Inleiding discrete simulatie : projekt O.R. 1

Syllabus

Inleiding Discrete Simulatie

Projekt O.R. I

Ir. J.A.G.M. Kerbosch en L.H. Kroep

Afdeling Bedrijfskunde i.o.

Technische Hogeschool Eindhoven

Literatuur.

1) Statistisch Compendium
behorend bij het college van prof. dr. H.C. Hamaker
onderafdeling der Wiskunde, T.H. Eindhoven.

2) Elementaire cursus programmeren in Algol 60
behorend bij het college Wiskunde IVb
onderafdeling der Wiskunde, T.H. Eindhoven.

3) Algol procedure RANDOM
R.C. informatie nr. 15
Rekencentrum, T.H. Eindhoven.

4) Trekkingen uit normale en exponentiele verdelingen
Algol procedure NORMAL
R.C. informatie nr. 16
Rekencentrum, T.H. Eindhoven.

5) Beschrijvingen voor de gebruiker van de systemen Algol 60 MCA en Algol 60 THE voor de EL X8
R.C. informatie nr. 30
Rekencentrum, T.H. Eindhoven.

6) Aanvullende beschrijving op R.C. Informatie nr. 30 met betrekking tot het systeem Algol 60 THE.
R.C. informatie nr. 32
Rekencentrum, T.H. Eindhoven

7) Fundamentals of Operations Research: Hfst 4.
Ackoff R.L. en Sasieni M.W.
Wiley Inc., New York.

8) Variance reduction techniques in Simulation:
Kleynen J.P.C.
dissertatie: Katholieke Hogeschool Tilburg, jan. 1971.

9) Simulation Programming Languages: Buxton J.N.
North Holland Publishing Company, Amsterdam.

Inhoudsopgave

Voorwoord.

De opzet van deze syllabus is:
- de lezer enigszins vertrouwd te maken met begrippen zoals die worden gehanteerd bij simulatiemodellen
- de lezer bij het ontwerpen van computerprogramma's voor discrete simulatie, het nodige gereedschap te verschaffen.

Na de behandeling van basisbegrippen wordt een pakket Algol-procedures geintroduceerd van eigen ontwerp ("ASP"-Algol Simulation Package).
Een aantal uitgewerkte voorbeelden zullen het geheel verduidelijken.

De procedures van het pakket "ASP" zijn tot stand gekomen in samenwerking met de heer R.W. Sierenberg, T.H. Delft. Wij danken hem voor zijn waardevolle suggesties en bijdragen.
De vergaderingen en discussies in de werkgroep "simulatietalen" van de Stichting "Het Nederlands studiecentrum voor Informatica" hebben stimulerend gewerkt.
Wij danken mej. A. Ummels voor de uitstekende verzorging van het typewerk.

1. Definitie van basisbegrippen.

1.1 Modellen (models).

In de wetenschap ontwerpt men een model van de realiteit. Aanvankelijk een verbaal model, later een mathematisch model. Bij mathematische modellen wordt het gedrag van de componenten uit de realiteit m.b.v. symbolen beschreven. In een model wordt slechts dat deel van de realiteit beschreven, dat voor het onderzoek van belang is. Het model is een afbeelding van de realiteit. Bij deze afbeelding kunnen afbeeldfouten gemaakt worden. De afbeeldfout is het verschil tussen realiteit en model. Wanneer men tot iedere prijs het maken van afbeeldfouten wenst te voorkomen, kan het model zeer gecompliceerd worden. Analyse van een dergelijk model is dan veelal niet meer mogelijk. Men streeft er derhalve naar de modellen zo eenvoudig mogelijk te maken. Men dient na te gaan welke afbeeldfouten gemaakt worden en in hoeverre deze afbeeldfouten de resultaten van het onderzoek beinvloeden.

Voorbeeld:

a) Realiteit: Aan een drukke weg staat een benzinestation, waar zich een pomp bevindt. Op het emplacement is hoogstens ruimte voor een wachtrij van p auto's. Men heeft de indruk, dat er vrij veel potentiele klanten doorrijden, omdat er geen plaats is op het emplacement.
Men vraagt zich af of vergroting van het emplacement gewenst is.

b) Model: Het benzinestation bestaat uit een emplacement waarop zich een pomp bevindt. Auto's arriveren op het emplacement.
De interaankomsttijd is negatief exponentieel verdeeld (met bv. gemiddelde 4). De helptijd bij een pomp is negatief exponentieel verdeeld (met bv. gemiddelde 3).
Indien er zich een wachtrij voor de pompen bevindt met lengte p, dan zal een aankomende auto het / verlaten zonder te tanken. In het andere geval voegt hij zich achteraan in de wachtrij (d.w.z. de rij- of queue-discipline is "first in, first out", afgekort fifo).
We nemen aan, dat de tijd nodig om zich al of niet in de wachtrij te voegen = 0 is en de tijd nodig om de wachtrij te verlaten en zich naar de pomp te begeven verdisconteerd is in de bedieningstijd.
Gevraagd wordt te berekenen: gemiddelde en variantie van wachttijden en het percentage doorrijders bij p = 3 en p = 4.

/ emplacement

1.1.1 Afbeeldfouten (model-errors).

1. de keuze van een negatief exponentiele verdeling voor tussenaankomsttijden en bedieningstijd met de gegeven parameters. We mogen aannemen, dat er waarnemingen zijn verricht, die deze veronderstellingen ondersteunen.
2. De tijd nodig om een doorrijdbeslissing te nemen en zich eventueel in de wachtrij te voegen neemt men = 0.
3. De tijd nodig om de wachtrij te verlaten en naar de pomp te rijden zal men rekenen bij de bedieningstijd.

We mogen aannemen, dat de laatste twee afbeeldfouten niet storend zijn. In het volgende zullen wij er naar streven de modellen zo eenvoudig mogelijk te maken, zonder dat de afbeeldfouten de resultaten ernstig kunnen beinvloeden. Het opsporen en beoordelen van de belangrijkheid van de afbeeldfouten wordt aan de lezer overgelaten als een eenvoudige oefening.

1.2 Systeem (system).

Een systeem is een verzameling componenten, waarbij tussen de componenten relaties bestaan.

Voorbeelden:

Systeem	Componenten	Relaties
benzinestation	auto's, pompen	de auto's tanken aan een pomp
postkantoor	klanten,loketten(beambten)	de klanten worden door de beambten aan de loketten geholpen
fabriek	orders,machines,arbeiders	de arbeiders voeren met behulp van de machines de orders uit

We onderscheiden:

1. statische en dynamische systemen. Bij de dynamische systemen is de tijd een essentiele variabele, bij de statische systemen niet.
2. deterministische en stochastische systemen. Bij de stochastische systemen is er tenminste een stochastische variabele, die het gedrag van het systeem beinvloedt.

