k] zijn in de vier gevallen

1. ugsla [k] = nill ugpla [k] = nill.
2. ugsla [k] = link address van de seizing transaction, ugpla [k]= nill.
3. ugsla [k] = nill, ugpla [k]= address van de preempting transaction.
4. ugsla [k] = address seizing transaction,
   ugpla [k] = address preempting transaction.

Als een transaction a een facility verlaat kan een volgende, wachtende, transaction b de facility bezetten. Vóór dat a zijn weg door het systeem vervolgt wordt gekeken of er zo'n transaction b aanwezig is. De transaction b behoort tot een queue. Ten behoeve van het kiezen van de transaction b zijn twee attributen ingevoerd:

fpsq [k] (= facility preferential seize queue)
fppq [k] (= facility preferential preempt queue).

De waarde van deze attributen is het nummer van een van de queues vóór facility k.

Pas nadat eventueel een volgende transaction is aangewezen vervolgt de active transaction zijn weg door het model. Zie verder hoofdstuk 6.2. voor een volledige beschrijving van de gevolgde keuzediscipline en de rol van de preferential queues daarbij.

3.3. Storages.

Een storage is een component, die parallel-processing mogelijk maakt.

Men zou kunnen zeggen dat het een facility is, die meer dan één transaction tegelijkertijd kan behandelen.
Een storage kan bijvoorbeeld een kade voorstellen, waaraan verschillende schepen tegelijkertijd kunnen worden gelost, of een buffer in een telefooncentrale. Storages worden in algpss gerepresenteerd door subscripted variables van arrays met lengte nstor. Met behulp van de definitieprocedure storage (no, cap) wordt de capaciteit opgegeven, d.w.z. het aantal plaatsen (cap) waaruit storage no bestaat.

Er zijn twee simple-blok-procedures:

1. enter (no, apl,j,sp) : bezet apl plaatsen van storage no. Als dit niet mogelijk is, ga dan in queue j staan, gesorteerd op sp.
2. leave (no, apl) : maak apl plaatsen van storage no vrij.

Indien een transaction meer plaatsen wil bezetten dan er in de storage vrij zijn, moet hij wachten totdat er minstens zoveel vrije plaatsen zijn als hij wil bezetten.
Intussen kan een andere transaction, die niet meer plaatsen nodig heeft dan er vrij zijn, wel de storage binnen gaan.
(Zie hoofdstuk 6.3 voor een beschrijving van de gevolgde voorrangsdiscipline).
Op overeenkomstige wijze als bij de facility kent ook de storage een preferente chain waaruit transactions de storage binnengaan.
Het nummer van de preferente chain is de waarde van het standaard attribuut spq [i] (storage preferential queue).
De waarde van spq [i] is gelijk aan het nummer van de queue-chain van één van de queues, die voor de storage geprojecteerd zijn.
De toestand van een storage wijzigt zich als er een transaction binnen komt én als een transaction de storage verlaat.

3.4. Queues.

Transactions, die de toegang tot een blok geweigerd worden (bijvoorbeeld in een seize-blok indien de facility reeds bezet is), worden in queues geplaatst.

Met iedere queue komt een chain overeen, die hetzelfde nummer heeft als de queue: transactions, die tot queue j behoren, bevinden zich in chain j.
Een transaction kan maar van één queue tegelijkertijd lid zijn.

Wenst de gebruiker statistische informatie over bepaalde queues dan dient hij de nummers van die queues op te geven met behulp van de procedure queuedata.

3.5. Tables.

Met behulp van tables kan statistische informatie over procesgrootheden worden verkregen. De opgeslagen informatie wordt in de vorm van een histogram gepresenteerd. De gebruiker geeft via de definitieprocedure table (no, akl, low, klbr) op: het aantal klassen (akl) van het histogram, de klassebreedte (klbr) en de ondergrens van de laagste klasse (low). Met het tabulate-simple-blok wordt de waarde van de grootheid x, waarvan de gebruiker een histogram wenst, bewaard in de bijbehorende table.
De procedure table vult het real array ughistdata [1:ugntable, 1:3] als volgt:

Figuur 3.5.1.

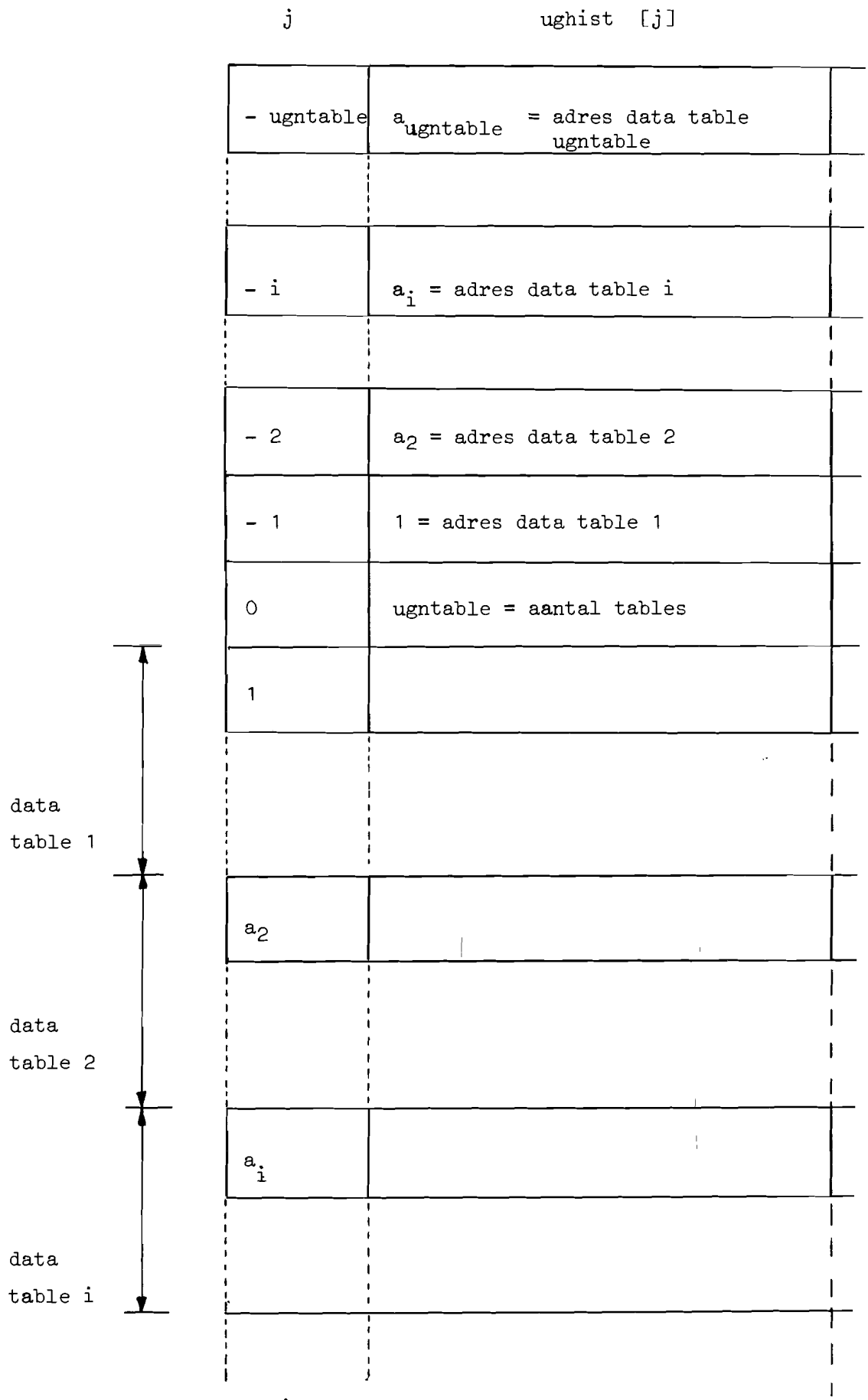
		1	2	3
no ↓	1	akl_1	low_1	$klbr_1$
	2	akl_2	low_2	$klbr_2$
	. . .			
	i	akl_i	low_i	$klbr_i$
	. . .			
	ugntable	akl ugntable	low ugntable	klbr ugntable

akl_i = aantal klassen van table i

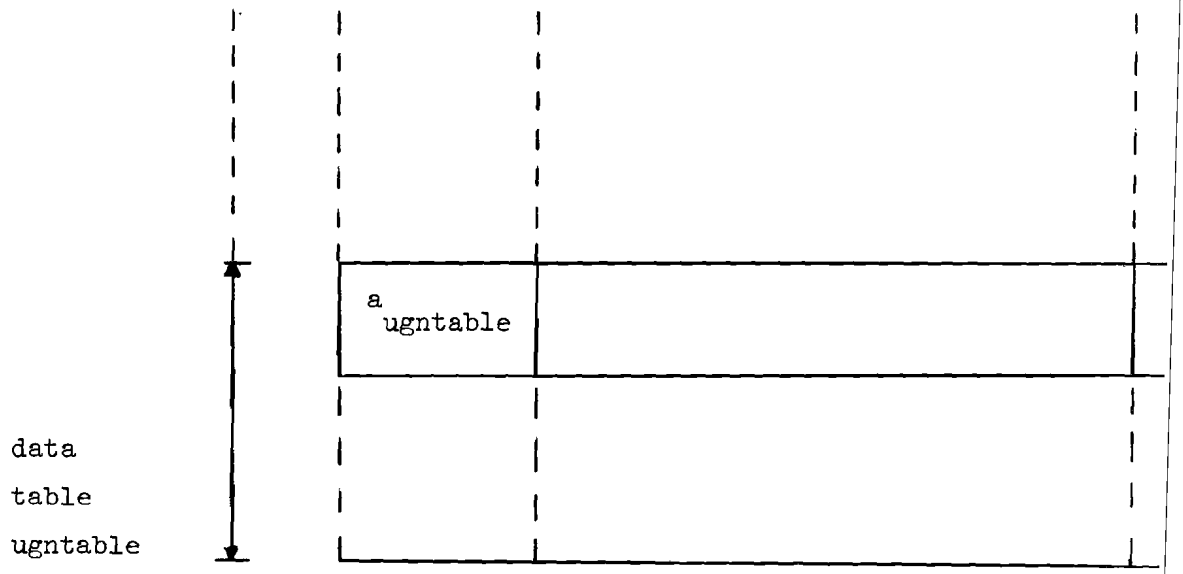
low_i = ondergrens laagste klasse van table i

$klbr_i$ = klassebreedte van table i.

De procedure tabulate (hoofdstuk 6.5) vult de datablokken van het real array ughist [-ugntable : ugubtable] (zie fig. 3.5.2. op blz.20,21 en fig. 3.5.3. op blz. 22).
De procedure printtable (i) zorgt voor het afdrukken van het histogram van table i.


j
ughist [j]
- ugntable
a ugntable = adres data table ugntable
- i
a i = adres data table i
- 2
a2 = adres data table 2
- 1
1 = adres data table 1
0
ugntable = aantal tables
1
data table 1
a2
data table 2
a i
data table i

figuur 3.5.2.



figuur 3.5.2.

Datablok van table i

j	ughist [j]
a_i	totaal aantal entries in table i
$a_i + 1$	minimale waarde van x
$a_i + 2$	maximale waarde van x
$a_i + 3$	Σx
$a_i + 4$	Σx^2
$a_i + 5$	aantal entries kleiner dan de ondergrens
$a_i + 6$	aantal entries in klasse 1
$a_i + 7$	aantal entries in klasse 2
$a_i + 5 + j$	aantal entries in klasse j
$a_i + akl_i + 5$	aantal entries in klasse akl_i
$a_i + akl_i + 6$	aantal entries groter dan hoogste klasse

aantal klassen van table i (akl_i)

fig. 3.5.3.

4. FUNCTIONS.

Voor het doen van aselecte trekkingen uit een verdeling staan er in algpss drie random-generatoren ter beschikking van de gebruiker, die drie verschillende reeksen van pseudo-random getallen genereren met een waarde x waarvoor geldt $0 < x < 1$. Een dergelijk aselect getal wordt verkregen door de aanroep van één van de standaard-attributen: RANDOM 1 en RANDOM 2. De startwaarden van de generatoren worden vastgesteld door aanroep van de initialisatieprocedures SETRANDOM (a), SETRANDOM 1 (a) en SETRANDOM 2 (a), waarin a een geheel getal is.

De verdelingen zelf worden in de vorm van functions gedefinieerd met behulp van de definitieprocedure function (no, ag, type).
De betekenis van de formele parameters is de volgende:

no : het nummer van de function;

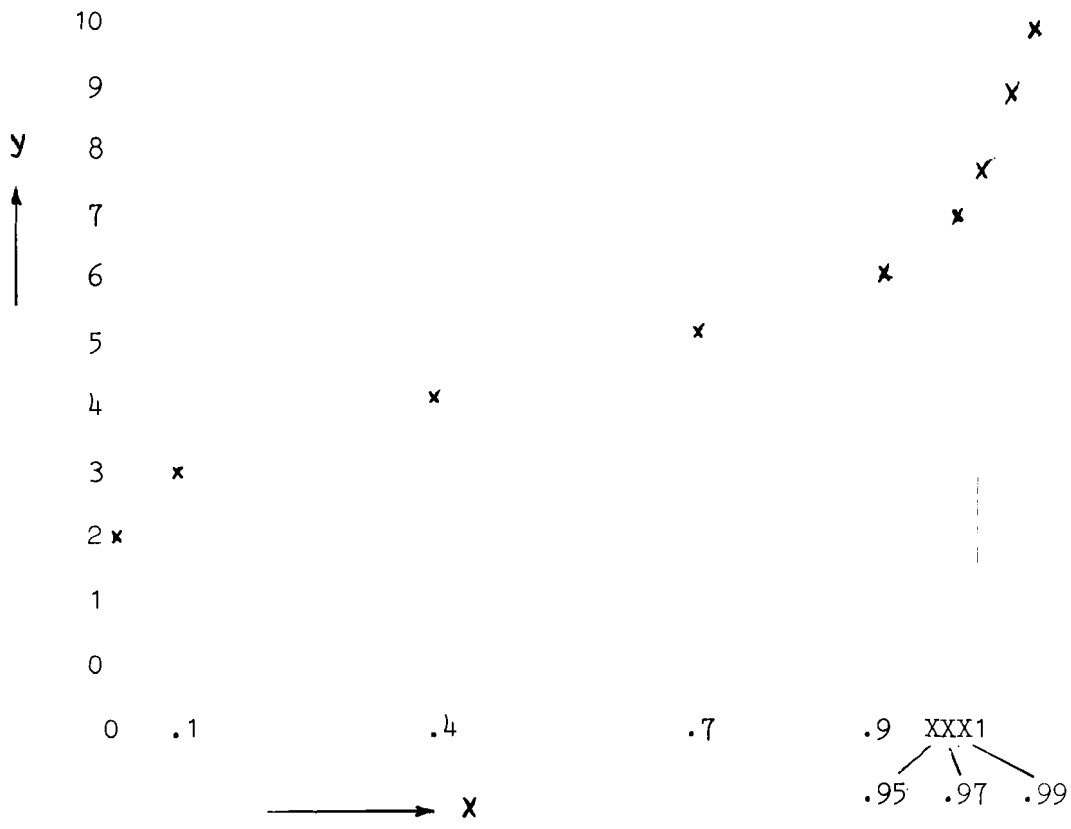
ag : het aantal getallen dat via een inputband wordt gelezen;
ag is een even getal.

type : het type van de function;
als type = 0 is de functie continu,
als type = 1 is de functie discreet.

De getallen, die op de inputband staan, worden geïnterpreteerd als resp. de abscis x en de ordinaat y van de ag/2 coördinaatpunten ($0 < x \leq 1$).

Voorbeeld: Stel dat een functie gedefiniëerd is door function (1,18,type) en de volgende reeks getallen:
0,2 ,.1,3 ,.4,4 ,.7,5 ,.9,6 ,.95,7 ,.97,8 ,.99,9 , 1,10.
dan is de functie bepaald door de volgende coördinaatpunten:
(0,2), (.1,3), (.4,4), (.7,5), (.9,6), (.95,7), (.97,8), (.99,9), (1,10).

In grafische vorm:



De waarde van een functie is de waarde van het standaard-attribuut func (no,x), waarin no het nummer van de function is en x de abscis, waarvoor de bijbehorende ordinaat wordt gezocht. Als de function in het voorbeeld een verdeling voorstelt waaruit men een aselecte trekking wil doen dan wordt de functie bepaald door bijvoorbeeld func (1, RANDOM) d.w.z. x is een pseudo-random getal tussen 0 en 1.
De functiewaarde wordt als volgt berekend:

1. als de functie continu is (type = 0) : door lineaire interpolatie tussen de ordinaten, die behoren bij de abscissen waartussen x ligt.
2. als de functie discreet is (type = 1): de ordinaat, die behoort bij de eerste abscis rechts van x. Als x gelijk is aan

een abscis, dan is de functiewaarde de bijbehorende ordinaat.

Stel dat in het voorbeeld de waarde van het pseudo-random getal is:

x = .25, dan is de functiewaarde:
1. als type = 0 (continu): func (1, .25) = 3,5 , en
2. als type = 1 (discreet):func (1, .25) = 4

De functie-definitie-faciliteit maakt het mogelijk zelf verdelingen te definiëren en daaruit trekkingen te doen. Op het rekencentrum van de T.H.E. heeft de gebruiker echter de beschikking over een groot arsenaal van verdelingen in de vorm van standaardprocedures, zodat hij deze niet zelf (gebrekkig) hoeft te definiëren .
De trekking uit twee veel voorkomende verdelingen kan eenvoudig als volgt worden uitgevoerd:
a. negatief exponentiële verdeling met gemiddelde mu:
   y is de waarde van het standaardattribuut NEGEXP (mu).
b. uniforme verdeling tussen a en b : y = a + (a-b) * RANDOM

Opm. Voor RANDOM mag ook RANDOM 1 of RANDOM 2 worden gelezen.