Wij zullen ons slechts bezighouden met dynamische, stochastische systemen. De bovenstaande systemen zijn dynamische, stochastische systemen. In het vervolg zal onder systeem steeds verstaan worden een dynamisch, stochastisch systeem, tenzij uitdrukkelijk anders vermeld. De bijvoeglijke naamwoorden dynamisch en stochastisch zullen gemakshalve worden weggelaten.

1.2.1 Componentklassen (class of components).

De componenten van een systeem kan men indelen in klassen op grond van gelijksoortig gedrag. Bij het benzinestation gedragen alle auto's zich op soortgelijke wijze. De auto's vormen een klasse. Hetzelfde geldt voor de pomp(en). Bij dit voorbeeld zou naast een klasse van personenauto's nog een klasse van vrachtauto's onderscheiden kunnen worden, indien het gedrag van vrachtauto's essentieel anders zou zijn dan het gedrag van personenauto's.
Men onderscheidt tijdelijke en permanente componenten. In de voorbeelden uit de vorige paragraaf zijn auto's, klanten en orders de tijdelijke, de overigen de permanente componenten.

1.2.2 Systeemmodel (system model).

Een systeemmodel is een model van een systeem. Wij zullen het woord systeemmodel in hoofdzaak gebruiken om een mathematisch model van een systeem aan te duiden.

1.2.3 De toestand van een systeem, toestandsvariabelen (state, statevariables).

Een dynamisch systeem verkeert op ieder moment in een bepaalde toestand. Deze toestand kan gekarakteriseerd worden door een eindig aantal variabelen, de zgn. toestandsvariabelen. Bij een benzinestation bv. kan de toestand gekarakteriseerd worden door twee variabelen, één die aangeeft of de pomp bezet is en één die aangeeft hoeveel auto's zich in de wachtrij bevinden.

1.2.4 Discrete en continue systemen (discrete and continuous systems).

Een systeem is een discreet systeem wanneer de toestand van het systeem slechts op een eindig aantal momenten verandert. Tussen twee opeenvolgende momenten verandert de toestand van het systeem niet. De veranderingen zelf verlopen tijdloos.
Bovengenoemde voorbeelden (zie 1.2) zijn voorbeelden van discrete systemen.
De componenten van een discreet systeem doorlopen scherp gescheiden fasen. Wanneer een component overgaat in een volgende fase verandert de toestand van het systeem. Dit zijn de enige momenten waarop de toestand verandert. Het systeem gaat als het ware schoksgewijs van de ene toestand over in de volgende toestand. Bij een benzinestation bv. worden de toestandsveranderingen veroorzaakt door aankomst resp. vertrek van

een auto. De auto's doorlopen de scherp gescheiden fasen; wachtend en in bediening, de pomp doorloopt de fasen; bezig en niet bezig.
Men spreekt van een continu systeem, wanneer de toestand van het systeem in de tijd continu verandert. De groei van planten en thermische processen zijn voorbeelden van continue systemen.
Verkeersafwikkeling is een gemengd continu systeem, d.w.z. op discrete momenten verandert de toestand van het systeem ingrijpend bv. bij rood worden van een stoplicht. Tussen deze momenten in verandert de situatie min of meer continu (passeren van auto's). Soms is het door een geschikte keuze van het model toch mogelijk te werken alsof dit zuiver discrete systemen zijn.

1.3 Discrete en continue simulatie.

Simuleren is het nabootsten van de realiteit. Meestal vertoont deze imitatie een grote gelijkenis met de realiteit. De graad van abstractie is bij simulatie veelal geringer, dan bij deductieve analyse.
Simuleren heeft in het spraakgebruik veelal een ongunstige betekenis.
Simulatie in de hier gebruikte wetenschappelijke betekenis is het doen van experimenten, waarin de reeële situatie nagebootst wordt op een kleinere en beter betaalbare schaal.
Het doel van simulatie is het doen van betere voorspelling over toekomstige reeële situaties. Wanneer we de definitie van simulatie zo ruim nemen is het vakgebied bijna onbegrensd, immers bijna alle wetenschappelijke experimenten vallen eronder. Wij willen het vakgebied begrenzen en zullen de term simulatie in het vervolg gebruiken voor een beperkter terrein:
nl. Computer-simulatie.
Dit is een techniek om op een computer experimenten uit te voeren met mathematische modellen, die symbolisch het gedrag van een systeem beschrijven.
Bij computer simulatie zullen we ons beperken tot het simuleren met behulp van digitale rekenmachines van discrete systemen. Korter: discrete simulatie.
Het is ook mogelijk continue systemen te simuleren, dit komt meestal neer op het oplossen van een stelsel differentiaal vergelijkingen. Dit kan in het algemeen gebeuren met een analoge rekenmachine, hoewel het ook mogelijk is continue systemen te simuleren op een digitale rekenmachine. In wezen komt dit neer op het numeriek oplossen van een stelsel differentiaal vergelijkingen.

1.4 Event, eventtijdstip (event, eventtime).

De term event zou men kunnen vertalen met gebeurtenis. De term is echter in het vakjargon reeds dermate ingeburgerd, dat vertaling ons niet opportuun leek.

Het moment waarop de toestand van een (discreet) systeem verandert noemt men een eventtijdstip. De gebeurtenis, die aanleiding geeft tot een verandering noemen we een event. We nemen in de modellen aan, dat de veranderingen zich instantaan voltrekken, d.w.z. de veranderingen zelf kosten geen tijd.

De events kan men indelen in klassen op grond van het gelijksoortig zijn van de veranderingen of acties, die plaats vinden. Bij het benzinestation zijn de aankomsten van de auto's events van dezelfde klasse, immers al deze events geven aanleiding tot een soortgelijke verandering van het systeem nl. verlenging van de wachtrij c.q. bezetting van een pomp. Het vertrek van een auto is een event van een andere klasse, immers dit event geeft aanleiding tot vrijgave van de pomp c.q. bezetting van de pomp door de eerste wachtende.

1.5 Proces.(process).

Een proces is de individuele realisatie van een gedragspatroon van een bepaalde component. Kortheidshalve zullen we de individuele realisatie van een gedragspatroon "het gedrag" noemen. Dit gedrag wordt bepaald door het gedragspatroon van de component en zijn relatie tot de overige componenten. Het gedrag van de auto, die om 10.00 uur aankwam en een bedieningstijd had van 5.00 minuten is een proces. Het gedrag van de pomp is ook een proces.

Wanneer het gedrag van een component stochastisch is, spreekt men van een stochastisch proces. Wij zullen ons slechts bezighouden met stochastische processen en niet met deterministische processen.

1.6 Attribuut (attribute).

Een component wordt nader gespecifeerd door zijn attributen. De attributen zijn de waarden van de parameters, die nodig zijn voor de beschrijving van het proces. Bij de auto's zijn dit het moment van aankomst en de bedieningstijd. Componenten, die tot één klasse behoren verschillen alleen t.a.v. hun attributen.

1.7 Notitie (notice).

Bij simulatie op een digitale rekenmachine zal men van verschillende dingen notities willen maken. Het is nuttig een streng onderscheid te maken tussen het ding zelf en de notitie (aantekening, weergave) van het ding. Er is immers ook een verschil tussen een afspraak zelf en de notitie van de afspraak in uw agenda. Dit onderscheid wordt niet altijd even scherp gemaakt. Men kan o.a. de volgende soorten notities onderscheiden:
eventnotitie, attribuutnotitie, componentnotitie, procesnotitie.
Deze notities kunnen bestaan uit variabelen en/of statements.

1.8 Beschrijvingen.

Alvorens iets te kunnen nabootsen (simuleren) zal men moeten beschikken over een beschrijving van het te simuleren object (systeem). Bij discrete simulatie blijkt het buitengewoon belangrijk op welke manier men een beschrijving van het systeem maakt.
Er zijn van een discreet systeem twee beschrijvingswijzen mogelijk:
1) procesbeschrijving
2) eventbeschrijving.
Simulatietalen zijn óf gebaseerd op een procesbeschrijving óf op een eventbeschrijving. In algemene talen als Algol 60 en Fortran is men vrij in de keuze welke van de twee beschrijvingen men als uitgangspunt kiest.
Bij deze laatsten is een eventbeschrijving wel het meest voor de hand liggend.
In de volgende twee paragrafen zullen beide beschrijvingswijzen nader uiteen gezet worden.

1.8.1 Procesbeschrijving.

Een procesbeschrijving is de beschrijving van het gedragspatroon van een component uit een bepaalde klasse, rekening houdend met relaties tot alle andere componenten.
Bij het benzinestation kan een procesbeschrijving van de auto's bv. als volgt luiden:

A: de auto's arriveren volgens een bepaald inputproces, bv. een Poisson-inputproces, d.w.z. hun tussenaankomsttijden zijn negatief exponentieel verdeeld.

B: na aankomst neemt de automobilist de beslissing: doorrijden (zie F) of niet (zie C). Deze beslissing is afhankelijk van de rijlengte.

C: de auto voegt zich achteraan in de wachtrij.

D: de auto blijft in de wachtrij totdat al zijn voorgangers de wachtrij verlaten hebben en de pomp vrij is.

E: de auto verlaat de wachtrij, rijdt naar de pomp en tankt. De bedieningstijd is een trekking uit een kansverdeling bv. een negatief exponentiele verdeling.

F: de auto verlaat het terrein.

We nemen aan, dat het rijden naar de pomp (E) en de fasen A, B, C en F tijdloos zijn.

We zien, dat deze procesbeschrijving grote overeenkomst vertoont met de min of meer intuitieve beschrijving van paragraaf 1. Een procesbeschrijving van een systeem is een natuurlijke en voor de hand liggende wijze van beschrijven.

Wanneer we bij deze beschrijving nog vermelden, dat er een pomp is, dan ligt het gedrag van het gehele systeem vast. Het gedrag van de pomp is reeds impliciet beschreven door de procesbeschrijving van de auto's. Indien de pomp storingen zou vertonen volgens een bepaalde kansverdeling, dan legt een beschrijving van het gedrag van de auto's nog niet het gedrag van het gehele systeem vast.

Een procesbeschrijving van een systeem bestaat uit procesbeschrijvingen van een voldoend aantal componentklassen van dit systeem. Het gedrag van het gehele systeem dient hierdoor vastgelegd te worden. Meestal is het niet nodig een procesbeschrijving van alle klassen van componenten te geven. Soms is het voldoende procesbeschrijvingen te geven van de klassen van tijdelijke componenten (zoals bij GPSS). Het gedrag van de componenten uit de niet beschreven klassen is dan impliciet vastgelegd door de gegeven procesbeschrijvingen.

1.8.2 Eventbeschrijving.

De beschrijving van de toestandsveranderingen die bij events van een bepaalde klasse behoren, noemt men een eventbeschrijving. Bij het benzinestation kennen we twee eventklassen resp. aankomst en vertrek van een auto.

De bijbehorende eventbeschrijvingen kunnen als volgt luiden:

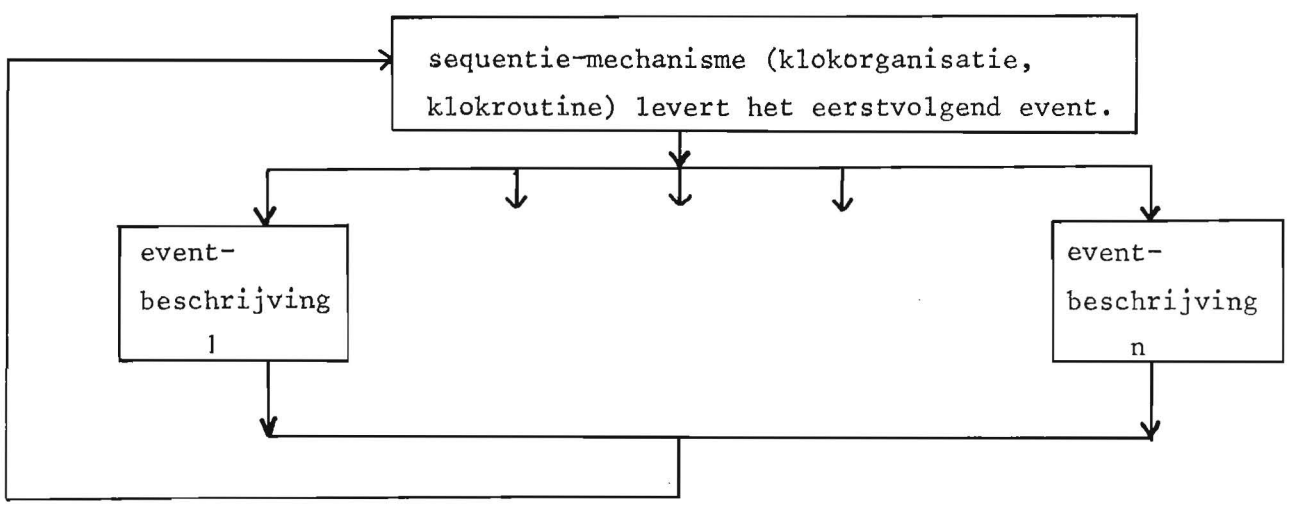
1. aankomst: Indien de pomp vrij is wordt deze bezet
   anders wordt de wachtrij één langer.