5. STANDAARDATTRIBUTEN.

Standaardattributen zijn algpss-variabelen, die ter beschikking staan van de gebruiker, d.w.z. hij kan refereren aan de waarde ervan, en op grond daarvan bijv. het verdere verloop van het proces vaststellen. Een overzicht van alle standaardattributen is gegeven op de hierna volgende bladzijden 26, 27, 28 en 29.

	naam	betekenis	type
algemeen:	ugt	systeemtijd (clock-time)	real
blokken:	ugent [j]	het aantal entries in blok j j = 1, 2, ..., ugnblock	integer
transactions:	ugready	het totaal aantal transactions dat in terminateblokken vernietigd is	integer
	par (i)	de waarde van de parameter i : i = 1, 2, ..., npar, par (i) = cc [la + i]	real procedure
	tno	(transaction no): het nummer van de transaction; tno = cc [la - 6]	integer procedure
	tt	(transit time), tt = ugt - cc [la - 5]	real procedure
facilities:	ugfe [k]	het aantal entries in facility k	integer

	ugfs [k]	het nummer van de seizing transaction; als er geen seizing transaction is, is ugfs [k] = 0	integer
	ugfp [k]	het nummer van de preempting transaction; als er geen preempting transaction is, is ugfp [k] = 0	integer
	fpsq [k]	facility preferential seize queue (zie hfdst. 3.2.)	integer
	fppq [k]	facility preferential preempt queue (zie hfdst. 3.2.)	integer
	ugfsc [k]	het tijdstip van de laatste toestandswijziging (status change) van facility k	real
	ugfcti [k]	cumulative time integral: de totale tijd dat de facility gebruikt is geweest (seized en/of preempted) k = 1, 2, ..., ugnfac.	real
storages:	ugse [i]	het aantal entries in storage i	integer
	ugsc [i]	current contents: het aantal bezette plaatsen in storage i	integer
	ugsm [i]	maximum contents: het maximale aantal plaatsen dat bezet is geweest	integer
	ugscap [i]	de capaciteit van storage i (= het aantal plaatsen dat bezet kan worden)	integer
	spq [i]	storage preferential queue (zie hfdst. 4.3.)	integer
	ugssc [i]	het tijdstip van de laatste toestandwijziging (status change) van storage i	real
	ugscti [i]	cumulative time integral = de totale tijd dat de storage gebruikt is geweest, gewogen met het aantal bezette plaatsen i = 1, 2, ..., ugnstor	real

queues:	ugqe [j]	het aantal non-zero entries in queue j	integer
	length [j]	de lengte van queue j	integer procedure
	ugqm [j]	maximum contents: de maximale lengte die queue j heeft gehad	integer
	ugqz [j]	het aantal zero entries: het aantal transactions dat niet heeft hoeven wachten in queue j	integer
	ugqsc [j]	het tijdstip van de laatste toestandwijziging (status change)	real
	ugqcti [j]	cumulative time integral: de totale tijd dat de queue gebruikt is, gewogen met het aantal transactions in de queue	real
	qtp (j,pl,p)	queue transaction parameter: de waarde van parameter p van de transaction die op de pl-de plaats in queue j staat j = 1,2, ..., ugnqueue	real procedure
	x)	Deze attributen zijn alleen beschikbaar voor die queues, waarvoor statistische informatie is gevraagd	
tables:	mean (j)	de gemiddelde waarde van de grootheid, die in table j is opgeslagen	real procedure
	var (j)	de variantie van de grootheid, die in table j is opgelagen	real procedure

functions:	RANDOM	een pseudo-random getal > 0 en ≤ 1	real procedure
	RANDOM 1	idem	
	RANDOM 2	idem	
	NEGEXP	trekking uit een negatief exponentiële verdeling met gemiddelde	
	(mu)	mu. De waarde wordt berekend als: NEGEXP = -muxln (RANDOM)	
	func	de functiewaarde van function no als de abscis x is (zie hfdst.	real procedure
	(no,x)	4.1.)	

6. SIMPLE-BLOK-PROCEDURES.

Alle simple-blok-procedures opereren op de active transaction, d.w.z. op de transaction met link address ugla. Het aanwijzen van de active transaction, dus het bepalen van de waarde van ugla, wordt gedaan door de clock-routine.
Alleen de essentiële operaties van de procedures worden behandeld.

6.1. Transaction-georiënteerde simple-blok-procedures.

6.1.1. generate (te, ln).

Met behulp van de procedure generate (te, ln) worden de transactions gecreëerd. De nieuwe transaction wordt voorgesteld door een schakel van het array ugcc met link address p. De generatie van de transaction p vindt plaats op het moment dat de active transaction met generate-simple-blok binnenkomt. De transaction p komt in de eventchain te staan, gesorteerd op waarde van ugcc [p] (zie interne procedure tosort).
De twee formal parameters hebben de volgende betekenis:

te (type real) : time of entry: de tijd tussen nu en het moment waarop de transaction p het model binnenkomt (ugcc [p] := ugt + te)

ln (type integer) : label number: de waarde van het subscript van de switch s.
De transaction komt met systeem in de toekomst binnen via met blok met label s[ln] . De waarde van ln wordt bewaard in ugcc [p--].

Voorbeeld:

switch s: = A, B, C, D, E;
A: generate (7.8, 3) : De transaction p komt op het tijdstip ugt + 7.8 het systeem binnen via het blok met de label C.

Bij de simulatie van wachttijdproblemen is het meestal gewenst dat de transactions het model binnenkomen met een tussentijd (interarrival time) die een trekking is uit een bepaalde verdeling. In dat geval komen de transactions het model binnen via het generate-blok en zorgen allereerst voor de generatie van hun opvolger.

Voorbeeld:

switch s: = A, B, C,.....;

A: generate (NEGEXP (6),1); transactions komen het systeem binnen via het blok met label A. Hun tussenaankomsttijd is negatief exponentiëel verdeeld met gemiddelde mu = 6.

6.1.2. terminate (m).

De pendant van het generate-simple-blok is het terminate-simple-blok. De procedure terminate (m) zorgt voor de "vernietiging" van de active transaction d.w.z. de schakel la wordt toegevoegd aan de chain van vrijgekomen schakels. De parameter m is van het type integer en geeft aan met hoeveel eenheden het totaal aantal afgewerkte transactions moet worden verhoogd (ugready:= ugready + m). Tevens wordt gekeken of het einde van de simulatie niet reeds is bereikt (ugready = ugntrans) en of het systeem toe is aan een tussentijdse presentatie van output-gegevens (zie 10.2).

6.1.3. advance (nat).

De functie van de procedure advance (nat) is dat de active transaction gedurende een bepaalde tijd op non-actief wordt gezet. De formal parameter nat (non active time) is van het type real en geeft aan hoe lang de transaction niet actief zal zijn. Transaction la wordt in de eventchain gezet, gesorteerd op de waarde van ugcc [ugla] (= ugt + nat).

Voorbeeld: als orders op een machine een bewerkingstijd hebben die uniform verdeeld is tussen 1 en 3 minuten dan kan dit in algpss worden voorgesteld door het blok:

```
label:    advance (1 + 2 * RANDOM);
```

6.1.4. transfer (ln).

De normale weg van transactions door het systeem is van het blok met label s [i] naar het blok met label s [i + 1]. (0<i<ugnblock). De procedure transfer (ln) biedt de mogelijkheid van deze normale gang af te wijken en transactions door te sturen naar een willekeurig ander blok. Het blok waarnaar de transactions gaan is het blok met label s [ln] : de parameter ln (label number) is van het type integer. Hieronder volgen enkele voorbeelden van het gebruik van het transferblok:

a. absolute transfer-opdracht:
   B: transfer (12); ga verder bij het blok met label s [12].

b. conditionele transferopdracht:

   D: if fs [2] ≠ 0 then transfer (3) else transfer (7);

   als facility 2 bezet is door een seizing transaction ga dan naar het blok met label s [3]. Is dit niet het geval, ga dan verder bij het blok met label s [7].

   C: if RANDOM <.4 then transfer (8) else transfer (11).

   40% van de transactions gaat verder bij blok 8,
   60% bij blok 11.

6.2. Facility-georiënteerde simple-blok-procedures.

De formal parameter k in de procedure seize, release, preempt, en return is van het type integer en stelt het nummer van de facility voor waarop de procedure betrekking heeft. De parameter j (integer) en sp (real) in seize en preempt stellen respectievelijk voor het nummer van de queue, waarin de transactions komen te staan als de toegang tot de facility geweigerd wordt, en de grootheid waarop zij in de queue worden gesorteerd.

6.2.1. seize (k, j, sp).

Wat betreft de toestand van de facility onderscheiden we twee gevallen:

1. de facility is bezet (seized en/of preempted),
2. de facility is vrij (not seized, not preempted).

De gevolgde procedure is in de twee gevallen verschillend en wel:

ad 1. De transaction wordt de toegang tot de facility geweigerd, en komt in queue j te staan.
Het programma wordt vervolgd met de clock-routine.

ad 2. De transaction bezet de facility (ugsla [k] : = ugla) en gaat door naar het volgende blok.

Het bepalen van de facility preferential seize queue (fpsq [k] kan op drie manieren gebeuren:

1. Vóórdat de eerste transaction de facility bezet, is de preferential seize queue onbepaald, d.w.z. fpsq[k] = ugnill.

2. Indien de preferential seize queue onbepaald is (fpsq[k] = ugnill) wordt queue j de preferential seize queue bij binnenkomst van een transaction in het seize-simple-blok.

3. De gebruiker kan op elk moment de preferential seize queue zelf vaststellen of onbepaald maken.

N.B. Als een transaction toegang probeert te krijgen tot een facility en queue j is niet de preferential seize queue, wordt hem de toegang altijd geweigerd, ook al is de facility vrij.

6.2.2. release (k).

Alléén de transaction die via een seize-blok de facility bezet heeft, kan de facility weer vrij maken. Zou een andere transaction dit trachten te doen dan volgt de foutmelding "release error 1" en stopt het programma.
Voor de verdere operaties van release (k) onderscheiden we twee situaties:
1. Er bevinden zich geen transactions in de preferential seize queue.
2. Er is minstens één transaction in de preferential seize queue.

ad. 1 De facility wordt vrijgegeven (ugsla[k] : = ugnill).
De transaction gaat door naar het volgende blok.

ad.2. De facility wordt vrijgegeven (ugsla[k] : = ugnill) en de eerste transaction in de preferential seize queue wordt in staat gesteld het seize-blok binnen te gaan. Daartoe gebeurt het volgende.

a. aan de globale variabele ugnext wordt het link-address van die transaction toegekend.
b. de active transaction komt vooraan in de eventchain te staan
c. het programma wordt vervolgd met de clock-routine.

Opmerking

Als de preferential seize-queue leeg is, betekent dit nog niet dat er geen transactions zijn, die staan te wachten om via een seize-blok de facility te bezetten. Dit is namelijk mogelijk als zij zich in een queue bevinden die niet de preferential seize-queue is. Zij kunnen

echter pas aan de beurt komen als fpsq[k] op de juiste wijze is veranderd. (zie voorbeeld van de job-shop, hfdst. 11.2).

6.2.3. preempt. (k,j,sp).

Er worden drie verschillende toestanden van de facility onderscheiden:

1. not seized, not preempted (ugsla[k] = ugpla[k] = ugnill),
2. seized, not preempted (ugsla[k] ≠ ugnill, ugpla[k] = ugnill),
3. preempted (ugpla[k] ≠ ugnill).

De gevolgde procedure in de drie gevallen is:

ad 1. De transaction bezet de facility (ugpla[k] : = ugla) en gaat door naar het volgende blok.

ad 2. De facility wordt bezet (ugpla[k] . = ugla). Als de seizing transaction p zich in een chain bevindt, wordt hij daar uitgehaald. Is deze chain de eventchain, dan komt de resterende tijd, die de transaction nog in de eventchain zou moeten doorbrengen, in ugcc [p] te staan.
(Dit is de enige uitzondering op de regel, dat een niet-actieve transaction tot een chain behoort). De active transaction gaat door naar het volgende blok.

ad 3. De transaction wordt de toegang tot de facility geweigerd en komt in queue j te staan. Het programma wordt vervolgd bij de clock-routine.

Het vaststellen van de facility preferential preempt queue (fppq[k]) geschiedt op analoge wijze als het bepalen van fpsq[k], nl. op de volgende drie manieren:

1. Vóórdat de eerste transaction het preempt-simple-blok binnengaat, is de preferential preempt queue onbepaald, d.w.z. fppq[k] = ugnill.

2. In het geval dat de preferential preempt queue onbepaald is, wordt queue j de preferential preempt queue bij binnenkomst van een transaction in het preempt-simple-blok.
3. Op elk moment kan de gebruiker zelf de preferential preempt queue vaststellen of onbepaald maken.

N.B. Als een transaction, het preempt-simple-blok probeert binnen te gaan en queue j is niet de preferential preempt queue, dan wordt de toegang altijd geweigerd, ook al is de facility vrij.

6.2.4. return (k).

Alléén de transaction, die via een preempt blok bezit heeft genomen van de facility, kan deze weer teruggeven aan diens vorige bezigheid. Indien een andere transaction dit zou proberen te doen, volgt de foutmelding "return error 1" en stopt het programma.

Voor een verdere beschrijving van de procedure return (k) onderscheiden we drie gevallen:

1. Er is minstens één transaction in de preferential preempt queue.
2. Er bevinden zich geen transactions in de preferential preempt queue. Er is geen transaction waarvan de behandeling onderbroken is (ugsla [k] = ugnill).
3. Er bevinden zich geen transactions in de preferential preempt queue. Er is een transaction, waarvan de behandeling onderbroken is (ugsla [k] ≠ ugnill).

ad 1. De toestand van de facility wordt: not preempted (ugpla [k] : = ugnill). De eerste transaction in de preferential preempt queue wordt in staat gesteld het preempt-blok binnen te gaan. Daartoe gebeurt het volgende:

a. Aan de variabele ugnext wordt het link-address van de eerste transaction toegekend.
b. De active transaction komt vooraan in de eventchain te staan.
c. Het programma wordt vervolgd bij de clockroutine.

ad 2. De toestand van de facility wordt: not seized, not preempted (ugpla [k] : = ugnill). Gekeken wordt nu of er misschien transactions zijn die staan te wachten om de facility via een seize-blok binnen te gaan. Als er zich geen transactions in de preferential seize queue bevinden, gaat de active transaction door naar het volgende blok. Is er minstens één transaction in de preferential seize queue dan gebeurt het volgende:

a. Aan de variabele ugnext wordt het link-address van de eerste transaction in de preferential seize queue toegekend.
b. De active transaction komt vooraan in de eventchain te staan.
c. Het programma wordt vervolgd met de clockroutine.

ad 3. De toestand van de facility wordt: seized, not preempted. De transaction p, waarvan de behandeling onderbroken is, wordt teruggezet in de chain, waaruit hij door een preempting transaction is verwijderd. Is die chain de eventchain dan is het tijdstip t waarop hij daar in de toekomst uit mag: t = ugt + ugcc [p] (de waarde van ugcc [p] is in het preempt-simple-blok vastgesteld).
De active transaction gaat door naar het volgende blok.

6.3. Storage-georiënteerde simple-blok-procedures.

6.3.1. enter (no, apl, j, sp).

De eerste drie formal parameters zijn van het type integer, no is het nummer van de storage en apl het aantal plaatsen dat de transaction in de storage wil bezetten;

j is het nummer van de queue, waarin de transaction komt te staan als de toegang tot de storage wordt geweigerd. De parameter sp is van het type real en stelt de grootheid voor waarop een transaction eventueel in queue j wordt gesorteerd.

De operaties zijn verschillend in vier onderscheiden gevallen, die hierna volgen.

1. Het aantal vrije plaatsen is kleiner dan apl.
   De transaction wordt niet toegelaten tot de storage.
   Het programma gaat verder met de clockroutine.
2. Het aantal vrije plaatsen is gelijk aan apl.
   De transaction gaat de storage binnen, bezet apl plaatsen en gaat door naar het volgende blok.
3. Het aantal vrije plaatsen is groter dan apl.

   De transaction gaat de storage binnen en bezet apl plaatsen. Er wordt nu gekeken of er zich in de storage preferential queue transactions bevinden, die de storage kunnen binnengaan. Daartoe wordt deze queue van voren naar achteraan afgezocht naar de aanwezigheid van een transaction p, die een aantal plaatsen wil bezetten (ugcc [p - 7]), dat kleiner is dan of gelijk is aan het aantal vrije plaatsen. Zo gauw zo'n transaction gevonden is, gebeurt het volgende:

a. Aan de variabele ugnext wordt het link-address van die transaction toegekend.
b. De active transaction komt vooraan in de eventchain te staan.
c. Het programma gaat verder bij de clockroutine.

Wordt er geen transaction p gevonden dan gaat de active transaction door naar het volgende blok.

Het vaststellen van de storage preferential queue, die een soortgelijke functie heeft als de facility preferential queues, gebeurt op drie manieren:

1. Vóórdat de eerste transaction het enter-simple-blok binnengaat is de storage preferential queue onbepaald (spq [no] = ugnill).
2. Indien spq [no] = ugnill wordt queue j de storage preferential queue bij binnenkomst van een transaction in het enter-simple-blok.
3. De gebruiker kan zelf op elk moment de waarde van spq [no] wijzigen.

N.B. Transactions, die zich in een queue bevinden, die niet de storage preferential queue is, worden altijd geweigerd, ook al zouden er voldoende vrije plaatsen zijn.

6.3.2. leave (no, apl).