2. vertrek: de pomp komt vrij. Indien er andere auto's in de wachtrij staan, dan gaat de voorste auto uit de wachtrij naar de pomp. Indien er geen andere auto's in de wachtrij staan gebeurt er niets.

De werking van het systeem is hiermee nog niet beschreven. Er is nl. nog niet beschreven op welke tijdstippen de diverse events plaatsvinden. Wanneer we zouden beschikken over een lijst van alle events, dan zou een systeembeschrijving als volgt kunnen luiden: "werk de lijst van events in chronologische volgorde af en breng in de eventbeschrijving voorgeschreven toestandsveranderingen aan".
Het ontwerpen van een volledige lijst van events is omslachtig en onoverzichtelijk, zoniet onmogelijk.
We zullen een andere tactiek moeten volgen.
Merk op, dat wij op ieder moment alleen geinteresseerd zijn in het eerstvolgende event. Het is mogelijk de lijst van events beperkt te houden. We moeten er alleen voor zorgen, dat deze lijst altijd tenminste één event (het eerstvolgende) bevat. We zullen hiertoe een mechanisme moeten beschrijven, dat er voor zorgt dat: de lijst van toekomstige gebeurtenissen zo klein mogelijk en toch voldoende groot is.
Schematisch is de gang van zaken als volgt:

1. sequentiemechanisme: selecteer het eerstvolgend event.
   Dit kan in ons geval zowel een aankomst als een vertrek zijn.
2. werk het onder 1 geselecteerde event af.
   Verwijder dit event uit de lijst. Voeg indien nodig een nieuw event aan de lijst toe.
3. ga naar 1.

Het algemeen blokschema voor discrete simulatie, uitgaande van eventbeschrijvingen ziet er als volgt uit:



Het bepalen van toekomstige eventtijdstippen dient in de eventbeschrijvingen te worden opgenomen.

Bij het benzinestation bv.

- bij aankomst: bepaal door loting de volgende aankomst
- bij vertrek: bepaal door loting het vertrekmoment van de auto die hierna geholpen gaat worden (indien aanwezig).

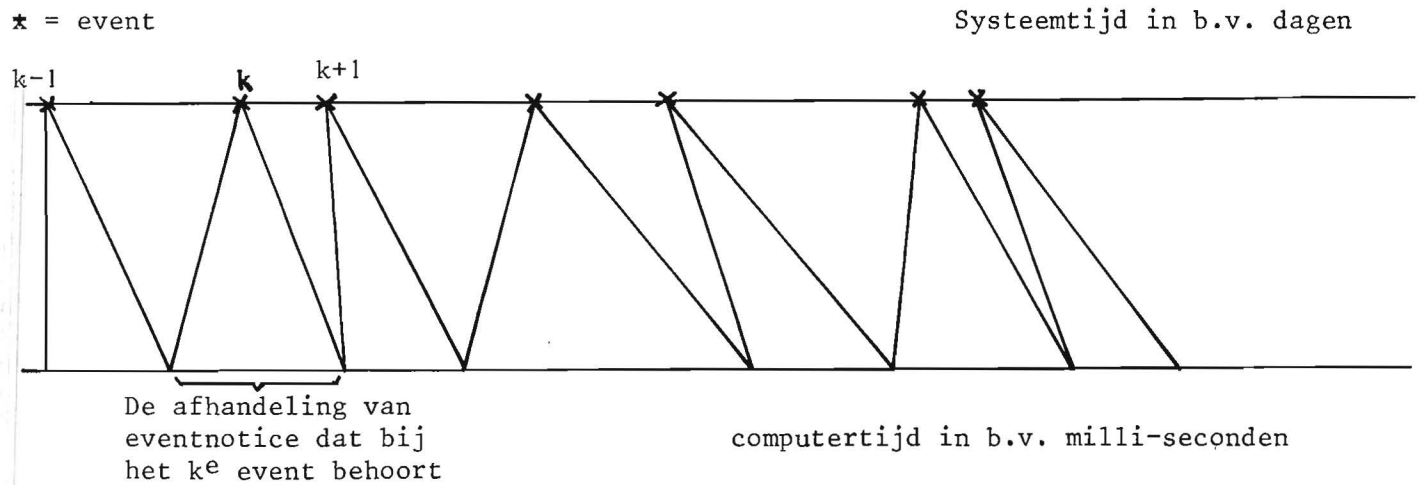
1.9 Computer simulatie van discrete systemen.

In het systeem zijn meerdere componenten tegelijkertijd bezig dingen te doen. De computer is op zichzelf echter één ding en kan slechts één ding tegelijk doen. De computer is een sequentiële processor.

Bij computer simulatie lost men dit als volgt op: In het systeem verandert de toestand tussen twee opeenvolgende eventtijdstippen niet. Deze tussenliggende tijd kunnen we in de computer a.h.w. overslaan.

In het systeem zijn de events tijdloos. Bij een event verandert echter de toestand van het systeem. In de simulatie zullen wij dan een eventnotice in behandeling moeten nemen. Deze behandeling kost computertijd.

Uit het bovenstaande is duidelijk, dat er een streng onderscheid dient te worden gemaakt tussen systeemtijd en computertijd. Het volgende schema moge dit verduidelijken:



Merk op dat:

1. - de handelingen van de computer dezelfde volgorde hebben als de events
   Dit is een andere volgorde dan die van de zinnen in de procesbeschrijving van 1.8.1
   Toch zijn sommige simulatietalen gebaseerd op een procesbeschrijving

(G P S S-Simula). Deze talen hebben "onder water" een sequentie-mechanisme. Men dient te beseffen, dat de volgorde van handelingen van de computer totaal afwijkt van de volgorde van de statements in de programma's in deze talen.

2. - de afhandeling van een eventnotice uit meerdere computerhandelingen kan bestaan.
   Bijgevolg kan het gesimuleerde systeem tijdens de afhandelingen een situatie representeren, die in het reële systeem niet bestaat, bv. de pomp is vrij en er is een wachtrij. Een dergelijke situatie noemen we een irreële situatie (zie 1.11.1.2).

Het ontwerpen van een computerpackage dat een eventbeschrijving accepteert is vrij eenvoudig.
Dat het werken met zo'n package niet buitengewoon moeilijk is, moge de rest van deze syllabus bewijzen.

1.10 Computer-talen.

Naarmate de computer bij meer vakgebieden wordt ingeschakeld, ontstaat de behoefte om talen te ontwerpen die voor een bepaald vakgebied een bijzondere geschiktheid hebben.
Men kan de computertalen onderscheiden in "general purpose" talen en "special purpose" talen. De simulatietalen zijn voorbeelden van een "special purpose" taal. We kunnen bij de simulatietalen onderscheid maken in "echte" talen en packages.
Bij "echte" talen gelden eigen syntactische regels.
Een package is een verzameling procedures (Algol) of subroutines (FORTRAN).
De voordelen van een package zijn:

1. In de programma's blijven alle faciliteiten van de algemene taal behouden. Ditzelfde geldt alleen voor simulatietalen, die een echte uitbreiding zijn van een algemene taal zoals Simula.
2. Een package is i.h.a. veel beter te definieren omdat men uitgaat van een goed gedefinieerde taal. Voor de geinteresseerde gebruiker zijn de procedures of subroutines leesbaar.
   Het sluitend definieren van een "echte" taal is geen eenvoudige zaak.

Er zal u een package van eigen ontwerp (ASP) in Algol 60 gepresenteerd worden.

In de hierna volgende overzichten zullen enkele general en special purpose talen genoemd worden.

General purpose talen:

Naam	Betekenis van naam	Opmerkingen	Literatuur
ALGOL 60	Algorithmic Language	Zeer geschikt voor wiskundige problemen. Op X8 mogelijk	"Elementaire cursus programmeren in ALGOL 60" behorend bij het college wiskunde IVb van prof. dr. E.W. Dijkstra (introductie). Revised report on the algorithmic language ALGOL 60 Numerische Math.4, 420-453, 1963 (basis) Mc Cracken:"A guide to ALGOL-programming" New York, Wiley 1963.
FORTRAN	Formula Translation	Geschikt voor vele wiskundige problemen. Veel uitzonderingen. Daardoor minder eenvoudig te leren als ALGOL. Minder algemeen, meer machine-oriented. Vele series: Fortran II, Fortran IV, etc. Op X8 mogelijk.	Manuals van computer-fabrikanten. Mc Cracken "A guide to Fortran-IV programming" New York, Wiley 1963.
COBOL	Common Business Oriented Language	Geschikt voor administratieve problemen. File-handling	Manuals van computer-fabrikanten. Mc Cracken "A guide to COBOL-programming"
PLI	Programming Language I	Door IBM gebracht als vervanger van Fortran.	Manuals van IBM bij machines uit de 360-serie. G.M. Neinberg "PL I programming primer" Mc Graw Hill. (gevorderd)
ALGOL'68	Algorithmic Language 68	Een voorgestelde uitbreiding van ALGOL-60 o.a. met lijstverwerkende faciliteiten. Niet internationaal aanvaard. Er bestaat géén compiler	v.Wijngaarde A.O. "Algorithmic Language ALGOL-68" Amsterdam, Math. Centrum 1968
SIMULA'67	Simulation Language 67	Een uitbreiding van ALGOL met lijstfaciliteiten. Op enkele machines compilers aanwezig. De naam is verwarrend, het is een algemene taal, waarmee ook simulatieproblemen handig aan te pakken zijn.	"Simula 67, Common Base Defenition". 2nd edition O.J. Dahl and K. Nygaard, Norwegian Computing Centre, 1971.

1.10.2

Special purpose talen

Naam	betekenis van naam	Uitsluitend geschikt voor	Literatuur
APT	Automatically Programmed Tools	Numerieke besturing	J.J. Vlietstra: De APT programmeer taal voor numerieke besturing van gereedschapswerktuigen Philips Techn. Tijdschrift 1967 no. 11.
LISP 1.5	List-processing	Geschikt voor het manipuleren met lijststructuren	J.Mc Carthy "LISP 1.5 programmer's manual" The M.I.T. Press 1962 (gevorderd). E. Berkeley and D.G. Bobrow:"The programming language LISP, its operation and applications" Inform. Intern. Inc., Cambridge (Mass) 1964 (introduction)
FORMAN	Formula-Manipulation	Het manipuleren met formules o.a. differentieren. Op EL-X8 mogelijk	R.P. van Riet: "Algol 60 as formula manipulation language". Amsterdam. Math. Centrum 1967.
AUTOMATH	Automatic mathematics	Het controleren van wiskundige bewijzen (in ontwerp) Op EL-X8 mogelijk.	N.C. de Bruyn "Automath, a language for mathematics" T.H. Report, 68 wsk-05, nov. 1968
Simulatie-talen		Simulaties	zie overzicht, 1.10.3

1.10.3 D = Diskrete simulatie T = taal E = eventbeschrijving

Simulatietalen C = continue P = package Pr= procesbeschrijving

Naam					Opmerkingen	
C S L	Control and Simulation Language	D	E	T	Speciale sequentie-routine	Buxton J.N. and Laski J.G. (1962) "Control and Simulation language". The computer journal. Vol.5, p. 194. CSL Reference Manual (1965) IBM United Kingdom Ltd.
SIMULA	Simulation Language	D	Pr	T	Een syntaktische uitbreiding van Algol'60 met een "procesconcept"	Dahl, O.J. and K. Nygaard:"Simula", A. Language for programming and discription of discrete event system" Norwegian Computer Center Forsknings-Veien 1b. Oslo, 4th ed. 1967, 118 pp.
GASP II	A General Activity Simulator Package	D	E	P	in "Fortran" geschreven	Pritsker, A.A.B., Kiviat, P.J. "Simulation with GASP II Prentice Hall 1969
GPSS	General Purpose Systems Simulator	D	Pr	T	Veel gebruikt. Het is een onvolledige procesbeschrijving. Iets gekompliceerde gevallen geven aanleiding tot zeer ingewikkelde programma's	Efron, P. and Gordon, G. "A general purpose digital simulator, and example of its application IBM. Sys.J. 3 (1964) 1, p 22 - 34.
A S	An Algol Simulation Language	D	E	P	in "Algol" geschreven	Parslow R.D. "AS users" Manual. Brunel University 1967
SIMSCRIPT II		D	E	T		Kiviat, P.J. Villanueva, R. and Makowitz, H.M. "The simscript II programming language" Prentice Hall 1969
ASP	Algol Simulation Package	D	E	P	in Algol geschreven	zie Syllabus
DYNAMO	Dynamic System Modelling	D	-	T	Ontworpen om Forresters "Dynamic Systems" te simuleren	Pugh A.L. "Dynamo user's manual" M.I.T. Press Cambridge, Mass. 1961
CSMP	Continuous System Modelling Program	C	-	T	CSMP is een "two-pass language dwz.het CSMP programma wordt eerst omgezet in een FORTAN-programma, waarna pas de eigenlijke vertaling in machinetaal kan beginnen. Men kan gebruik maken van FORTRAN IV Statements	1) IBM-application program GH20-0367-3: System/360 CSMP user's manual 2) S/360 CSMP gebruiksaanwijzing J.W.Andriesse afd. E. T.H. Twente