De formal parameters zijn van het type integer; no is het nummer van de storage en apl het aantal plaatsen dat door de transaction wordt vrijgemaakt. Als een transaction een storage verlaat en meer plaatsen wil vrijmaken dan er bezet zijn (apl < ugsc [no]), volgt de foutmelding "leave error 1" en stopt het programma. Is dit niet het geval dan worden door de active transaction apl plaatsen vrij gemaakt.
In de storage preferential queue wordt nu gezocht naar een transaction p, zoals die beschreven is in 6.3.1. Vindt het programma zo'n transaction dan wordt aan de variabele ugnext het link-address van deze transaction toegekend. De active transaction komt dan vooraan in de eventchain te staan en het programma wordt vervolgd bij de clockroutine. Als er geen transaction p gevonden wordt gaat de active transaction door naar het volgende blok. Het is mogelijk dat in het laatste geval er wel transactions staan te wachten om de storage binnen te gaan, maar in een andere queue dan de storage preferential queue. Deze komen echter pas aan de beurt als spq [no] op de juiste wijze is veranderd.

6.4. Table-georiënteerde simple-blok-procedures.

6.4.1. tabulate (j, x, nox).

De formal parameters j en nox zijn van het type integer; j is het nummer van de table, nox is het aantal eenheden waarmee de telling in de juiste klasse van table j moet worden verhoogd.
De formal parameter x is van het type real; x is de grootheid, die getabelleerd wordt.

7. OVERIGE PROCEDURES.

7.1. Interne procedures.

7.1.1. fromf.

De procedure levert een nieuwe schakel op, die van de chain van vrijgekomen schakels is gehaald, of die gemaakt is uit de vrije werkruimte van array ugcc. De waarde van fromf is het link address van de nieuwe schakel.

7.1.2. fromch (la, j).

De schakel met link address la wordt uit chain nummer j gehaald. De transaction heet nu "los".

7.1.3. tosort (la, j, sp).

De schakel met link address la komt in chain j te staan, gesorteerd op de waarde van sp. Deze waarde wordt bewaard in ugcc [la].
De schakels s in de chain worden zodanig gesorteerd dat de waarden van ugcc [s] een niet dalende rij vormen, te beginnen vooraan in de chain.
Is er in de chain een schakel b waarvoor geldt:
ugcc [b + sp] = ugcc [la + sp] dan komt de schakel achter schakel b te staan.

7.1.4. errordata.

Errordata wordt aangeroepen indien zich een fout voordoet. De procedure verzorgt het afdrukken van informatie betreffende het array cc.

7.1.5. jumpout.

Indien zich ergens een ontoelaatbare fout voordoet wordt de procedure jumpout aangeroepen, die het einde van de executie forceert.

7.1.6. queuein (j), queueout (j).

Deze procedures bevatten de statements voor het bijhouden van queue-statistieken. Zij worden aangeroepen voor die queues, waarvoor statistiek gevraagd is d.m.v. de initialisatie-procedure queuedata.

7.2. Hulpprocedures.

7.2.1. assign (i, data).

De formal parameter i (type integer) is het nummer van de transaction-parameter; i = 1, 2, ..., npar;
data is van het type real en stelt het reële getal voor dat in parameter i bewaard moet worden.
Berekening: ugcc [ugla + i] : = data.

7.3. Definitieprocedures.

7.3.1. table (no, akl, low, klbr).

De procedure definiëert de table no door het opgeven van de karakteristieken:
akl : het aantal klassen waarin de table verdeeld is,
low : de ondergrens van de laagste klasse,
klbr : de klassebreedte.

7.3.2. storage (no, cap).

De procedure definiëert de storage no door het vaststellen

van de capaciteit, d.w.z. het aantal plaatsen in de storage dat maximaal bezet kan worden.

7.3.3. function (no,ag,type).

De functie no wordt gedefiniëerd door het aantal getallen (ag) op de inputband en het type: continu (type = 0) of discreet (type = 1).

7.4. Initialisatieprocedures.

7.4.1. setalgpss (ugnblock, ugnfac, ugnstor, ugnqueue, ugntable, ugnfunc, ugnpar, ugntrans, ugnprint).

Voor het vaststellen van de benodigde geheugenruimte die de entiteiten in het model vragen, is een aanroep van de procedure setalgpss vereist. De formal parameters zijn allen van het type integer. Hun betekenis is:

ugnblock : het aantal blokken,
ugnfac : het aantal facilities,
ugnstor : het aantal storages,
ugnqueue : het aantal queues,
ugntables : het aantal tables,
ugnfunc : het aantal functions,
ugnpar : het aantal transaction parameters.

Verder wordt met behulp van ugntrans opgegeven voor hoeveel transactions de simulatie wordt uitgevoerd en met behulp van ugnprint, na hoeveel afgewerkte (terminated) transactions steeds de output-procedure printout moet worden aangeroepen.

7.4.2. SETRANDOM (x), SETRANDOM 1 (x), SETRANDOM 2 (x).

Deze procedures bepalen de startwaarden van de drie RANDOM-generatoren. In het algpss-programma is voorzien in de aan-

roepen
SETRANDOM (1), SETRANDOM1 (1) en SETRANDOM2 (1).

7.4.3. initialize (te, ln).

De simulatie begint op het moment dat de eerste transaction voor het eerst een blok binnenkomt.
Deze transaction is door de clockroutine uit de eventchain gehaald, en doorgestuurd naar dat blok.
Er moet zich dus vóór de start van de simulatierun minstens één transaction in de eventchain bevinden. Het plaatsen van de eerste transaction in de eventchain geschiedt door middel van een aanroep van de procedure initialize. De formal parameters zijn van het type integer; te is het tijdstip (= de waarde van ugt) waarop de transaction het model voor het eerst zal binnenkomen en ln is het labelnummer van het blok, waarin de eerste binnenkomst zal plaatsvinden.

7.4.4. queuedata.

Van de queues, waarvan op de inputband de nummers worden gelezen, zal tijdens de simulatie gegevens worden bijgehouden.

7.5. outputprocedures.

7.5.1. printout.

De procedure printout verzorgt het printen van de output, zoals die gegeven is in hoofdstuk . Het programma roept de procedure zelf aan als simulatie is beëindigd en eventueel tussentijds, als ugnprint ≠ ugntrans (zie procedure setalgpss).

7.5.2. printtable (j).

Bij aanroep van printtable (j) wordt het histogram van table j afgedrukt. In de procedure printout is voorzien in de aanroep van deze procedure voor alle gebruikte tables.

7.5.3. printtransaction.

Voor het opsporen van fouten in het model staat de procedure printtransaction ter beschikking. Bij aanroep wordt van de active transaction de inhoud van alle schakelplaatsen afgedrukt.

7.5.4. job (string).

Ter identificatie van het simulatieprobleem kan met behulp van de procedure job tekst worden geprint.

8. INTERNE ORGANISATIE.

8.1. Blok- en modeldefinitie.

Bij de opbouw van het model in algpss is zeer veel toegestaan. Er zijn echter bepaalde syntactische regels waaraan de gebruiker zich dient te houden. Een poging de syntactische correcte opbouw van het algpss model zo exact mogelijk te definiëren volgt hieronder. De definitie wordt gedeeltelijk gegeven in de vorm van een quasi-uitbreiding van de syntax van algol-60.

< protected identifier > :: = SETRANDOM/SETRANDOM1/
SETRANDOM2/RANDOM/RANDOM1/
RANDOM2/NEGEXP/queuein/
queueout/job/setalgpss/jump-
out/fromf/fromch/queuedata/
tosort/errordata/initialize/
table/storage/function/
generate/terminate/advance/
assign/transfer/mark/seize/
release/preempt/return/
enter/leave/tabulate/
printtransaction/printtable/
length/printout/par/tno/qtp/
tt/mean/var/func/
elke identifier die begint
met de letters ug (underground).

Opm. fpsq, fppq en spq zijn geen protected identifiers.

Verbale definities:

legal identifier : elke identifier, die geen protected identifier is en die ook niet fpsq, fppq of spq is.

legal declaration	:	elke declaration, waarin als identifier een legal identifier wordt gebruikt.
legal statement	:	1. aanroep van een hulp-procedure. 2. aanroep van een output-procedure. 3. elke statement, die niet het effect heeft dat een variabele, aangeduid door een protected identifier, wordt veranderd.
simple-blok	:	de aanroep van een simple-blokprocedure.
label s [i]	:	het label dat op de i-de plaats in switch s gedeclareerd is.

< simple instruction > :: = < legal statement > | < empty > .

< instruction > :: = < simple instruction > |
< instruction >
< simple instruction > .

< blok > :: = < instruction > < simple-blok >
< instruction > .

< algpss model > :: = < label s [1]> : < blok >
< label s [2]> : < blok > ....
..... < label s [nblock]>:
< blok > .

9. GEBRUIK VAN ALGPSS.

De gehele algol-tekst van algpss (zie appendix 2) staat op magneetband, verdeeld in 40 groepen. De algol-block-structuur is als volgt:

groep	inhoud
0 t/m 9	begin declaraties statements
10 t/m 19	begin declaraties statements
20	begin declaraties statements begin
21 t/m 38	declaraties
39	(statements) clockroutine
40	end end end end

9.1. Layout.

In fig. 9.1. op blz. 50 is de layout van een programma schematisch weergegeven. Tussen de aanroepen van mtape 355, waarop zich de algpss-algoltekst bevindt, is aangegeven welke declaraties en statements verplicht of toegestaan zijn. Onder het hoofd "toegestaan" zijn echter alleen die declaraties en statements vermeld, die op de betreffende plaats ook zinvol zijn en/of op die plaats het beste kunnen worden uitgevoerd.

9.2. Output.

Het printen van output geschiedt behalve door de output-procedures ook door de procedure setalgpss, namelijk betreffende de aantallen entiteiten en door de procedure function, die de x- en y-waarden van alle coördinaatpunten afdrukt.
De vorm, waarin de output gepresenteerd wordt, kan het beste worden verduidelijkt aan de hand van de voorbeelden (hoofdstuk 11).

9.3. Foutmeldingen.

Het programma algpss is op de belangrijkste plaatsen beschermd tegen fouten, die een ongewenst verloop van het simulatieproces ten gevolge zouden hebben. Na een foutmelding stopt het programma en wordt zoveel mogelijk output gegeven. Een alfabetische lijst van foutmeldingen en hun betekenis is te vinden op blz. 51 en 52.

Figuur 9.1.

Layout van het simulatieprogramma.

mtape 355, 0-9;
verplicht : aanroep van setalgpss toegestaan : aanroep van job, SETRANDOM, SETRANDOM1, SETRANDOM2.
mtape 355, 10-19;
verplicht : 1. aanroep van initialize 2. voor zover van toepassing: aanroep van table, storage, function. toegestaan : meer aanroepen van initialize.
mtape 355, 20;
verplicht : declaratie van de switch s. toegestaan : legal declarations (zie hoofdstuk 8).
mtape 355, 21-38;
toegestaan : legal statements (zie hoofdstuk 8) met uitzondering van de aanroep van een hulpprocedure en/of een outputprocedure.
mtape 355, 39;
verplicht : het algpss model (zie hoofdstuk 8).
mtape 355, 40;

FOUTMELDING	BETEKENIS
advance error	actual parameter is kleiner dan 0.
array ugcc too small	het is niet meer mogelijk transactions te genereren. Misschien schuilt er een fout in het model. Eventueel kan array ugcc worden vergroot door aan ugubcc een waarde groter dan 10.000 toe te kennen.
assign error	actual parameter 1 is kleiner dan 1 of groter dan ugnpar.
enter error 1	actual parameter 1 is kleiner dan 1 of groter dan ugnstor.
enter error 2	actual parameter 2 is kleiner dan 0.
enter error 3	actual parameter 3 is kleiner dan 1 of groter dan ugnqueue.
func error	actual parameter 1 is kleiner dan 1 of groter dan ugnfunc.
function error 1	actual parameter 1 is kleiner dan 1 of groter dan ugnfunc.
function error 2	actual parameter 2 is oneven.
function error 3	actual parameter 3 is niet 0 of 1 (type onbekend).
generate error 1	actual parameter 1 is kleiner dan 0.
generate error 2	actual parameter 2 is kleiner dan 1 of groter dan ugnblock.
initialize error	actual parameter 2 is kleiner dan 1 of groter dan ugnblock.
leave error 1	actual parameter 1 is kleiner dan 1 of groter dan ugnstor.
leave error 2	actual parameter 2 is groter dan het aantal bezette plaatsen in de storage.
length error	actual parameter is kleiner dan 1 of groter dan ugnqueue.
mean error	actual parameter is kleiner dan 1 of groter dan ugntable.
par error	actual parameter is kleiner dan 1 of groter dan ugnpar.
preempt error 1	actual parameter 1 is kleiner dan 1 of groter dan ugnfac.
preempt error 2	actual parameter 2 is kleiner dan 1 of groter dan ugnqueue.

printtable error	actual parameter 1 is kleiner dan 1 of groter dan ugntable.
qtp error 1	actual parameter 1 is kleiner dan 1 of groter dan ugnqueue.
qtp 2	actual parameter 2 is kleiner dan 0.
qtp error 3	actual parameter 3 is kleiner dan 0.
release error 1	actual parameter 1 is kleiner dan 1 of groter dan ugnfac.
release error 2	de releasing transaction is niet dezelfde als de seizing transaction.
return error 1	actual parameter 1 is kleiner dan 1 of groter dan ugnfac.
return error 2	de returning transaction is niet dezelfde als de preempting transaction.
seize error 1	actual parameter 1 is kleiner dan 1 of groter dan ugnfac.
seize error 2	actual parameter 2 is kleiner dan 1 of groter dan ugnqueue.
setalgpss error	één van de actual parameters is kleiner dan 0.
storage error 1	actual parameter 1 is kleiner dan 1 of groter dan ugnstor.
storage error 2	actual parameter 2 is kleiner dan 0.
table error 1	actual parameter 1 is kleiner dan 1 of groter dan ugntable.
table error 2	actual parameter 2 en/of 3 is kleiner dan 0.
tabulate error 1	actual parameter 1 is kleiner dan 1 of groter dan ugntable.
tabulate error 2	actual parameter 3 is kleiner dan 0.
terminate error	actual parameter is kleiner dan 0.
transfer error	actual parameter is kleiner dan 1 of groter dan ugnblock.
var. error	actual parameter is kleiner dan 1 of groter dan ugntable.

10. NABESCHOUWING.

Het uitgangspunt van het afstudeerwerk was een applicatie-programma te construeren dat dezelfde mogelijkheden zou bieden als GPSS/360.
Het ligt voor de hand een beoordeling van algpss te geven op grond van een vergelijking met GPSS. Hierbij treden de volgende verschillen op de voorgrond.

1. GPSS bevat meer entiteiten, waaronder een groter aantal simple-bloks, in GPSS "blocks" geheten.
2. De nauwkeurigheid in de representatie van getallen is in algpss (c.q. de EL/X-8) beter dan in GPSS. In GPSS is de nauwkeurigheid afhankelijk van de ordegrootte van het getal.
3. Algpss kan vrij eenvoudig worden uitgebreid en/of gewijzigd.
4. Algpss geeft volledige klaarheid over de wijze waarop ("onder water") de eventbehandeling plaatsvindt, GPSS blijft op dit punt vaag.
5. De complexiteit van het model in GPSS stijgt sneller dan in algpss naar gelang de complexiteit van het te simuleren systeem toeneemt.
6. Algpss kan worden gebruikt op iedere computer die een algol-compiler heeft, voor GPSS is een aparte compiler nodig.
7. De aantallen entiteiten, die in een GPSS-programma gebruikt kunnen worden, worden al gauw beperkt door een te kleine geheugencapaciteit van de rekenmachine.

ad. 1. Dit is een schijnbaar voordeel van GPSS. De grotere verscheidenheid van entiteiten karakteriseert tevens de beperktheid van het programma. Het aantal verschillende blokken in GPSS is namelijk vergeleken met algpss veel kleiner. Een blok in GPSS komt overeen met een simple-blok in algpss. Een blok in algpss kan op een zeer groot aantal verschillende manieren worden geconstrueerd.

ad. 2. De grootte van de getalwaarden in GPSS dient zorgvuldig gekozen te worden om ongewenste berekeningen te vermijden. De oorzaak hiervan is gelegen in het feit, dat in GPSS alle getallen integers zijn en dat elk resultaat van een berekening dus wordt afgerond.
Een voorbeeld mag het gevaar van deze vorm van getalrepresentatie verduidelijken:

Zou men in GPSS een uniforme verdeling definiëren tussen 1 en 3 tijdseenheden en daaruit randomtrekkingen doen dan is het resultaat 1, 2 of 3 met de volgende percentages van voorkomen:
een 1 : 25%; een 2 : 50% en een 3 : 25%.
Zo'n verdeling kan dan moeilijk meer uniform worden genoemd, een betere naam is misschien : normaal.

ad. 3. Onder ad. 1 is de grotere starheid van GPSS ten opzichte van algpss reeds aangestipt. Het is in GPSS niet mogelijk buiten de scope van de bestaande entiteiten te gaan. Zou men bijvoorbeeld in GPSS proberen het job-shop-probleem te simuleren dan is het niet mogelijk zoiets als de procedure nexttype te definiëren.

ad. 5. Dit is te verklaren uit de grotere flexibiliteit die algpss heeft.

ad. 7. Voor bijvoorbeeld een 64k-machine bestaan de volgende maximumaantallen:

transactions	: 200
facilities	: 35
storages	: 35
queues	: 70
tables	: 15
functions	: 20
blokken	: 120

Voor algpss zullen deze aantallen in de praktijk vrijwel onbeperkt zijn, d.w.z. het zal nauwelijks voorkomen dat de geheugenruimte voor een algpss-programma te klein

is.
In algpss geeft men voor elke entiteit op: het aantal dat men wenst. Voor het opgegeven aantal wordt geheugenruimte vrij gemaakt.
De GPSS-compiler echter claimt altijd zoveel geheugenruimte als nodig is voor het maximale aantal.

Conclusie.