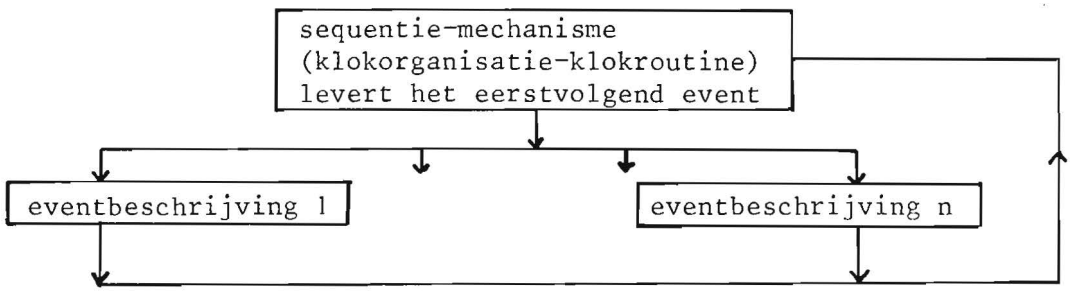
1.11 Uitwerking van de eventbeschrijving.

Deze uitwerking kan geschieden m.b.v. stroomschema's of pseudo-algol. Voor ons doel zijn beide uitwerkingen even goed bruikbaar. In de meer gecompliceerde gevallen waarbij bv. gebruik gemaakt wordt van recursie is een stroomschema niet bruikbaar meer.

1.11.1 Stroomschema's.

1.11.1.1 Variant 1:

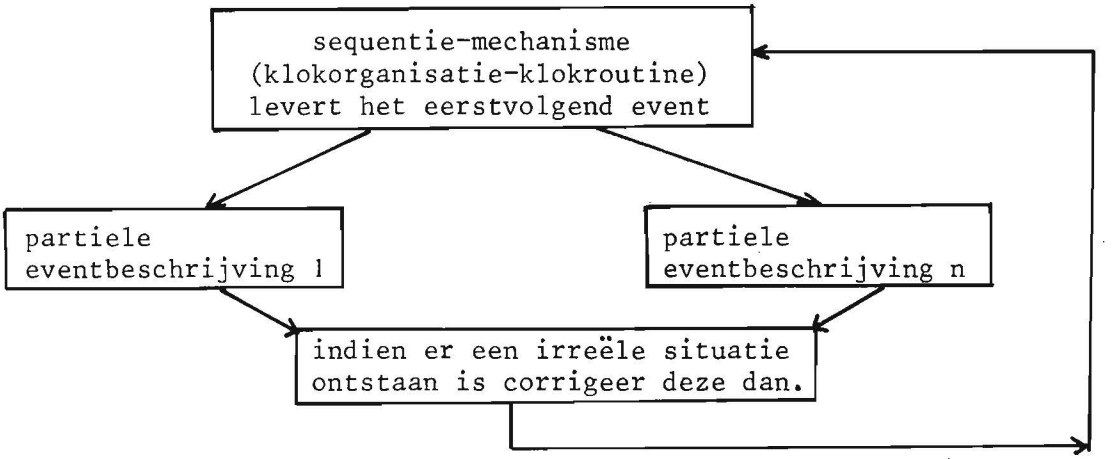
Bij de eventbeschrijving is het onderstaande, eveneens onder 1.8.2 beschreven algemene stroomschema, van toepassing.



Vaak blijken de individuele eventbeschrijvingen een grote mate van overeenkomst te vertonen.
Het kan bv. zo zijn dat ze slechts in parameters veranderen. De eventbeschrijving van aankomsten van een order van soort 1 en die van soort 2 verschillen slechts in geringe mate.
Men kan de aankomsten van verschillende ordersoorten onderbrengen in één eventbeschrijving: de aankomst van een order.
Deze eventbeschrijving heeft als parameter: de ordersoort. Ditzelfde kan men doen voor de eventbeschrijving van het gereed komen van machines. Hierdoor kan men zelfs in een vrij ingewikkeld systeem volstaan met een gering aantal eventsoorten, waardoor het stroomschema eenvoudig blijft.

1.11.1.2 Variant 2:

Soms blijken de laatste gedeelten van de eventbeschrijvingen gelijk te zijn. Deze luiden bv.: indien er een machine vrij is en er is een wachtrij voor deze machine, bezet deze dan. Deze irreële situatie (zie 1.9) kan zowel bij aankomst van een nieuwe order als bij het gereed komen van een machine ontstaan zijn. In een dergelijke situatie is ook het volgende stroomschema mogelijk.



In de praktische problemen is variant 2 altijd toepasbaar en o.i. eleganter programmeerbaar. In de voorbeelden in deze syllabus wordt steeds variant 2 gebruikt. Variant 1 is hier slechts om didactische redenen behandeld.

1.11.2 Pseudo-Algol.

I.p.v. stroomschema's kunnen we ook gebruik maken van pseudo-algol. Hierbij houden we ons niet strikt aan de regels van Algol.

1.11.2.1 Variant 1:

```
clock: klokroutine;
       i = eventsoort;
           case i of begin
eventbeschrijving 1: ......
       :             ......;
       :
       :
eventbeschrijving n: ......
                     ......;
                        end;
           goto clock;
```

1.11.2.2 Variant 2:

```
clock: klokroutine;
       i: = eventsoort;
       case i of begin
partiele eventbeschrijving 1: ......
           :                 ......;
           :
           :
partiele eventbeschrijving 2: ......
                             ......;
                          end;
       if irreële situatie then corrigeer;
       goto  clock;
```

2. Trekkingen uit verdelingen.

2.1. Random-pseudo random numbers.

Random numbers (aselecte getallen) zijn getallen die met behulp van een eerlijk lotingsmechanisme kunnen worden verkregen b.v. met een tienkantige zuivere dobbeltol. In een computer hebben we niet de beschikking over een dergelijk lotingsmechanisme.