Algpss blijkt in veel opzichten aantrekkelijker te zijn dan GPSS/360.
Dit voordeel geldt vooral voor de geoefende gebruiker, die algol beheerst. GPSS is algemener, maar voor ingewikkelder problemen, steeds moeilijker te gebruiken.
Een uitbreiding van algpss kan wenselijk zijn, indien daaruit tenminste niet de bezwaren voortvloeien die kleven aan een algemeen standaardprogramma zoals GPSS. De huidige uitvoering van algpss biedt mijns inziens echter voorlopig voldoende mogelijkheden. In de praktijk zal de noodzaak of het gemak van een uitbreiding moeten blijken. Dit is afhankelijk van het type probleem dat men wil simuleren.

11. APPENDIX 1.

11.1. Dokterspraktijk.

Voor het in de inleiding aangehaalde voorbeeld van de dokterspraktijk maken we de volgende specificaties:

1. De tussenaankomsttijden zijn negatief exponentiëel verdeeld met gemiddelde 5.
2. Van de patiënten die langer dan 10 minuten hebben gewacht, wordt, als zij aan de beurt zijn, 30% naar huis gestuurd.
3. Er wordt een histogram gemaakt van de wachttijden van behandelde patiënten.
4. Het aantal weglopers is het aantal entries in blok 16.
5. De consulttijd is een trekking uit de verdeling die voorgesteld wordt door function 1.
6. Telefoongesprekken arriveren met een negatief exponentiëel verdeelde tuusentijd met gemiddelde 6.
7. De gespreksduur is uniform verdeeld tussen 1 en 3 minuten.

```
  1 MIABE 368, 0-9;
 84 JOB(£DOKTERSPRAKTIJK$);
 85 SETALGPSS(15,1,0,2,1,1,0,1000,500);
 86 MIABE 368,10-20;
265 INITIALIZE(0,1);
266 INITIALIZE(0,8);
267 TABLE(1,30,0,2);
268 FUNCTION(1,18,0);
269 QUEUEDATA;
270 MIABE 368,21;
301 SWITCH S:= PATIENTEN,DOKTER,WEGLOPERS,WACHTTIJDNOTITIE,CONSULT,DOKTER VRIJ 1,WEG,
302           TELEFOONGESPREKKEN,BEZETTOON,OPROEP,GESPREK,HAAK NEER,EINDE GESPREK,DOKTER VRIJ 2,WEGLOPERS WEG;
303 MIABE 368,22-40;
737 PATIENTEN:                  GENERATE(NEGEXP(5),1);
738 DOKTER:                          SEIZE(1,1,NOW);
739 WEGLOPERS:                 IF TT>10^RANDOM<.3 THEN TRANSFER(14) ELSE TRANSFER(4);
740 WACHTTIJDNOTITIE:          TABULATE(1,TT,1);
741 CONSULT:                         ADVANCE(FUNC(1,RANDOM));
742 DOKTER VRIJ 1:             RELEASE(1);
743 WEG:                       TERMINATE(1);
744 TELEFOONGESPREKKEN:        GENERATE(NEGEXP(6),8);
745 BEZETTOON:                 IF FP[1]=0 THEN TRANSFER(10) ELSE TRANSFER(13);
746 OPROEP:                          PREEMPT(1,2,0);
747 GESPREK:                         ADVANCE(1+2*RANDOM);
748 HAAK NEER:                 RETURN(1);
749 EINDE GESPREK:            TERMINATE(0);
750 DOKTER VRIJ 2:            RELEASE(1);
751 WEGLOPERS WEG:            TERMINATE(0);

752 MIABE 368,41;
758 EROGEND
```

DOKTERSPRAKTIJK

NUMBER OF BLOCKS	+15
NUMBER OF FACILITIES	+1
NUMBER OF STORAGES	+0
NUMBER OF QUEUES	+2
NUMBER OF TABLES	+1
NUMBER OF FUNCTIONS	+1
NUMBER OF PARAMETERS	+0
NUMBER OF TRANSACTIONS	+1000

FUNCTION	TYPE	X	Y
1	CONTINU	+.00000	+2.00000
		+.10000	+3.00000
		+.40000	+4.00000
		+.70000	+5.00000
		+.90000	+6.00000
		+.95000	+7.00000
		+.97000	+8.00000
		+.99000	+9.00000
		+1.00000	+10.00000

CLOCK +.3127117622431E+ 4

BLOCK NO	ENTRIES
1	696
2	646
3	646
4	501
5	501
6	500
7	500
8	534
9	534
10	404
11	404
12	404
13	534
14	145
15	145

FACILITY	AVERAGE UTILIZATION	ENTRIES
1	.962	1050

QUEUE	AVERAGE CONTENTS	MAXIMUM CONTENTS	CURRENT CONTENTS	TOTAL ENTRIES	ZERO ENTRIES
1	4.036	15	10	634	22
2	.000	0	0	0	404

TABLE 1

MEAN	+17.1
VARIANCE	+194.2
MINIMUM	+0
MAXIMUM	+79
ENTRIES	+501

CELL	ENTRIES	PERCENTAGE	
< +0	0	0	|
-0 - +1	41	8	| ********
+2 - +3	30	6	| ******
+4 - +5	29	6	| ******
+6 - +7	38	8	| ********
+8 - +9	59	12	| ************
+10 - +11	33	7	| *******
+12 - +13	34	7	| *******
+14 - +15	30	6	| ******
+16 - +17	26	5	| *****
+18 - +19	13	3	| ***
+20 - +21	18	4	| ****
+22 - +23	19	4	| ****
+24 - +25	11	2	| **
+26 - +27	14	3	| ***
+28 - +29	19	4	| ****
+30 - +31	14	3	| ***
+32 - +33	17	3	| ***
+34 - +35	7	1	| *
+36 - +37	10	2	| **
+38 - +39	7	1	| *
+40 - +41	7	1	| *
+42 - +43	3	1	| *
+44 - +45	3	1	| *
+46 - +47	2	0	|
+48 - +49	2	0	|
+50 - +51	2	0	|
+52 - +53	1	0	|
+54 - +55	0	0	|
+56 - +57	1	0	|
+58 - +59	3	1	| *
> +59	8	2	| **

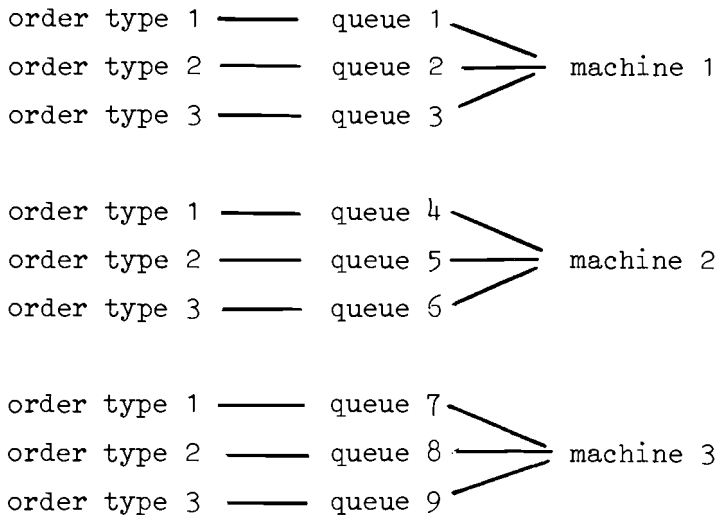
11.2. Job-shop.

Het in de inleiding geformuleerde job-shop probleem specificeren we als volgt nader:

1. De tussenaankomsttijd is voor elk type order negatief exponentiëel verdeeld met gemiddelde 1.
2. De bewerkingstijden zijn eveneens negatief exponentiëel verdeeld en wel voor machine 1 met gemiddelde 1 en voor machine 2 en 3 met gemiddelde 2.
3. De transactions (= orders) zijn uitgebreid met elf parameters, die de volgende betekenis hebben:

par. nr.	inhoud
1	type (1, 2 of 3)
2	aantal af te werken machines
3	aantal afgewerkte machines
4	het nummer van de eerste machine
5	het nummer van de tweede machine
6	het nummer van de derde machine
7	bewerkingstijd op de eerste machine
8	bewerkingstijd op de tweede machine
9	bewerkingstijd op de derde machine
10	het nummer van de eerstvolgende machine
11	de bewerkingstijd op de eerstvolgende machine

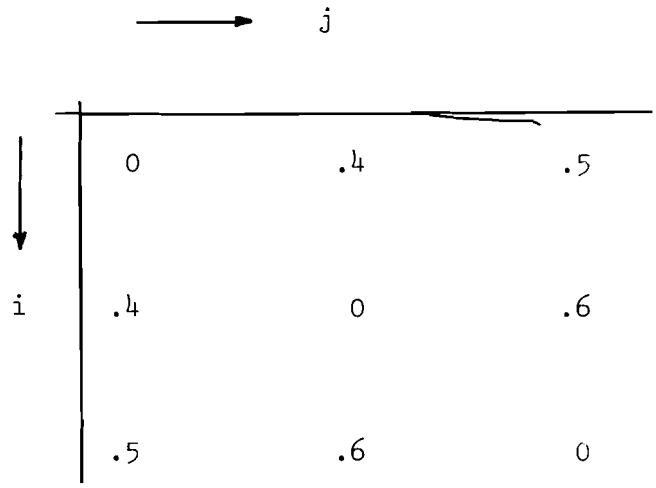
4. Vóórdat een order een machine verlaat, wordt de volgende order aangewezen volgens de in de inleiding gesuggereerde discipline. Voor de uitvoering hiervan wordt gebruik gemaakt van:
   a. drie queues voor iedere machine:



b. de procedure nexttype.

Deze wordt aangeroepen als de queue waaruit de order, die een machine verlaat, komt, leeg is. De procedure maakt een van de twee andere queues tot preferential seize queue, óf laat de preferential seize queue onbepaald (namelijk als de andere twee queues ook leeg zijn).

5. Als een order die op een machine komt, niet van hetzelfde type is als de vorige order op de machine, wordt de bewerkingstijd verlengd met de omschakeltijd. Het bepalen van de omschakeltijd gebeurt met behulp van:
   a. het integer array type [1: 3, 1: 2].
      De waarde van type [i, 1] is: het type nummer van de vorige order op machine i en de waarde van type [i, 2] is: het typenummer van de huidige order op machine i. Als type [i, 1] ≠ type [i, 2] moet bij de bewerkingstijd de omschakeltijd worden opgeteld.
   b. het real array oms [1: 3, 1: 3] met de volgende waarden:



De totale bewerkingstijd van een order op machine i is dan steeds:
par (11) + oms [type [i, 1] , type [i, 2]] .

6. Nadat een order op een machine bewerkt is, wordt de parameter 3 met 1 opgehoogd en worden parameter 10 en 11 bijgewerkt.
Als par (3) = par (2) wordt de order vernietigd.

```
  1 MIABE 354; 0-9;
 84 UBCC:=50000;
 85 JOB(‡JOB SHOP PROBLEEM UIT SIMULATIESYLLABUS‡);
 86 SETALGPBS([4,3,0,9,3,0,11,1000,500]);

 87 MIABE 354,10-20;

266 INITIALIZE(0,1); INITIALIZE(0,3); INITIALIZE(0,5);
267 TABLE(1,30,0,.5); TABLE(2,30,0,.5); TABLE(3,30,0,.5);
268 QUEUEDATA;

269 MIABE 354,21;

300 SWITCH S:=L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12,L13,L14;
301 INTEGER I1,I2,Q1,Q2,Q3,MACH;
302 INTEGER ARRAY TYPE[1:3,1:2], SAVEQ[1:2]; REAL ARRAY OMS[1:3,1:3];
303 PROCEDURE NEXTTYPE;
304 BEGIN   MACH:=PAR(I0); I2:=0; Q1:=3*MACH-2; Q2:=3*MACH-1; Q3:=3*MACH;
305       FOR I1:=Q1,Q2,Q3 DO
306       BEGIN IF I1≠FPSQ[MACH] THEN
307             BEGIN   I2:=I2+1; SAVEQ[I2]:=I1;
308             END
309       END;
310       IF  LENGTH(SAVEQ[1])>0 THEN
311       BEGIN   IF  LENGTH(SAVEQ[2])>0 THEN
312             BEGIN   IF QTP(SAVEQ[1],1,0)> QTP(SAVEQ[2],1,0) THEN
313                                 FPSQ[MACH]:=SAVEQ[1] ELSE FPSQ[MACH]:=SAVEQ[2]
314             END
315             ELSE    FPSQ[MACH]:=SAVEQ[1]
316       END
317       ELSE    IF LENGTH(SAVEQ[2])>0 THEN FPSQ[MACH]:=SAVEQ[2]
318       ELSE    FPSQ[MACH]:=NILL
319 END NEXTTYPE;

320 MIABE 354,22-39;

745 FOR I1:=1,2,3 DO TYPE[I1,1]:=TYPE[I1,2]:=NILL;
746 PRINTTEXT(‡MATRIX VAN OMSCHAKELTYDEN‡); CARRIAGE(2);
747 FOR I:=1,2,3 DO BEGIN   FOR J:=1,2,3 DO ABSFIXT(2,2,(OMS[I,J]:=READ)); CARRIAGE(4)
748             END;

749 MIABE 354,40;


759 L1:     GENERATE(NEGEXP(1),1);
760       ASSIGN(1,1); ASSIGN(2,2); ASSIGN(3,0); ASSIGN(4,1); ASSIGN(7,NEGEXP(.1));
761       IF RANDOM1<.5   THEN BEGIN ASSIGN(5,2); ASSIGN(8,NEGEXP(.2)) END
762                       ELSE BEGIN ASSIGN(5,3); ASSIGN(8,NEGEXP(.2)) END;
763 L2:     TRANSFER(6);
764 L3:     GENERATE(NEGEXP(1),3);
765       ASSIGN(1,2); ASSIGN(2,3); ASSIGN(3,0);
766       IF RANDOM1<.5   THEN BEGIN ASSIGN(4,2); ASSIGN(5,1); ASSIGN(6,3); ASSIGN(7,NEGEXP(.2)); ASSIGN(8,NEGEXP(.1)); ASSIGN(9,NEGEXP(.2))
767                       END
768                 ELSE BEGIN ASSIGN(4,3); ASSIGN(5,1); ASSIGN(6,2); ASSIGN(7,NEGEXP(.2)); ASSIGN(8,NEGEXP(.1)); ASSIGN(9,NEGEXP(.2))
769                       END;
770 L4:     TRANSFER(6);
771 L5:     GENERATE(NEGEXP(1),5);
772       ASSIGN(1,3); ASSIGN(2,3); ASSIGN(3,0); ASSIGN(4,3); ASSIGN(5,2); ASSIGN(6,1);
773       ASSIGN(7,NEGEXP(.2)); ASSIGN(8,NEGEXP(.2)); ASSIGN(9,NEGEXP(.1));
774 L6:     IF PAR(2)-PAR(3)<0 THEN         BEGIN ASSIGN(10,PAR(4+PAR(3))); ASSIGN(11,PAR(7+PAR(3))); TRANSFER(7)
```

```
775                                   END ELSE TRANSFER(13);
776 L7:  TRANSFER(8);
777 L8:     SEIZE(PAR(10),3*PAR(10)+PAR(1)-3,NOW);
778 L9:     TYPE(PAR(10),2)=PAR(1); IF TYPE(PAR(10),1)=NILL THEN BEGIN TYPE(PAR(10),1)=PAR(1); TRANSFER(10); END ELSE TRANSFER(15);
779 L10:    ADVANCE(PAR(11)*OMS(TYPE(PAR(10),1),TYPE(PAR(10),2)));
780 L11:    TYPE(PAR(10),1)=PAR(1); IF LENGTH(FPSO(PAR(10)))=0 THEN NEXTTYPE;
781         RELEASE(PAR(10));
782 L12:    ASSIGN(3,PAR(3)+1);
783         TRANSFER(6);
784 L13:    TABULATE(PAR(1),TT,1);
785 L14:    TERMINATE(1);
786 MIARE 354;41;
792 EPROGEND
```

JOB SHOP PROBLEEM UIT SIMULATIESYLLABUS

NUMBER OF BLOCKS	+14
NUMBER OF FACILITIES	+3
NUMBER OF STORAGES	+0
NUMBER OF QUEUES	+9
NUMBER OF TABLES	+3
NUMBER OF FUNCTIONS	+0
NUMBER OF PARAMETERS	+11
NUMBER OF TRANSACTIONS	+1000

MATRIX VAN OMSCHAKELTYDEN

.00	.40	.50
.40	.00	.60
.50	.60	.00

CLOCK +.146196612117 6+ 3

BLOCK NO	ENTRIES
1	162
2	162
3	179
4	179
5	167
6	1856
7	1356
8	1351
9	1351
10	1351
11	1348
12	1348
13	500
14	500

FACILITY	AVERAGE UTILIZATION	ENTRIES
1	.796	505
2	.903	429
3	.908	417

QUEUE	AVERAGE CONTENTS	MAXIMUM CONTENTS	CURRENT CONTENTS	TOTAL ENTRIES	ZERO ENTRIES
1	.599	8	0	138	24
2	.620	4	1	148	29
3	.644	5	0	147	20
4	.675	4	1	76	12
5	1.161	8	0	161	14
6	1.304	10	0	154	13
7	.900	7	0	69	5
8	1.114	8	3	166	13
9	1.385	10	0	157	10

TABLE 1

MEAN	+2.86
VARIANCE	+5.73
MINIMUM	+.2
MAXIMUM	+15.5
ENTRIES	+160

CELL		ENTRIES	PERCENTAGE	
<	+.0	0	.0	|
-.0 -	+.4	1	.6	| *
+.5 -	+.9	12	7.5	| ********
+1.0 -	+1.4	30	18.7	| **********
+1.5 -	+1.9	33	20.6	| **********
+2.0 -	+2.4	26	16.2	| **********
+2.5 -	+2.9	13	8.1	| ********
+3.0 -	+3.4	11	6.9	| *******
+3.5 -	+3.9	4	2.5	| ***
+4.0 -	+4.4	3	1.9	| **
+4.5 -	+4.9	4	2.5	| ***
+5.0 -	+5.4	5	3.1	| ***
+5.5 -	+5.9	5	3.1	| ***
+6.0 -	+6.4	3	1.9	| **
+6.5 -	+6.9	0	.0	|
+7.0 -	+7.4	2	1.2	| *
+7.5 -	+7.9	1	.6	| *
+8.0 -	+8.4	1	.6	| *
+8.5 -	+8.9	0	.0	|
+9.0 -	+9.4	2	1.2	| *
+9.5 -	+9.9	0	.0	|
+10.0 -	+10.4	0	.0	|
+10.5 -	+10.9	0	.0	|
+11.0 -	+11.4	0	.0	|
+11.5 -	+11.9	0	.0	|
+12.0 -	+12.4	3	1.9	| **
+12.5 -	+12.9	0	.0	|
+13.0 -	+13.4	0	.0	|
+13.5 -	+13.9	0	.0	|
+14.0 -	+14.4	0	.0	|
+14.5 -	+14.9	0	.0	|
>	+14.9	1	.6	| *

TABLE 2

MEAN	+3.64
VARIANCE	+2.59
MINIMUM	+.5
MAXIMUM	+9.0
ENTRIES	+173

CELL	ENTRIES	PERCENTAGE	
< +.0	0	.0	
-.0 - +.4	0	.0	
+.5 - +.9	4	2.3	**
+1.0 - +1.4	4	2.3	**
+1.5 - +1.9	14	8.1	********
+2.0 - +2.4	26	15.0	***************
+2.5 - +2.9	18	10.4	**********
+3.0 - +3.4	31	17.9	******************
+3.5 - +3.9	18	10.4	**********
+4.0 - +4.4	12	6.9	*******
+4.5 - +4.9	13	7.5	********
+5.0 - +5.4	10	5.8	******
+5.5 - +5.9	8	4.6	*****
+6.0 - +6.4	6	3.5	***
+6.5 - +6.9	3	1.7	**
+7.0 - +7.4	2	1.2	*
+7.5 - +7.9	1	.6	*
+8.0 - +8.4	1	.6	*
+8.5 - +8.9	1	.6	*
+9.0 - +9.4	1	.6	*
+9.5 - +9.9	0	.0	
+10.0 - +10.4	0	.0	
+10.5 - +10.9	0	.0	
+11.0 - +11.4	0	.0	
+11.5 - +11.9	0	.0	
+12.0 - +12.4	0	.0	
+12.5 - +12.9	0	.0	
+13.0 - +13.4	0	.0	
+13.5 - +13.9	0	.0	
+14.0 - +14.4	0	.0	
+14.5 - +14.9	0	.0	
> +14.9	0	.0	

TABLE 3

MEAN	+4.26
VARIANCE	+4.34
MINIMUM	+.3
MAXIMUM	+10.6
ENTRIES	+167

CELL			ENTRIES	PERCENTAGE	
	<	+.0	0	.0	
-.0	-	+.4	1	.6	*
+.5	-	+.9	2	1.2	*
+1.0	-	+1.4	6	3.6	****
+1.5	-	+1.9	5	3.0	***
+2.0	-	+2.4	17	10.2	**********
+2.5	-	+2.9	18	10.8	**********
+3.0	-	+3.4	27	16.2	**********
+3.5	-	+3.9	15	9.0	*********
+4.0	-	+4.4	13	7.8	********
+4.5	-	+4.9	16	9.6	**********
+5.0	-	+5.4	12	7.2	*******
+5.5	-	+5.9	7	4.2	****
+6.0	-	+6.4	2	1.2	*
+6.5	-	+6.9	6	3.6	****
+7.0	-	+7.4	6	3.6	****
+7.5	-	+7.9	1	.6	*
+8.0	-	+8.4	5	3.0	***
+8.5	-	+8.9	3	1.8	**
+9.0	-	+9.4	0	.0	
+9.5	-	+9.9	1	.6	*
+10.0	-	+10.4	3	1.8	**
+10.5	-	+10.9	1	.6	*
+11.0	-	+11.4	0	.0	
+11.5	-	+11.9	0	.0	
+12.0	-	+12.4	0	.0	
+12.5	-	+12.9	0	.0	
+13.0	-	+13.4	0	.0	
+13.5	-	+13.9	0	.0	
+14.0	-	+14.4	0	.0	
+14.5	-	+14.9	0	.0	
	>	+14.9	0	.0	

CLOCK +.3331997453193E+ 3

BLOCK NO	ENTRIES
1	334
2	334
3	346
4	346
5	333
6	3692
7	2692
8	2682
9	2682
10	2682
11	2679
12	2679
13	1000
14	1000

FACILITY	AVERAGE UTILIZATION	ENTRIES
1	.833	1004
2	.902	860
3	.929	818

QUEUE	AVERAGE CONTENTS	MAXIMUM CONTENTS	CURRENT CONTENTS	TOTAL ENTRIES	ZERO ENTRIES
1	.681	8	0	285	49
2	.667	7	0	296	46
3	.649	5	2	297	33
4	.628	4	2	164	25
5	1.089	8	1	313	30
6	1.051	10	1	307	25
7	.720	7	0	133	11
8	1.160	8	3	323	22
9	1.102	10	1	309	24

TABLE 1

MEAN	+2.70
VARIANCE	+4.45
MINIMUM	+.2
MAXIMUM	+15.5
ENTRIES	+331

CELL		ENTRIES	PERCENTAGE	
<	+.0	0	.0	|
-.0 -	+.4	2	.6	| *
+.5 -	+.9	17	5.1	| *****
+1.0 -	+1.4	74	22.4	| **********
+1.5 -	+1.9	65	19.6	| **********
+2.0 -	+2.4	55	16.6	| **********
+2.5 -	+2.9	31	9.4	| *********
+3.0 -	+3.4	20	6.0	| ******
+3.5 -	+3.9	11	3.3	| ***
+4.0 -	+4.4	10	3.0	| ***
+4.5 -	+4.9	10	3.0	| ***
+5.0 -	+5.4	10	3.0	| ***
+5.5 -	+5.9	7	2.1	| **
+6.0 -	+6.4	4	1.2	| *
+6.5 -	+6.9	0	.0	|
+7.0 -	+7.4	2	.6	| *
+7.5 -	+7.9	1	.3	|
+8.0 -	+8.4	1	.3	|
+8.5 -	+8.9	0	.0	|
+9.0 -	+9.4	3	.9	| *
+9.5 -	+9.9	1	.3	|
+10.0 -	+10.4	1	.3	|
+10.5 -	+10.9	1	.3	|
+11.0 -	+11.4	1	.3	|
+11.5 -	+11.9	0	.0	|
+12.0 -	+12.4	3	.9	| *
+12.5 -	+12.9	0	.0	|
+13.0 -	+13.4	0	.0	|
+13.5 -	+13.9	0	.0	|
+14.0 -	+14.4	0	.0	|
+14.5 -	+14.9	0	.0	|
>	+14.9	1	.3	|

TABLE 2

MEAN	+3.80
VARIANCE	+2.84
MINIMUM	+.5
MAXIMUM	+9.8
ENTRIES	+341

CELL		ENTRIES	PERCENTAGE	
<	+.0	0	.0	|
-.0 -	+.4	0	.0	|
+.5 -	+.9	6	1.8	| **
+1.0 -	+1.4	10	2.9	| ***
+1.5 -	+1.9	24	7.0	| *******
+2.0 -	+2.4	40	11.7	| **********
+2.5 -	+2.9	42	12.3	| **********
+3.0 -	+3.4	54	15.8	| **********
+3.5 -	+3.9	32	9.4	| *********
+4.0 -	+4.4	30	8.8	| *********
+4.5 -	+4.9	27	7.9	| ********
+5.0 -	+5.4	18	5.3	| *****
+5.5 -	+5.9	23	6.7	| *******
+6.0 -	+6.4	12	3.5	| ****
+6.5 -	+6.9	6	1.8	| **
+7.0 -	+7.4	6	1.8	| **
+7.5 -	+7.9	3	.9	| *
+8.0 -	+8.4	4	1.2	| *
+8.5 -	+8.9	1	.3	|
+9.0 -	+9.4	2	.6	| *
+9.5 -	+9.9	1	.3	|
+10.0 -	+10.4	0	.0	|
+10.5 -	+10.9	0	.0	|
+11.0 -	+11.4	0	.0	|
+11.5 -	+11.9	0	.0	|
+12.0 -	+12.4	0	.0	|
+12.5 -	+12.9	0	.0	|
+13.0 -	+13.4	0	.0	|
+13.5 -	+13.9	0	.0	|
+14.0 -	+14.4	0	.0	|
+14.5 -	+14.9	0	.0	|
>	+14.9	0	.0	|

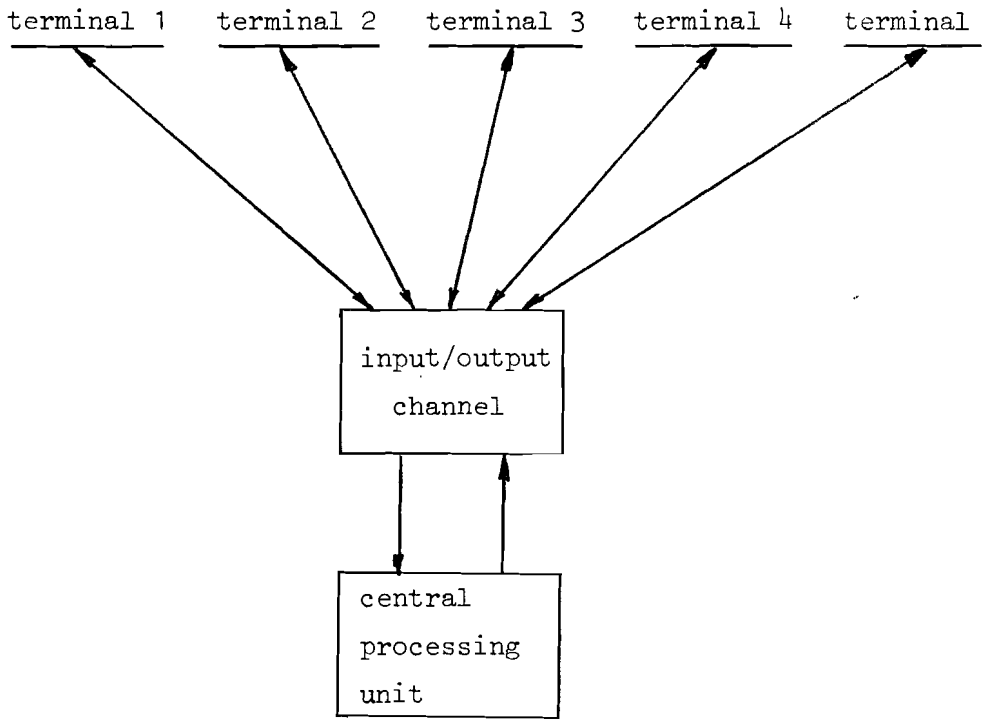
TABLE 3

MEAN	+5.85
VARIANCE	+3.16
MINIMUM	+.3
MAXIMUM	+10.6
ENTRIES	+328

CELL		ENTRIES	PERCENTAGE	
<	+.0	0	.0	|
-.0 -	+.4	1	.3	|
+.5 -	+.9	2	.6	| *
+1.0 -	+1.4	11	3.4	| ***
+1.5 -	+1.9	14	4.3	| ****
+2.0 -	+2.4	40	12.2	| **********
+2.5 -	+2.9	50	15.2	| **********
+3.0 -	+3.4	55	16.8	| **********
+3.5 -	+3.9	36	11.0	| **********
+4.0 -	+4.4	28	8.5	| *********
+4.5 -	+4.9	24	7.3	| *******
+5.0 -	+5.4	19	5.8	| ******
+5.5 -	+5.9	11	3.4	| ***
+6.0 -	+6.4	8	2.4	| **
+6.5 -	+6.9	7	2.1	| **
+7.0 -	+7.4	8	2.4	| **
+7.5 -	+7.9	1	.3	|
+8.0 -	+8.4	5	1.5	| **
+8.5 -	+8.9	3	.9	| *
+9.0 -	+9.4	0	.0	|
+9.5 -	+9.9	1	.3	|
+10.0 -	+10.4	3	.9	| *
+10.5 -	+10.9	1	.3	|
+11.0 -	+11.4	0	.0	|
+11.5 -	+11.9	0	.0	|
+12.0 -	+12.4	0	.0	|
+12.5 -	+12.9	0	.0	|
+13.0 -	+13.4	0	.0	|
+13.5 -	+13.9	0	.0	|
+14.0 -	+14.4	0	.0	|
+14.5 -	+14.9	0	.0	|
>	+14.9	0	.0	|

11.3. Online informatie-systeem.

Beschouw onderstaand schema.



Bij de terminals komen vraagmessages aan van constante lengte. Ze worden gelijkerlijk over de vijf terminals verdeeld. Als de terminal vrij is, wordt de message via de communicatielijn tussen terminal en de I/O channel overgebracht naar de central processing unit. Deze kan tien messages tegelijk bewaren, maar behandelt er steeds maar één. Nadat een vraagmessage behandeld is, wordt de antwoord-message teruggestuurd via de I/O channel en de communicatielijn naar de terminal, waarvandaan de vraagmessage is gekomen. Interessant voor dit systeem is, wat de totale doorlooptijd van de message is, d.w.z. de tijd tussen de aankomst van de vraag bij de terminal en het moment waarop het antwoord terugontvangen is.

De tussenaankomsttijd van de messages is negatief exponentiëel verdeeld met gemiddelde 500. Het toewijzen van de terminal geschiedt met behulp van de discrete function 1. De transmissietijden zijn constant (n.l. 50 en 75) en de procesduur in de cpu is uniform verdeeld tussen 150 en 450 tijdseenheden.

```
297 --  0 =   280471 - 08164367 ALGPSS

  1 MIABE 368, 0-9;
 84 JOB(¢VOORBEELD GPSS (BM MANUAL$);
 85 SETALGPSS(22,8,1,10,1,1,1,1000,500);
 86 MIABE 368,10-20;
265 INITIALIZE(0,1);
266 STORAGE(1,10);
267 TABLE(1,100,0,100);
268 FUNCTION(1,10,1);
269 QUEUEDATA;
270 MIABE 368,21;
301 SWITCH S:= L1,L3,L5,L6,L7,L8,L9,L10,L11,L12,L13,L14,L15,L16,L17,L18,L19,L20,L21,L22,L23,L24;
302 MIABE 368,22-40;
736 L1:     GENERATE(NEGEXP(500),1);
737         ASSIGN(1,FUNC(1,RANDOM));
738 L3:     SEIZE(PAR(1),PAR(1),NOW);
739 L5:     SEIZE(6,6,NOW);
740 L6:     ADVANCE(50);
741 L7:     ENTER(1,1,7,NOW);
742 L8:     PREEMPT(7,8,NOW);
743 L9:     ADVANCE(10);
744 L10:    RETURN(7);
745 L11:    RELEASE(6);
746 L12:    SEIZE(7,9,NOW);
747 L13:    ADVANCE(150+300*RANDOM);
748 L14:    RELEASE(7);
749 L15:    SEIZE(8,10,NOW);
750 L16:    PREEMPT(7,8,NOW);
751 L17:    ADVANCE(15);
752 L18:    RETURN(7);
753 L19:    LEAVE(1,1);
754 L20:    ADVANCE(75);
755 L21:    RELEASE(8);
756 L22:    RELEASE(PAR(1));
757 L23:    TABULATE(1,TT,1);
758 L24:    TERMINATE(1);

759 MIABE 368,41;
765 PROGEND
```

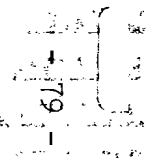
VOORBEELD GPSS IBM MANUAL

NUMBER OF BLOCKS	+22
NUMBER OF FACILITIES	+8
NUMBER OF STORAGES	+1
NUMBER OF QUEUES	+10
NUMBER OF TABLES	+1
NUMBER OF FUNCTIONS	+1
NUMBER OF PARAMETERS	+1
NUMBER OF TRANSACTIONS	+1000

FUNCTION	TYPE	X	Y
1	DISCREET		
		+.20000	+1.00000
		+.40000	+2.00000
		+.60000	+3.00000
		+.80000	+4.00000
		+1.00000	+5.00000

CLOCK +.2691910450401E+ 6

BLOCK NO	ENTRIES
1	500
2	500
3	500
4	500
5	500
6	500
7	500
8	500
9	500
10	500
11	500
12	500
13	500
14	500
15	500
16	500
17	500
18	500
19	500
20	500
21	500
22	500



FACILITY	AVERAGE UTILIZATION	ENTRIES
1	.212	92
2	.206	94
3	.243	105
4	.258	107
5	.253	102
6	.112	500
7	.616	1500
8	.167	500

STORAGE	CAPACITY	AVERAGE CONTENTS	MAXIMUM CONTENTS	CURRENT CONTENTS	ENTRIES
1	10	.934	5	0	500

QUEUE	AVERAGE CONTENTS	MAXIMUM CONTENTS	CURRENT CONTENTS	TOTAL ENTRIES	ZERO ENTRIES
1	.040	1	0	23	69
2	.021	0	0	17	77
3	.057	3	0	22	83
4	.064	3	0	31	76
5	.032	1	0	23	79
6	.005	1	0	43	457
7	.000	0	0	0	0
8	.000	0	0	6	994
9	.294	3	0	257	243
10	.000	0	0	0	500