Een van de meest opvallende eigenschappen van een goede computer is juist dat het resultaat van een berekening voorspelbaar (deterministisch) is. Een computer werkt deterministisch en niet stochastisch. Het genereren van random-getallen op een computer schijnt een onoplosbaar probleem. Juist bij simulaties hebben we voortdurend behoefte aan random numbers. Ter doorbreking van deze impasse heeft men zich afgevraagd of het niet mogelijk zou zijn een reeks te verzinnen, zodanig dat deze reeks niet of nauwelijks te onderscheiden is van een reeks van random numbers. Wij zouden dan een reeks pseudo-random numbers verkrijgen. Dergelijke reeksen bestaan inderdaad. Een voorbeeld van een dergelijke reeks die pseudo-random getallen tussen 0 en 1 oplevert, is:

$$x_{n+1} = a * x_n \pmod 1$$

bij geschikte keuze van a en x_1. Deze geschikte keuze hangt mede af van de woordlengte en het decimaal of binair rekenen van de computer. Wij gaan hierop niet nader in.
Een voor de EL-X8 geschikte randomgenerator staat beschreven in RC-Informatie nr. 15. [3]

2.2. Trekkingen uit discrete verdelingen.

Stel de stochastische variabele x heeft een Bernouilli verdeling, d.w.z. P (succes) = p, P (geen succes) = 1-p. Succes kan zijn kruis gooien met een munt, 6 gooien met een dobbelsteen etc. Een trekking uit een dergelijke verdeling is als volgt te verkrijgen.

succes: = random $\leq$ p

waarbij succes een Boolean variabele is.
Volgens ditzelfde principe zijn trekkingen uit iedere willekeurige discrete verdeling te verkrijgen.

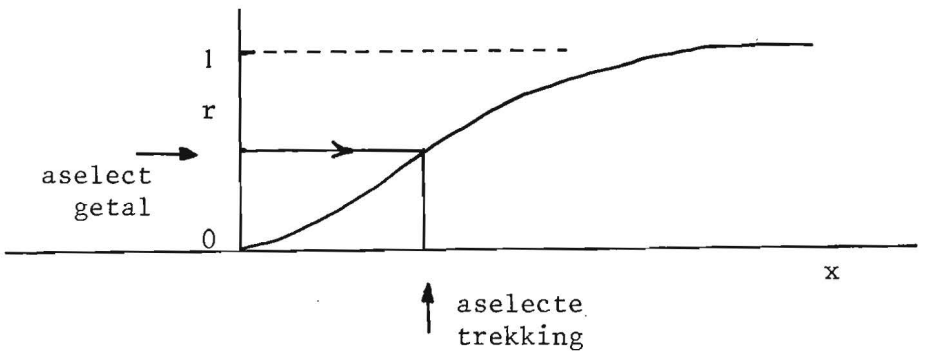
2.3. Trekkingen uit continue verdelingen.

Indien men beschikt over een generator van aselecte getallen kan men deze gebruiken voor het verkrijgen van aselecte trekkingen uit andere verdelingen door selectie en transformatie van deze aselecte getallen. Hiervoor bestaan een aantal algemene methoden, waarvan wij slechts de directe methode (transformatie) zullen bespreken.
Aangezien in wachttijdproblemen de helptijden en de tussenaankomsttijden veelal een exponentiele verdeling hebben zijn wij voornamelijk in trekkingen uit deze verdeling geïnteresseerd.

2.3.1. De directe methode

Indien F(x) de (cumulatieve) verdelingsfunctie is van een continue verdeling waaruit een aselecte trekking gevraagd wordt dan is de directe methode:

1. Trek een aselect getal r.
2. Bepaal x uit de vergelijking $r = F(x)$, dus $x = F^{-1}(r)$.
   x is dan de gevraagde trekking.



Trekkingsprocedure bij directe methode.

Indien we kunnen aantonen dat $P(\underline{x} \leq x) = F(x)$ waarbij x de aselecte trekking voorstelt, dan is procedure juist.
Dit volgt uit,

$$P(\underline{x} \leq x) = P(\underline{r} \leq F(x)) = F(x)$$

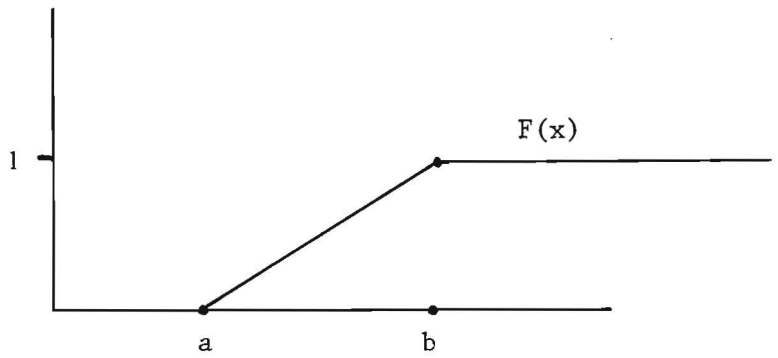
De directe methode is eenvoudig indien de inverse functie $F^{-1}(x)$ van de verdelingsfunctie gemakkelijk te bepalen en te berekenen is
Voorbeelden worden behandeld in 2.3.1.1. en 2.3.1.2.

2.3.1.1. Trekkingen uit de homogene verdeling op (a, b).

$\underline{x}$ heeft een homogene verdeling op het interval a,b (a < b);
de kansdichtheid $f(x) = \frac{1}{b-a}$ voor $a \leq x \leq b$, elders $f(x) = 0$

Voor de cumulatieve verdelingsfunctie geldt

$$F(x) = \frac{x}{b-a} - \frac{a}{b-a} \quad \text{voor } a \leq x \leq b$$
$$0 \text{ voor } x < a$$
$$1 \text{ voor } x > b$$



$r = F(x) \rightarrow x = F^{-1}(r) \rightarrow x = (b-a)r+a$

2.3.1.2. Trekkingen uit de exponentiele verdeling met gemiddelde $\mu = 1/\lambda$.

De kansdichtheid:

$f(x) = \lambda e^{-\lambda x}$ $\quad x \geq 0$ (zie [1] stat.comp.: pag.10)

De (cumulatieve) verdelingsfunctie:

$$F(x) = \int_0^x f(t)dt = \int_0^x \lambda e^{-\lambda t} dt = -e^{-\lambda t} \Big|_0^x = 1-e^{-\lambda x}$$

$r = 1-e^{-\lambda x}$ $\quad x = -1/\lambda \ln(1-r)$

Aangezien r en 1-r beiden homogeen verdeeld zijn op het interval (0,1) kan men deze transformatie ook vereenvoudigen tot $x=-1/\lambda \ln(r) = -\mu \ln(r)$.

3. Gewenste resultaten.

M.b.v. een simulatie wenst men een indruk te verkrijgen van de kansverdeling van één of meerdere variabelen. Hiertoe zal men i.h.a. berekenen en afdrukken:

3.1. Het gemiddelde $\bar{x} = \frac{\Sigma x_i}{n}$

en

3.2. De steekproefvariantie $S^2 = \frac{\Sigma(x_i-\bar{x})^2}{n-1} = \frac{\Sigma x_i^2 - \frac{(\Sigma x_i)^2}{n}}{n-1}$

(Zie [1] stat.comp.: pag. 2.)