TABLE 1

MEAN	+745.9
VARIANCE	+180319.3
MINIMUM	+301
MAXIMUM	+3194
ENTRIES	+500

CELL	ENTRIES	PERCENTAGE	
< +0	0	0	|
-0 - +99	0	0	|
+100 - +199	0	0	|
+200 - +299	0	0	|
+300 - +399	59	12	| **********
+400 - +499	77	15	| ***************
+500 - +599	117	23	| **********************
+600 - +699	47	9	| *********
+700 - +799	47	9	| *********
+800 - +899	38	8	| ********
+900 - +999	28	6	| ******
+1000 - +1099	18	4	| ****
+1100 - +1199	10	2	| **
+1200 - +1299	16	3	| ***
+1300 - +1399	8	2	| **
+1400 - +1499	10	2	| **
+1500 - +1599	2	0	|
+1600 - +1699	2	0	|
+1700 - +1799	4	1	| *
+1800 - +1899	3	1	| *
+1900 - +1999	0	0	|
+2000 - +2099	2	0	|
+2100 - +2199	4	1	| *
+2200 - +2299	1	0	|
+2300 - +2399	1	0	|
+2400 - +2499	2	0	|
+2500 - +2599	0	0	|
+2600 - +2699	1	0	|
+2700 - +2799	0	0	|
+2800 - +2899	0	0	|
+2900 - +2999	1	0	|
+3000 - +3099	1	0	|
+3100 - +3199	1	0	|
+3200 - +3299	0	0	|
+3300 - +3399	0	0	|
+3400 - +3499	0	0	|
+3500 - +3599	0	0	|
+3600 - +3699	0	0	|
+3700 - +3799	0	0	|
+3800 - +3899	0	0	|
+3900 - +3999	0	0	|
+4000 - +4099	0	0	|
+4100 - +4199	0	0	|
+4200 - +4299	0	0	|
+4300 - +4399	0	0	|
+4400 - +4499	0	0	|
+4500 - +4599	0	0	|
+4600 - +4699	0	0	|
+4700 - +4799	0	0	|
+4800 - +4899	0	0	|
+4900 - +4999	0	0	|

```
*** TAPE   368
*** GROEP    0
TOTAALNUMMER VAN DE EERSTE REGEL          +0

    0   BEGIN
    1           INTEGER NUM,READY,NREP,NTRANS,NPRINT,
    2                   UBCC,NILL,LA,NEXT,PRED,SUCC,LALOW,LAHIGH,
    3                   NBLOCK,NFAC,NSTOR,NQUEUE,NTABLE,UBTABLE,CELL,NC,NFUNC,UBF,NPAR,
    4                   I,J,L,N,P,DUMMY,NEXTBLOCK;
    5           REAL    NOW,
    6                   SP,LOW,HIGH,
    7                   CW,SUMX,XL,XH,YL,YH,
    8                   FRAND,FRAND1,FRAND2;
    9           BOOLEAN ARRAY   OUTPUT[1:6];



*** TAPE   368
*** GROEP    1
TOTAALNUMMER VAN DE EERSTE REGEL          +10

    0   PROCEDURE SETRANDOM(X); VALUE X; INTEGER X;
    1   BEGIN
    2         X:= ABS(X); IF X12*2=X THEN X:= X+1;
    3         FRAND:= X/67108864
    4   END SETRANDOM;
    5   REAL PROCEDURE RANDOM;
    6   BEGIN
    7         FRAND:= FRAND*8189;
    8         RANDOM:= FRAND:= FRAND-ENTIER(FRAND)
    9   END RANDOM;



*** TAPE   368
*** GROEP    2
TOTAALNUMMER VAN DE EERSTE REGEL          +20

    0   PROCEDURE SETRANDOM1(X); VALUE X; INTEGER X;
    1   BEGIN
    2         X:= ABS(X); IF X12*2=X THEN X:= X+1;
    3         FRAND1:= X/67108864
    4   END SETRANDOM1;
    5   REAL PROCEDURE RANDOM1;
    6   BEGIN
    7         FRAND1:= FRAND1*8181;
    8         RANDOM1:= FRAND1:= FRAND1-ENTIER(FRAND1)
    9   END RANDOM1;
```

```
*** TAPE   368
*** GROEP    3
TOTAALNUMMER VAN DE EERSTE REGEL          +30

   0   PROCEDURE SETRANDOM2(X); VALUE X; INTEGER X;
   1   BEGIN
   2         X:= ABS(X); IF X12*2=X THEN X:= X+1;
   3         FRAND2:= X/67108864
   4   END SETRANDOM2;
   5   REAL PROCEDURE RANDOM2;
   6   BEGIN
   7         FRAND2:= FRAND2*8173;
   8         RANDOM2:= FRAND2:= FRAND2-ENTIER(FRAND2)
   9   END RANDOM2;



*** TAPE   368
*** GROEP    4
TOTAALNUMMER VAN DE EERSTE REGEL          +40

   0   REAL PROCEDURE NEGEXP(MU); VALUE MU; REAL MU;
   1   BEGIN
   2         NEGEXP:= -MU*LN(RANDOM)
   3   END NEGEXP;



*** TAPE   368
*** GROEP    5
TOTAALNUMMER VAN DE EERSTE REGEL          +44

   0   PROCEDURE JOB (A);
   1   STRING A;
   2   BEGIN
   3         PRINTTEXT(A); CARRIAGE(5)
   4   END JOB;



*** TAPE   368
*** GROEP    6
TOTAALNUMMER VAN DE EERSTE REGEL          +49

   0   PROCEDURE SETALGPSS(N1,N2,N3,N4,N5,N6,N7,N8,N9);
   1   VALUE N1,N2,N3,N4,N5,N6,N7,N8,N9; INTEGER N1,N2,N3,N4,N5,N6,N7,N8,N9;
   2   BEGIN
   3         IF N1<0∨N2<0∨N3<0∨N4<0∨N5<0∨N6<0∨N7<0∨N8<0∨N9<0 THEN
   4         BEGIN CARRIAGE(4);PRINTTEXT('( SETALGPSS ERROR$)); JUMPOUT END;
   5         NBLOCK:=N1;  IF N1>0 THEN OUTPUT[1]:=TRUE;
   6         NPAC:=N2;    IF N2>0 THEN OUTPUT[2]:=TRUE;
   7         NSTOR:=N3;   IF N3>0 THEN OUTPUT[3]:=TRUE;
```

```
  8             NQUEUE:=N4;  IF N4>0 THEN OUTPUT[4]:=TRUE;
  9             NTABLE:=N5;  IF N5>0 THEN OUTPUT[5]:=TRUE;  UBTABLE:=0;
 10             NFUNC:=N6;   IF N6>0 THEN OUTPUT[6]:=TRUE; UBF:=1;
 11             NPAR:= N7;
 12             NTRANS:=N8;  READY:=0;
 13             NPRINT:=N9;  NREP:=1;
 14             PB!WTTEXT(<NUMBER OF BLOCKS        $); F!XT(6,0,NBLOCK); NLCR;
 15             PB!WTTEXT(<NUMBER OF FACILITIES    $); F!XT(6,0,NFAC); NLCR;
 16             PB!WTTEXT(<NUMBER OF STORAGES      $); F!XT(6,0,NSTOR); NLCR;
 17             PB!WTTEXT(<NUMBER OF QUEUES        $); F!XT(6,0,NQUEUE); NLCR;
 18             PB!WTTEXT(<NUMBER OF TABLES        $); F!XT(6,0,NTABLE); NLCR;
 19             PB!WTTEXT(<NUMBER OF FUNCTIONS     $); F!XT(6,0,NFUNC); NLCR;
 20             PB!WTTEXT(<NUMBER OF PARAMETERS    $); F!XT(6,0,NPAR); NLCR;
 21             PB!WTTEXT(<NUMBER OF TRANSACTIONS $); F!XT(6,0,NTRANS)); CARRIAGE(5);
 22    END SETALGPSS;



*** TAPE   368
*** GROEP     7
TOTAALNUMMER VAN DE EERSTE REGEL          +72


  0    PROCEDURE JUMPOUT;
  1    BEGIN
  2             BOOLEAN ARRAY ERROR[1:1];
  3             ERROR[2]:=TRUE
  4    END JUMP OUT OF BOUNDS;



*** TAPE   368
*** GROEP     8
TOTAALNUMMER VAN DE EERSTE REGEL          +77



*** TAPE   368
*** GROEP     9
TOTAALNUMMER VAN DE EERSTE REGEL          +77


  0    NEXT:=NILL:=-110244;
  1    UBCC:=10000;
  2    NUM:=0;
  3    FOR I:=1 STEP 1 UNTIL 6 DO OUTPUT[I]:=FALSE;
  4    SETRANDOM(1); SETRANDOM1(1); SETRANDOM2(1);



*** TAPE   368
*** GROEP    10
TOTAALNUMMER VAN DE EERSTE REGEL          +82
```

```
    0   BEGIN
    1           INTEGER ARRAY   ENT[0:NBLOCK],
    2                           SLA,PLA,FPSQ,FPPQ,FS,FP,FE[0:NFAC],
    3                           SC,SM,SE,SCAP,SPQ[0:NSTOR],
    4                           QC,QM,QE,QZ[0:NQUEUE],
    5                           INF[0:NFUNC,1:2];
    6           BOOLEAN ARRAY   QSTAT[0:NQUEUE];
    7           REAL ARRAY      FSC,FCT[[0:NFAC],
    8                           SSC,SCT[[0:NSTOR],
    9                           QSC,QCT[[0:NQUEUE],
   10                           CC[-5:UBCC],
   11                           HISTDATA[0:NTABLE,1:3]);
```

```
*** TAPE   368
*** GROEP    11
TOTAALNUMMER VAN DE EERSTE REGEL          +94
```

```
    0   INTEGER PROCEDURE FROMF;
    1   BEGIN
    2           IF CC[-2]#CC[-1] THEN
    3           BEGIN
    4                   P:=CC[-2]; CC[-2]:=CC[P]
    5           END
    6           ELSE
    7           BEGIN
    8                   P:= CC[-1]; CC[-2]:= CC[-1]:=P+CC[-4];
    9                   IF CC[-1]>CC[-5]+1 THEN
   10                   BEGIN
   11                           CARRIAGE(4); PRINTTEXT(‡ARRAY CC TOO SMALL‡);
   12                           ERRORDATA; JUMPOUT
   13                   END
   14           END;
   15                   CC[P+4]:= NILL;
   16                   FROMF:=P+7;
   17   END FROM FREE;
```

```
*** TAPE   368
*** GROEP    12
TOTAALNUMMER VAN DE EERSTE REGEL          +112
```

```
    0   PROCEDURE FROMCH(LA, J);
    1   VALUE LA, J; INTEGER LA, J;
    2   BEGIN
    3         P:=3*J;
    4         PRED:= CC[LA-2]; SUCC:= CC[LA-1];
    5         IF PRED=NILL THEN CC[P+1]:= SUCC ELSE CC[PRED-1]:= SUCC;
    6         IF SUCC=NILL THEN CC[P]:= PRED ELSE CC[SUCC-2]:= PRED;
    7         CC[P+2]:= CC[P+2]-1;
    8         CC[LA-3]:= NILL;
    9   END FROM CHAIN;
```

```
*** TAPE   368
*** GROEP    13
TOTAALNUMMER VAN DE EERSTE REGEL          +122


  0   EROCEDURE TOSORT(LA,J,SP);
  1   VALUE LA,J,SP; INTEGER LA,J; REAL SP;
  2   BEGIN
  3         P:= 3*J+1;  LALOW:= CC[P];  LAHIGH:= CC[P-1];
  4           CC[LA]:=SP;
  5         IF LALOW=NILL
  6         THEN
  7   EMPTY:      BEGIN CC[P]:= CC[P-1]:= LA;
  8                     CC[LA-1]:= CC[LA-2]:= NILL
  9               END
 10         ELSE
 11               BEGIN LOW:= CC[LALOW];  HIGH:= CC[LAHIGH];
 12                     IF SP<LOW
 13                     THEN
 14   HEAD:                   BEGIN CC[LALOW-2]:= CC[P]:= LA;
 15                                 CC[LA-1]:= LALOW;  CC[LA-2]:= NILL
 16                             END
 17                     ELSE
 18                             BEGIN IF HIGH≤SP
 19                                   THEN
 20   TAIL:                                BEGIN CC[LAHIGH-1]:= CC[P-1]:= LA;
 21                                              CC[LA-1]:= NILL;  CC[LA-2]:= LAHIGH
 22                                        END
 23                                   ELSE
 24                                        BEGIN IF (SP-LOW) < (HIGH-SP)
 25                                              THEN
 26   SORT FORWARD:                                   BEGIN SUCC:= LALOW;
 27                                                      FOR DUMMY:=0 WHILE SP≥CC[SUCC] DO SUCC:= CC[SUCC-1];
 28                                                      PRED:= CC[SUCC-2]
 29                                                  END
 30                                              ELSE
 31   SORT BACKWARD:                                  BEGIN PRED:= LAHIGH;
 32                                                      FOR DUMMY:=0 WHILE SP<CC[PRED] DO PRED:= CC[PRED-2];
 33                                                      SUCC:= CC[PRED-1]
 34                                                  END;
 35                                              CC[SUCC-2]:= CC[PRED-1]:= LA;
 36                                              CC[LA-1]:= SUCC;  CC[LA-2]:= PRED
 37                                        END
 38                             END
 39               END;
 40         CC[LA-3]:= J;
 41         CC[P+1]:= CC[P+1]+1
 42   END SORT IN CHAIN;


*** TAPE   368
*** GROEP    14
TOTAALNUMMER VAN DE EERSTE REGEL          +165
```

```
  0   PROCEDURE ERRORDATA;
  1   BEGIN
  2         CARRIAGE(2);  PRINTTEXT(£DATA OF ARRAY CC£);
  3         NLCR; PRINTTEXT(£   I    ARRAY[I]£);
  4         FOR J:=-5 STEP 1 UNTIL -1 DO BEGIN NLCR; FIXT(2,0, J); PRINT(CC[J]) END;
  5         P:= CC[-3]; NLCR;
  6         NLCR; PRINTTEXT(£LENGTHS OF CHAINS£);
  7         NLCR; PRINTTEXT(£   I    CHAIN[I]£);
  8         FOR J:=0 STEP 1 UNTIL P-1 DO
  9         BEGIN
 10               NLCR; ABSFIXT(2,0,J); N:=CC[3*J+2]; ABSFIXT(5,0,N);
 11               L:=0; LA:=CC[3*J+1];
 12               IF LA≠NILL THEN
 13               BEGIN
 14                                 L:=1;
 15                         FOR LA:=CC[LA-1] WHILE LA≠NILL DO L:=L+1;
 16               END;
 17               IF N≠L THEN BEGIN PRINTTEXT(£ERRONEOUS      LENGTH =£); FIXT(5,0,L) END;
 18         END;
 19         CARRIAGE(4);
 20   END ERRORDATA;
```

*** TAPE 368
*** GROEP 15
TOTAALNUMMER VAN DE EERSTE REGEL +186

```
  0   PROCEDURE INITIALIZE(TE,LABEL);
  1   VALUE TE,LABEL; INTEGER LABEL; REAL TE;
  2   BEGIN
  3         IF TE<0 THEN BEGIN CARRIAGE(4);PRINTTEXT(£INITIAL ERROR 2£);JUMPOUT END;
  4         IF LABEL<1∨LABEL>NBLOCK THEN BEGIN CARRIAGE(4); PRINTTEXT(£INITIALIZE ERROR 1£); JUMPOUT END;
  5         NUM:=NUM+1;
  6         LA:=FROMF; CC[LA-5]:=TE; CC[LA-4]:=LABEL;
  7         CC[LA-6]:= NUM; TOSORT(LA,0,TE);
  8   END INITIALIZE;
```

*** TAPE 368
*** GROEP 16
TOTAALNUMMER VAN DE EERSTE REGEL +195

```
  0   PROCEDURE TABLE (NO,AKL,LOW,KLBR);
  1   VALUE NO,AKL,LOW,KLBR; INTEGER NO; REAL AKL,LOW,KLBR;
  2   BEGIN
  3         IF NO<1∨NO>NTABLE THEN BEGIN CARRIAGE(4); PRINTTEXT(£TABLE ERROR 1£); JUMPOUT END;
  4         IF AKL<0 ∨ KLBR<0 THEN BEGIN CARRIAGE(4);PRINTTEXT(£ TABLE ERROR 2£); JUMPOUT END;
  5         HISTDATA[NO,1]:=AKL; HISTDATA[NO,2]:=LOW; HISTDATA[NO,3]:=KLBR;
  6         UBTABLE:=UBTABLE+HISTDATA[NO,1]+7;
  7   END TABLE;
```

```
*** TAPE   368
*** GROEP    24
TOTAALNUMMER VAN DE EERSTE REGEL          +324


   0   PROCEDURE ADVANCE (NAT);
   1   VALUE NAT; REAL NAT;
   2   BEGIN
   3           IF NAT<0 THEN BEGIN CARRIAGE(4);PRINTTEXT(*ADVANCE ERROR*);JUMPOUT END;
   4           ENT[CC[LA-4]]:=ENT[CC[LA-4]]+1;
   5           CC[LA-4]:=CC[LA-4]+1;
   6           TOSORT(LA,0,NOW+NAT);
   7           GOTO CLOCK;
   8   END ADVANCE;



*** TAPE   368
*** GROEP    25
TOTAALNUMMER VAN DE EERSTE REGEL          +333


   0   PROCEDURE ASSIGN(I,T);
   1   VALUE I,T; INTEGER I; REAL T;
   2   BEGIN
   3           IF I<1 V I>NPAR THEN
   4           BEGIN
   5                   CARRIAGE(4); PRINTTEXT(*ASSIGN ERROR 1*);
   6                   ERRORDATA; PRINTTRANSACTION; PRINTOUT; JUMPOUT
   7           END;
   8           CC[LA+I]:=T;
   9   END ASSIGN;
  10   PROCEDURE TRANSFER(R);
  11   VALUE R; INTEGER R;
  12   BEGIN
  13           IF R<1 V R>NBLOCK THEN BEGIN CARRIAGE(4);PRINTTEXT(*TRANSFER ERROR *);JUMPOUT END;
  14           ENT[CC[LA-4]]:=ENT[CC[LA-4]] +1;
  15           CC[LA-4]:= R; GOTO S[R];
  16   END TRANSFER;



*** TAPE   368
*** GROEP    26
TOTAALNUMMER VAN DE EERSTE REGEL          +350


   0   PROCEDURE SEIZE(K,J,SP);
   1   VALUE K,J,SP; INTEGER K,J; REAL SP;
   2   BEGIN
   3           IF K<1VK>NFAC THEN BEGIN CARRIAGE(4); PRINTTEXT(* SEIZE ERROR 1*); JUMPOUT END;
   4           IFJ<1 V J> NQUEUE THEN BEGIN CARRIAGE(4); PRINTTEXT(* SEIZE ERROR 2*); JUMPOUT END;
   5           IF FPSQ[K]=NILL THEN FPSQ[K]:=J;
   6           IF SLA[K]#NILLV PLA[K]#NILLV FPSQ[K]#J THEN
```

```
 7          BEGIN
 8                  IF QSTAT[J] THEN QUEUEIN(J); TOSORT(LA,J,SP);
 9                  GOTO CLOCK;
10          END ELSE
11          BEGIN
12                  IF CC[LA-3]=J THEN
13                          BEGIN
14                          IF QSTAT[J] THEN QUEUEOUT(J); FROMCH(LA,J);
15                  END
16                  ELSE QZ[J]:=QZ[J]+1;
17                  SLA[K]:=LA;
18                          ENT[CC[LA-4]]:=ENT[CC[LA-4]]+1;
19                  FSC[K]:=NOW;FE[K]:=FE[K]+1; FS[K]:=CC[LA-6];
20                  CC[LA-4]:=CC[LA-4]+1;
21          END
22  END SEIZE;
```

*** TAPE 368
*** GROEP 27
TOTAALNUMMER VAN DE EERSTE REGEL +373

```
 0  PROCEDURE RELEASE (K);
 1  VALUE K; INTEGER K;
 2  BEGIN
 3          IF K<1 ∨ K>NFAC THEN BEGIN CARRIAGE(4);PRINTTEXT(£RELEASE ERROR 2£);JUMPOUT END;
 4          IF FS[K]≠CC[LA-6] THEN
 5          BEGIN
 6                  CARRIAGE(4); PRINTTEXT(£RELEASE ERROR 1£);
 7                  ERRORDATA; PRINTTRANSACTION; PRINTOUT; JUMPOUT
 8          END;
 9          SLA[K]:=NILL;
10          ENT[CC[LA-4]]:=ENT[CC[LA-4]] + 1;
11          FS[K]:=0;
12          FCT[K]:=FCT[K]+NOW-FSC[K];
13          CC[LA-4]:=CC[LA-4] +1;
14          IF FPSQ[K]≠NILL THEN NEXT:=CC[3*FPSQ[K]+1];
15          IF NEXT≠NILL THEN
16          BEGIN
17                  TOSORT(LA,0,-1); GOTO CLOCK;
18          END
19  END RELEASE;
```

*** TAPE 368
*** GROEP 28
TOTAALNUMMER VAN DE EERSTE REGEL +393

```
 0  PROCEDURE PREEMPT(K,J,SP);
 1  VALUE K,J,SP; INTEGER K,J; REAL SP;
 2  BEGIN
 3          IF K<1 ∨ K> NFAC THEN BEGIN CARRIAGE(4);PRINTTEXT(£ PREEMPT ERROR 1£); JUMPOUT; END;
 4          IF J< 1 ∨ J> NQUEUE THEN BEGIN CARRIAGE(4);PRINTTEXT(£PREEMPT ERROR 2£);JUMPOUT;END;
 5          IF FPPQ[K]= NILL THEN FPPQ[K]:= J;
```

```
 6          IF PLA[K]# NILL ∨ FPPQ[K]#J THEN
 7            BEGIN
 8                  IF QSTAT[J] THEN QUEUEIN(J); TOSORT( LA,J,SP);
 9                  GOTO CLOCK;
10            END
11                  ELSE
12            BEGIN
13                  IF  CC[LA-3]=J THEN
14                  BEGIN
15                        IF QSTAT[J] THEN QUEUEOUT(J); FROMCH(LA,J)
16                  END
17                  ELSE
18                        QZ[J]:=QZ[J]+1;
19                  PLA[K]:=LA; L:=SLA[K];
20                  IF L=NILL THEN FSC[K]:=NOW
21                  ELSE IF CC[L-2]# 2* NILL THEN
22                  BEGIN
23                        J:=CC[L-3]; FROMCH(L,J);
24                        CC[L-2]:=2*NILL ;CC[L-3]:=J;
25                        IF J=0 THEN  CC[L]:=CC[L]-NOW
26                  END;
27                  FP[K]:= CC[LA-6];
28                  ENT[CC[LA-4]]:= ENT[CC[LA-4]]+1;
29                  FE[K]:= FE[K] +1 ;
30                  CC[LA-4]:=CC[LA-4] +1;
31            END
32    END PREEMPT;
```


*** TAPE   368
*** GROEP   29
TOTAALNUMMER VAN DE EERSTE REGEL   +426

```
 0    PROCEDURE RETURN(K);
 1    VALUE K; INTEGER K;
 2    BEGIN
 3          IF K<1 ∨ K>NFAC THEN BEGIN CARRIAGE(4);PRINTTEXT(£RETURN EROR 2£);JUMPOUT END;
 4          IF FP[K]≠CC[LA-6] THEN
 5          BEGIN
 6                CARRIAGE(4); PRINTTEXT(£RETURN ERROR 1£);
 7                ERRORDATA; PRINTTRANSACTION; PRINTOUT; JUMPOUT
 8          END;
 9          PLA[K]:=NILL;
10          ENT[CC[LA-4]]:=ENT[CC[LA-4]] +1;
11          IF SLA[K]=NILL THEN FCT[K]:=FCT[K]+NOW-FSC[K];
12          FP[K]:=0;
13          CC[LA-4]:=CC[LA-4]+1;
14          IF FPPQ[K]≠NILL THEN NEXT:=CC[3*FPPQ[K]+1];
15          IF NEXT=NILL THEN
16          BEGIN
17                L:=SLA[K]; IF L≠NILL THEN
18                BEGIN
19                      J:=CC[L-3];
20                      IF J=0 THEN CC[L]:=CC[L]+NOW;
21                      TOSORT(L,J,CC[L]);
22                END
23                ELSE
```

```
24                  IF FPSQ[K]#NILL THEN NEXT:=CC[3*FPSQ[K]+1];
25          END;
26          IF NEXT#NILL THEN
27          BEGIN
28                  TOSORT(LA,0,-1); GOTO CLOCK;
29          END
30  END RETURN;
```

*** TAPE 368
*** GROEP 30
TOTAALNUMMER VAN DE EERSTE REGEL +457

```
 0  PROCEDURE ENTER( NO,APL,J,SP);
 1  VALUE NO,APL,J,SP;  INTEGER NO,APL,J; REAL SP;
 2  BEGIN   IF NO<1∨NO>NSTOR THEN BEGIN CARRIAGE(4);PRINTTEXT(‹ ENTER ERROR 1›); JUMPOUT END;
 3          IF APL<0 THEN BEGIN CARRIAGE(4); PRINTTEXT(‹ ENTER ERROR 2›); JUMPOUTEND;
 4          IF J<1∨J>NQUEUE THEN BEGIN CARRIAGE(4);PRINTTEXT(‹ ENTER ERROR 3›); JUMPOUT END;
 5          IF SPQ[NO]=NILL THEN SPQ[NO]:=J; P:=SCAP[NO]-SC[NO];
 6          IF APL>P∨ SPQ[NO]≠J THEN
 7      BEGIN
 8          CC[LA-7]:=APL;
 9          IF QSTAT[J] THEN QUEUEIN(J); TOSORT(LA,J,SP);
10          GOTO CLOCK;
11      END
12          ELSE
13      BEGIN
14          IF CC[LA-3]=J THEN
15          BEGIN
16              IF QSTAT[J] THEN QUEUEOUT(J); FROMCH(LA,J)
17          END;
18          SCT[NO]:=SCT[NO]+SC[NO]*(NOW-SSC[NO]);
19          SSC[NO]:=NOW; SC[NO]:=SC[NO]+APL;
20          IF SC[NO]>SM[NO] THEN SM[NO]:=SC[NO];
21              SE[NO]:=SE[NO]+1;
22          ENT[CC[LA-4]]:=ENT[CC[LA-4]]+1;
23          CC[LA-4]:=CC[LA-4]+1; P:=SCAP[NO]-SC[NO];
24          IF P>0 THEN
25          BEGIN
26              L:=CC[3*J+1];
27              FOR DUMMY:=0 WHILE L≠NILL DO BEGIN IF CC[L-7]>P THEN L:= CC[L-1] ; END;
28              IF L≠NILL THEN
29              BEGIN
30                  NEXT:=L; TOSORT(LA,0,-1); GOTO CLOCK
31              END
32          END
33      END
34  END ENTER;
```

*** TAPE 368
*** GROEP 31
TOTAALNUMMER VAN DE EERSTE REGEL +492

```
    0   PROCEDURE LEAVE(NO,APL);
    1   VALUE NO,APL; INTEGER NO,APL;
    2   BEGIN
    3           IF NO<1 ∨ NO > NSTOR THEN BEGIN CARRIAGE(4);PRINTTEXT(‡LEAVE ERROR 2‡); JUMPOUT END;
    4           IF SC[NO]-APL<0 THEN
    5           BEGIN
    6                   CARRIAGE(4); PRINTTEXT(‡LEAVE ERROR 1‡);
    7                   ERRORDATA; PRINTTRANSACTION; PRINTOUT; JUMPOUT
    8           END;
    9           SCT[NO]:=SCT[NO]+SC[NO]*(NOW-SSC[NO]);
   10           SSC[NO]:=NOW; SC[NO]:=SC[NO]-APL;
   11           ENT[CC[LA-4]]:=ENT[CC[LA-4]] +1;
   12           CC[LA-4]:=CC[LA-4] +1;
   13           IF SPQ[NO]≠NILL THEN
   14           BEGIN
   15                   P:= SCAP[NO] -SC[NO]; L:= CC[3*SPQ[NO]+1]
   16           END;
   17                   EOR DUMMY:=0 WHILE L≠NILL DO BEGIN IF CC[L-7]>P THEN L:=CC[L-1]; END;
   18           IF L≠ NILL THEN
   19           BEGIN
   20                   NEXT:= L;  TOSORT(LA,0,-1); GOTO CLOCK;
   21           END
   22   END LEAVE;
```

*** TAPE 368
*** GROEP 32
TOTAALNUMMER VAN DE EERSTE REGEL +515

```
    0   PROCEDURE QUEUEIN(J);
    1   VALUE J; INTEGER J;
    2   BEGIN   N:=CC[3*J+2];
    3           QCT[J]:=QCT[J]+N*(NOW-QSC[J]);
    4           QSC[J]:=NOW;
    5           IF N> QM[J] THEN QM[J]:=N;
    6           QE[J]:=QE[J]+1;
    7   END QUEUEIN;
```

*** TAPE 368
*** GROEP 33
TOTAALNUMMER VAN DE EERSTE REGEL +523

```
    0   PROCEDURE QUEUEOUT(J);
    1   VALUE J; INTEGER J;
    2   BEGIN
    3           QCT[J]:=QCT[J]+CC[3*J+2]*(NOW-QSC[J]);
    4           QSC[J]:=NOW;
    5   END QUEUEOUT;
    6   INTEGER PROCEDURE LENGTH(J); VALUE J; INTEGER J;
    7   BEGIN
    8           IF J<0∨J>NQUEUE THEN BEGIN CARRIAGE(4);PRINTTEXT(‡LENGTH ERROR‡);JUMPOUT;END;
    9           LENGTH:=CC[3*J+2];
   10   END LENGTH;
```

```
*** TAPE   368
*** GROEP    34
TOTAALNUMMER VAN DE EERSTE REGEL          +534

   0   PROCEDURE TABULATE(J,X,NOX);
   1   VALUE J, X, NOX; INTEGER J, NOX; REAL X;
   2   BEGIN
   3           IF J<1 v J> NTABLE THEN BEGIN CARRIAGE(4);PRINTTEXT(‡ TABLE ERROR 1‡); JUMPOUT END;
   4           IF  NOX<0 THEN BEGIN CARRIAGE(4);PRINTTEXT(‡TABULATE ERROR 2‡); JUMPOUT END;
   5           ENT[CC[LA-4]]:=ENT[CC[LA-4]] +1;
   6           CC[LA-4]:=CC[LA-4]+1;
   7         IF NOX<0 THEN GOTO READY;
   8         NC:= HISTDATA[J,1];  LOW:= HISTDATA[J,2];  CW:= HISTDATA[J,3];
   9         P:= HIST[-J];
  10         IF X<LOW
  11         THEN CELL:= -1
  12         ELSE BEGIN CELL:= ENTIER( (X-LOW)/CW );
  13                    IF CELL>NC THEN CELL:= NC
  14              END;
  15         CELL:= P+6+CELL;
  16   FILL: HIST[CELL]:= HIST[CELL]+NOX;  HIST[P]:= HIST[P]+NOX;
  17         IF X<HIST[P+1] THEN HIST[P+1]:= X
  18       : IF X>HIST[P+2] THEN HIST[P+2]:= X;
  19         IF X#0 THEN BEGIN SUMX:= NOX*X;
  20                           HIST[P+3]:= HIST[P+3]+SUMX;
  21                           HIST[P+4]:= HIST[P+4]+SUMX*X
  22                     END;
  23   READY:
  24   END TABULATE;


*** TAPE   368
*** GROEP    35
TOTAALNUMMER VAN DE EERSTE REGEL          +559

   0   PROCEDURE PRINTTRANSACTION;
   1   BEGIN
   2           CARRIAGE(2);
   3           FOR I:=-7 STEP 1 UNTIL  NPAR DO BEGIN FIXT(2,0,I);PRINT(CC[LA+I]);NLCR END;
   4           CARRIAGE(3)
   5   END PRINTTRANSACTION;


*** TAPE   368
*** GROEP    36
TOTAALNUMMER VAN DE EERSTE REGEL          +565

   0   PROCEDURE PRINTTABLE(STRING,J);
   1   VALUE J; INTEGER J; STRING STRING;
```

```
258 -  16 -   280471 - 08163764PRINTPUNCH
```

```
    2  BEGIN
    3        INTEGER NC, N, I, DEC, P;
    4        REAL LOW, CW, CBEGIN, CEND, M, V
    5        ;
    6        PROCEDURE DIAGRAM(FR);
    7        VALUE FR; REAL FR;
    8        BEGIN
    9              INTEGER I, PRC2; REAL PRC1;
   10              PRC1:= 100*FR/N;  ABSFIXT(9, DEC, PRC1);
   11        PRC2:= PRC1; SPACE(2); PRSYM(127); PRSYM(93);
   12        FOR I:=1 STEP 1 UNTIL PRC2 DO PRSYM(66);
   13        END DIAGRAM
   14        ;
   15  INTEGER PROCEDURE DECIMALS(V);
   16        VALUE V; REAL V;
   17        BEGIN
   18              INTEGER DEC;
   19              DEC:= -1;
   20              FOR DEC:= DEC+1 WHILE ABS(V-ENTIER(V)) > 10&(-10)*V DO V:= V*10;
   21              DECIMALS:= DEC
   22        END DECIMALS
   23        ;
   24          IF J<1 ∨ J> NTABLE THEN BEGIN CARRIAGE(4);PRINTTEXT(£PRINTTABLE ERROR £); JUMPOUT END;
   25        P:= HIST[-J];
   26        N:= HIST[P];  NLCR;
   27        IF LINENUMBER+HISTDATA[J,1]>50 THEN NEWPAGE ELSE CARRIAGE(4);
   28        PRINTTEXT(STRING); ABSFIXT(2, 0, J); NLCR;
   29        IF N=0 THEN BEGIN NLCR; PRINTTEXT(£HISTOGRAM EMPTY£); GOTO READY END;
   30        IF N<0 THEN BEGIN NLCR; PRINTTEXT(£INCORRECT HISTOGRAM£) END;
   31        NC:= HISTDATA[J,1];  LOW:= HISTDATA[J,2];  CW:= HISTDATA[J,3];
   32        DEC:= DECIMALS(CW);
   33        NLCR; PRINTTEXT(£MEAN    £); FIXT(8,DEC+1,MEAN(J));
   34        NLCR; PRINTTEXT(£VARIANCE£); FIXT(8,DEC+1,VAR(J));
   35        NLCR; PRINTTEXT(£MINIMUM £); FIXT(8, DEC, HIST[P+1]);
   36        NLCR; PRINTTEXT(£MAXIMUM £); FIXT(8, DEC, HIST[P+2]);
   37        NLCR; PRINTTEXT(£ENTRIES £); FIXT(8,0,N); CARRIAGE(2);
   38        IF DEC≠0 THEN SPACE(DEC+1); PRINTTEXT(£     CELL£); IF DEC≠0 THEN SPACE(DEC+1);
   39        PRINTTEXT(£          ENTRIES     PERCENTAGE£); NLCR;
   40        IF DEC≠0 THEN SPACE(DEC+1);
   41        PRINTTEXT(£      <£); FIXT(4, DEC, LOW); ABSFIXT(9,0, HIST[P+5]); DIAGRAM(HIST[P+5]);
   42        CBEGIN:= LOW-CW;  CEND:= LOW-10&(-DEC);
   43        FOR I:=0 STEP 1 UNTIL NC-1 DO
   44        BEGIN NLCR;  CBEGIN:= CBEGIN+CW;  CEND:= CEND+CW;
   45              FIXT(4, DEC, CBEGIN);  PRINTTEXT(£-£);  FIXT(4, DEC, CEND);
   46              ABSFIXT(9, 0, HIST[P+6+I]);  DIAGRAM(HIST[P+6+I])
   47        END; NLCR;
   48        IF DEC≠0 THEN SPACE(DEC+1);
   49        PRINTTEXT(£      >£); FIXT(4, DEC, CEND); ABSFIXT(9,0, HIST[P+6+NC]); DIAGRAM(HIST[P+6+NC]);
   50  READY:
   51  END PRINTTABLE;
```

```
*** TAPE   368
*** GROEP   37
TOTAALNUMMER VAN DE EERSTE REGEL          +617
```

```
    0  PROCEDURE PRINTOUT;
```

```
BEGIN
        PRINTTEXT(¢CLOCK¢); SPACE(10); PRINT(NOW); CARRIAGE(4);
        IF OUTPUT(1) THEN
        BEGIN
                PRINTTEXT(¢BLOCK NO          ENTRIES¢); CARRIAGE(2);
                FOR J:=1 STEP 1 UNTIL NBLOCK DO
                BEGIN
                        ABSFIXT(7,0,J); ABSFIXT(14,0,ENT[J]); NLCR
                END;
                NEW PAGE;
        END;
        IF OUTPUT(2) THEN
        BEGIN
                PRINTTEXT(¢FACILITY          AVERAGE          ENTRIES¢); NLCR;
                PRINTTEXT(†                UTILIZATION†); CARRIAGE(2);
                FOR J:=1 STEP 1 UNTIL NFAC DO
                BEGIN
                        ABSFIXT(7,0,J); ABSFIXT(10,3,FCT[J]/NOW); ABSFIXT(14,0,FE[J]); NLCR
                END;
                CARRIAGE(4);
        END;
        IF OUTPUT(3) THEN
        BEGIN
                PRINTTEXT(¢ STORAGE        CAPACITY          AVERAGE          MAXIMUM          CURRENT          ENTRIES¢); NLCR;
                PRINTTEXT(¢                                  CONTENTS         CONTENTS         CONTENTS¢); CARRIAGE(2);
                FOR J:=1 STEP 1 UNTIL NSTOR DO
                BEGIN
                        ABSFIXT(7,0,J); ABSFIXT(14,0,SCAP[J]); ABSFIXT(10,3,SCT[J]/NOW);
                        ABSFIXT(14,0,SM[J]); ABSFIXT(14,0,SC[J]); ABSFIXT(14,0,SE[J]); NLCR
                END;
                CARRIAGE(4);
        END;
        IF OUTPUT(4) THEN
        BEGIN
                PRINTTEXT(¢   QUEUE          AVERAGE          MAXIMUM          CURRENT          TOTAL          ZERO¢); NLCR;
                PRINTTEXT(¢                  CONTENTS         CONTENTS         CONTENTS         ENTRIES        ENTRIES¢); CARRIAGE(2);
                FOR J:=1 STEP 1 UNTIL NQUEUE DO
                BEGIN IF  QSTA[J] THEN BEGIN
                        ABSFIXT(7,0,J); ABSFIXT(10,3,QCT[J]/NOW); ABSFIXT(14,0,QM[J]);
                ABSFIXT(14,0,LENGTH(J)); ABSFIXT(14,0,QE[J]); ABSFIXT(14,0,QZ[J]); NLCR;
                        END
                END;
                CARRIAGE(4);
        END;
        IF OUTPUT(5) THEN
        BEGIN
                NEW PAGE; FOR J:=1 STEP 1 UNTIL NTABLE DO PRINTTABLE(¢TABLE¢,J);
        END;
        NEW PAGE
END PRINTOUT;
```

*** TAPE 368
*** GROEP 3A
TOTAALNUMMER VAN DE EERSTE REGEL +668

```
0   REAL PROCEDURE PAR(I);
```

```
    1   VALUE I; INTEGER I;
    2   BEGIN
    3   IF I<1 ∨ I>NPAR THEN BEGIN CARRIAGE(4);PRINTTEXT(¢PAR ERROR¤); JUMPOUT END;
    4   PAR:= CC[LA+I];
    5   END PAR;
    6   INTEGER PROCEDURE TNO;
    7   TNO:=CC[LA-6];
    8   REAL PROCEDURE QTP(J,PL,P);
    9   VALUE J,PL,P; INTEGER J,PL,P;
   10   BEGIN
   11   IF J<1 ∨ J>NQUEUE∨PL<0∨P<0 THEN BEGIN CARRIAGE(4);PRINTTEXT(¢QTP ERROR¤); JUMPOUT END;
   12           L:=CC[3*J+1]; I:=0;
   13           FOR I:=I+1 WHILE I<PL∧L≠NILL DO L:=CC[L-1];
   14           QTP:=IF L=NILL THEN NILL ELSE CC[L+P];
   15   END PARAMETERVALUE OF SPECIFIC TRANSACTION IN QUEUE;
   16   REAL PROCEDURE TT;
   17   TT:=NOW-CC[LA-5];
```

*** TAPE 368
*** GROEP 39
TOTAALNUMMER VAN DE EERSTE REGEL +686

```
    0   REAL PROCEDURE MEAN(J);
    1   VALUE J; INTEGER J;
    2   BEGIN
    3           IF J<1 ∨ J>NTABLE THEN BEGIN CARRIAGE(4);PRINTTEXT(¢MEAN ERROR¤); JUMPOUT END;
    4           P:=HIST[-J]; N:=HIST'P];
    5           MEAN:=IF N=0 THEN NILL ELSE HIST[P+3]/N
    6   END MEAN;
    7   REAL PROCEDURE VAR(J);
    8   VALUE J; INTEGER J;
    9   BEGIN
   10           IF J<1 ∨ J> NTABLE THEN BEGIN CARRIAGE(4);PRINTTEXT(¢VAR, ERROR¤); JUMPOUT END;
   11           P:=HIST[-J]; N:=HIST'P];
   12           VAR:=IF N=0 THEN NILL ELSE IF N>1 THEN (HIST[P+4]-HIST[P+3]*HIST[P+3]/N)/(N-1) ELSE 0
   13   END VARIANCE;
   14   REAL PROCEDURE FUNC(NO,X);
   15   VALUE NO,X; INTEGER NO; REAL X;
   16   BEGIN
   17           IF NO<1 ∨ NO>NFUNC THEN BEGIN CARRIAGE(4);PRINTTEXT(¢FUNC ERROR¤); JUMPOUT; END;
   18           I:=0;
   19           FOR DUMMY:=0 WHILE X>FUNCV[NO,2*I+1] DO I:=I+1;
   20           XH:=FUNCV[NO,2*I+1];
   21           IF INF[NO,2]=0 THEN
   22           BEGIN
   23                   XL:=FUNCV[NO,2*I-1]; YL:=FUNCV[NO,2*I]; YH:=FUNCV[NO,2*I+2];
   24                   FUNC:=YL+(YH-YL)/(XH-XL)*(X-XL)
   25           END
   26           ELSE IF INF[NO,2]=1 THEN FUNC:=FUNCV[NO,2*I+2];
   27   END FUNC;
```

*** TAPE 368
*** GROEP 40

```
    0   PROCEDURE ERRORDATA;
    1   BEGIN
    2         CARRIAGE(2);  PRINTTEXT(*DATA OF ARRAY CC*);
    3         NLCR; PRINTTEXT(*  I    ARRAY[I]*);
    4         FOR J:=-5 STEP 1 UNTIL -1 DO BEGIN NLCR; FIXT(2,0, J); PRINT(CC[J]) END;
    5         P:= CC[-3]; NLCR;
    6         NLCR; PRINTTEXT(*LENGTHS OF CHAINS*);
    7         NLCR; PRINTTEXT(*  I    CHAIN[I]*);
    8         FOR J:=0 STEP 1 UNTIL P-1 DO
    9         BEGIN
   10               NLCR; ABSFIXT(2,0,J); N:=CC[3*J+2]; ABSFIXT(5,0,N);
   11               L:=0; LA:=CC[3*J+1];
   12               IF LA#NIL THEN
   13               BEGIN
   14                                 L:=1;
   15                         FOR LA:=CC[LA-1] WHILE LA#NIL DO L:=L+1;
   16               END;
   17               IF N#L THEN BEGIN PRINTTEXT(*ERRONEOUS      LENGTH =*); FIXT(5,0,L) END;
   18         END;
   19         CARRIAGE(4);
   20   END ERRORDATA;
```

*** TAPE 368
*** GROEP 15
TOTAALNUMMER VAN DE EERSTE REGEL +186

```
    0   PROCEDURE INITIALIZE(TE,LABEL);
    1   VALUE TE,LABEL; INTEGER LABEL; REAL TE;
    2   BEGIN
    3         IF TE<0 THEN BEGIN CARRIAGE(4);PRINTTEXT(*INITIAL ERROR 2*);JUMPOUT END;
    4         IF LABEL<1∨LABEL>NBLOCK THEN BEGIN CARRIAGE(4); PRINTTEXT(*INITIALIZE ERROR 1*); JUMPOUT END;
    5         NUM:=NUM+1;
    6         LA:=FROMF; CC[LA-5]:=TE; CC[LA-4]:=LABEL;
    7         CC[LA-6]:= NUM; TOSORT(LA,0,TE);
    8   END INITIALIZE;
```

*** TAPE 368
*** GROEP 16
TOTAALNUMMER VAN DE EERSTE REGEL +195

```
    0   PROCEDURE TABLE (NO,AKL,LOW,KLBR);
    1   VALUE NO,AKL,LOW,KLBR; INTEGER NO; REAL AKL,LOW,KLBR;
    2   BEGIN
    3         IF NO<1∨NO>NTABLE THEN BEGIN CARRIAGE(4); PRINTTEXT(*TABLE ERROR 1*); JUMPOUT END;
    4         IF AKL<0 ∨ KLBR<0 THEN BEGIN CARRIAGE(4);PRINTTEXT(* TABLE ERROR 2*); JUMPOUT END;
    5         HISTDATA[NO,1]:=AKL; HISTDATA[NO,2]:=LOW; HISTDATA[NO,3]:=KLBR;
    6         UBTABLE:=UBTABLE+HISTDATA[NO,1]+7;
    7   END TABLE;
```

```
*** TAPE   368
*** GROEP    17
TOTAALNUMMER VAN DE EERSTE REGEL          +203


   0   PROCEDURE QUEUEDATA;
   1   BEGIN   OUTPUT[4]:=TRUE;
   2   AGAIN :  P:=READ;
   3            IF P>0∧P≤NQUEUE THEN
   4             BEGIN QSTAT[P]:=TRUE; GOTO AGAIN END
   5   END QUEUEDATA;



*** TAPE   368
*** GROEP    18
TOTAALNUMMER VAN DE EERSTE REGEL          +209


   0   PROCEDURE STORAGE(NO,CAP);
   1   VALUE NO,CAP; INTEGER NO,CAP;
   2   BEGIN
   3            IF NO<1∨NO>NSTOR THEN BEGIN CARRIAGE(4); PRINTTEXT(£STORAGE ERROR 1£); JUMPOUT END;
   4             IF CAP<0 THEN BEGIN CARRIAGE(4);PRINTTEXT(£ STORAGE ERROR 2£);JUMPOUT END;
   5            SCAP[NO]:=CAP;
   6   END STORAGE;



*** TAPE   368
*** GROEP    19
TOTAALNUMMER VAN DE EERSTE REGEL          +216


   0   PROCEDURE FUNCTION(NO,AG,TYPE);
   1   VALUE NO,AG,TYPE; INTEGER NO,AG,TYPE;
   2   BEGIN
   3            IF NO<1∨NO>NFUNC THEN BEGIN CARRIAGE(4); PRINTTEXT(£FUNCTION ERROR 1£); JUMPOUT END;
   4            IF ENTIER(AG/2)≠AG/2 THEN BEGIN CARRIAGE(4); PRINTTEXT(£FUNCTION ERROR 2£); JUMPOUT END;
   5            IF TYPE≠0∧TYPE≠1 THEN BEGIN CARRIAGE(4); PRINTTEXT(£FUNCTION ERROR 3£); JUMPOUT END;
   6            INF[NO,1]:=AG; INF[NO,2]:=TYPE;
   7            IF AG>UBF THEN UBF:=AG;
   8   END FUNCTION;



*** TAPE   368
*** GROEP    20
TOTAALNUMMER VAN DE EERSTE REGEL          +225


   0   CC[-5]:=UBCC; CC[-4]:=NPAR +8; CC[-3]:= 1+NQUEUE;
   1   CC[-2]:=CC[-1]:=3*CC[-3];
   2   FOR I:=0 STEP 3 UNTIL CC[-2]-1 DO
   3   BEGIN
   4            CC[I]:=CC[I+1]:=NILL;
   5            CC[I+2]:=0
```

```
    END;
    IF OUTPUT[1] THEN
    BEGIN
            FOR J:=1 STEP 1 UNTIL NBLOCK DO ENT[J]:=0;
    END;
    IF OUTPUT[2] THEN
    BEGIN
            FOR J:=1 STEP 1 UNTIL NFAC DO
            BEGIN
                    SLA[J]:=PLA[J]:=FPSQ[J]:=FPPQ[J]:=NILL;
                    FCT[J]:=0; FS[J]:=FP[J]:=FE[J]:=0
            END
    END;
    IF OUTPUT[3] THEN
    BEGIN
            FOR J:=1 STEP 1 UNTIL NSTOR DO
            BEGIN
                    SCT[J]:=0; SC[J]:=SM[J]:=SE[J]:=0
                    ; SPQ[J]:=NILL
            END
    END;
    IF OUTPUT[4] THEN
    BEGIN
            FOR J:=1 STEP 1 UNTIL NQUEUE DO
            BEGIN
                    QCT[J]:=0; QC[J]:=QM[J]:=QE[J]:=QZ[J]:=0; QSTAT[J]:= FALSE;
            END
    END;
    OUTPUT[4]:=FALSE;
```


*** TAPE 368
*** GROEP 21
TOTAALNUMMER VAN DE EERSTE REGEL +260

```
    BEGIN
            REAL ARRAY      HIST[-NTABLE:UBTABLE], FUNCV[0:NFUNC,1:UBF];
            IF OUTPUT[5] THEN
            BEGIN
                    FOR I:=1 STEP 1 UNTIL UBTABLE DO HIST[I]:=0;
                    P:= 1;
                    FOR I:=1 STEP 1 UNTIL NTABLE DO
                    BEGIN
                            HIST[-I]:= P;
                            HIST[P+1]:= -.600;  HIST[P+2]:= -.600;
                            P:= P+HISTDATA[I,1]+7
                    END
                    HIST[0]:=NTABLE
            END;
            IF OUTPUT[6] THEN
            BEGIN
                    PRINTTEXT('FUNCTION                    TYPE                    X                    Y');  SPACE(4);
                    FOR J:=1 STEP 1 UNTIL NFUNC DO
                    BEGIN
                            ABSFIXT(7,0,J); SPACE(11);
                            IF INF[J,2]=0 THEN PRINTTEXT('CONTINUOUS') ELSE
                            IF INF[J,2]=1 THEN PRINTTEXT('DISCREET'); NLCR;
```

```
22                      EQB :=1 STEP 1 UNTIL INF[J,1]12 DO
23                      BEGIN
24                              SPACE(32); FIXT(13,5,(FUNCV[J,2*I-1]:=READ)); FIXT(13,5,(FUNCV[J,2*I]:=READ)); NLCR
25                      END;
26                      CARRIAGE(2)
27              END
28          END;
29          BEGIN
```

*** TAPE 368
*** GROEP 22
TOTAALNUMMER VAN DE EERSTE REGEL +290

```
 0  PROCEDURE GENERATE(T,LABEL);
 1  VALUE T,LABEL; INTEGER LABEL; REAL T;
 2  BEGIN
 3          IF T<0 THEN BEGIN CARRIAGE(4);PRINTTEXT(£GENERATE ERROR 2£);JUMPOUT END;
 4          IF LABEL<1 ∨ LABEL>NBLOCK THEN
 5          BEGIN
 6                  CARRIAGE(4); PRINTTEXT(£GENERATE ERROR 1£);
 7                  ERRORDATA; PRINTTRANSACTION; PRINTOUT; JUMPOUT
 8          END;
 9          NUM:=NUM+1;
10          L:=FROMF; CC[L-5]:=NOW +T;CC[L-4]:= LABEL;
11          CC[L-6]:= NUM; TOSORT(L,0,CC[L-5]);
12          ENT[CC[LA-4]]:=ENT[CC[LA-4]]+1;
13          CC[LA-4]:=CC[LA-4]+1;
14  END GENERATE;
```

*** TAPE 368
*** GROEP 23
TOTAALNUMMER VAN DE EERSTE REGEL +305

```
 0  PROCEDURE TERMINATE (M);
 1  VALUE M; INTEGER M;
 2  BEGIN
 3          IF M<0 THEN BEGIN CARRIAGE(4);PRINTTEXT(£TERMINATE ERROR£);JUMPOUT END;
 4          ENT[CC[LA-4]]:=ENT[CC[LA-4]] +1;
 5          P:= LA-7; CC[P]:= CC[-2]; CC[-2]:= P;
 6          READY:=READY+M;
 7          IF READY<NTRANS THEN
 8          BEGIN
 9                  IF READY=NREP*NPRINT THEN
10                  BEGIN
11                          NREP:=NREP+1; PRINTOUT
12                  END;
13                  GOTO CLOCK
14          END ELSE
15          BEGIN
16                  PRINTOUT; GOTO EINDE
17          END
18  END TERMINATE;
```

```
255 - 19 - 280471 - 08163764PRINTPUNCH

TOTAALNUMMER VAN DE EERSTE REGEL          +714

    0   CLOCK:  IF NEXT=NILL THEN
    1           BEGIN
    2                   LA:=CC[1]; FROMCH(LA,0); IF CC[LA]#-1 THEN NOW :=CC[LA]]
    3           END
    4           ELSE
    5           BEGIN
    6                   LA:=NEXT; NEXT:=NILL
    7           END;
    8           NEXTBLOCK:=CC[LA-4]; GOTO S[NEXTBLOCK]]



*** TAPE    368
*** GROEP    41
TOTAALNUMMER VAN DE EERSTE REGEL          +723

    0   EINDE:
    1                       END
    2                   END
    3           END
    4   END


      REEL NR     LAATSTE GROEP NR   VOLGEND ELEMENT NR   LAATSTE MODUS

         +368                  +41                  +64           LEES
```