Veelal zal men ook een indruk willen krijgen van de vorm van de kansverdeling door middel van een:

3.3. Histogram.

Bij een serie waarnemingen kan men een klasseindeling ontwerpen en door "turven" het aantal waarnemingen in iedere klasse noteren. Men verkrijgt een zgn. turfstaat. (zie [1] stat.comp.: pag. 28). Een histogram verkrijgt men hieruit door het aantal waarnemingen in iedere klasse te delen door n (het totaal aantal waarnemingen) en het percentage door kruisjes weer te geven.
Programmatisch kunnen we bij het maken van een histogram twee gevallen onderscheiden.

3.3.1. Het aantal klassen en de klassebreedte van het histogram is van te voren bekend.

Dit is bij de meeste eenvoudige simulaties wel het geval. Stel we moeten een histogram maken met n klassen met breedte a. Hiertoe declareren we een integer array bijv. hist [0: n+1], rekening houdend met de restklassen aan beide zijden.
Tijdens de simulatie wordt bij een waarneming in de passende klasse "een turfje" geplaatst.
Dit kan gebeuren d.m.v. de entierfunctie.
Alleen dit geval wordt in deze syllabus behandeld.

3.3.2 Van de klassebreedte is van te voren niets te zeggen.

De eenvoudigste weg is dan de waarnemingen in een achtergrondgeheugen op te slaan en ze m.b.v. een vervolg programma weer op te halen. Door dan de grootste en de kleinste waarde op te sporen kan de klassebreedte uit het aantal gewenste klassen bepaald worden.
Een andere manier is om in een (toch al meestal noodzakelijke) voorlooprun de grootste en kleinste waarde te bepalen.

4. Wachtrijen.

We kunnen de discrete systemen indelen in twee klassen:

4.1 er ontstaan géén wachtrijen.

Bestel en Voorraadsystemen vallen hier dikwijls onder.
De simulatieprogramma's voor systemen van deze klasse zijn zonder bijzondere moeilijkheden in een algemene taal te programmeren. Wij zullen dit soort simulaties hier niet nader behandelen.

4.2 er ontstaan wel wachtrijen.

"Job shop" en "loket" systemen zijn hiervan typische voorbeelden.
De implementatie van een "wachtrij-notitie" is een minder eenvoudige zaak. In het volgende zullen we zien dat een passend procedure pakket onmisbaar is. In wezen verzorgen deze procedures eenvoudige manipulaties met lijststructuren.
In het eerste , zeer eenvoudige, voorbeeld (zie 5) zijn deze gereedschappen nog niet noodzakelijk.

5. Simulatie van een benzinepompstation.

5.1 Probleemstelling.

Het pompstation bestaat uit een emplacement waarop zich 1 pomp bevindt. Auto's arriveren op het emplacement, de interaankomsttijd is negatief exponentieel verdeeld met $\mu = 4$. De helptijd bij een pomp is negatief exponentieel verdeeld met gemiddelde 3.
Indien er zich een wachtrij voor de pompen bevindt met lengte p dan zal een aankomende auto het emplacement verlaten zonder te tanken.
Gevraagd: gemiddelde en variantie van wachttijden p=3 en p=4 berekend bij 1000 behandelde auto's.
Met aanloop-verschijnselen behoeft geen rekening te worden gehouden.

5.2 Eventsoorten.

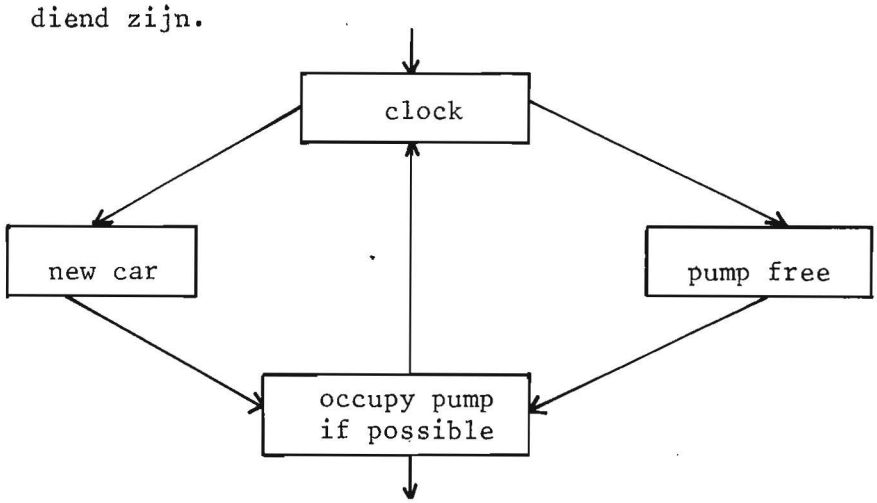
Indien we op een willekeurig moment het station waarnemen, dan constateren we bv. dat de pomp bezet is en er een aantal wachtenden in de wachtrij staan. Deze toestand blijft voortbestaan totdat er een van de 2 volgende events plaatsvindt:

1. de pomp komt vrij.
2. er komt een auto het emplacement oprijden.

Omdat een auto, die in de wachtrij staat, deze niet zonder te tanken zal verlaten, zijn dit de enige soorten events die de toestand van het systeem kunnen veranderen.

5.3 Het Stroomschema.

De klokroutine dient te bepalen van welke soort het eerstvolgende event is. Op grond hiervan gaat het programma verder met "new car" of "pumpfree". Hierna dient zo mogelijk de pomp bezet te worden (zie figuur).
Uiteraard zal dit voorafgegaan worden door een declaratie en initialisatie stuk en gevolgd door het afdrukken van de output. Tevens dienen we ervoor te zorgen, dat het programma zo vaak herhaald wordt totdat 1000 auto's bediend zijn.



Wij zullen nu de in bovenstaand schema genoemde onderdelen nader toelichten. Voor betekenis van gebruikte namen van variabelen zie 5.4.1.

5.3.1 clock:

now: = min (et[i]); evnumb:= rangnummer behorend bij dit minimum.
i=1,2

De gegevens van dit eerstkomend event staan genoteerd in "now" en "evnumb". "Now" is de eventtijd, dwz. het is de systeemtijd van het reële systeem. Evnumb geeft de soort event aan: d.w.z.

evnumb = 2 komt overeen met aankomst nieuwe auto

evnumb = 1 komt overeen met pomp 1 komt klaar

Alle gegevens van dit event zijn genoteerd, het event zelf kan (in "et") "vernietigd" worden. Sterker nog: het moet worden vernietigd, aangezien anders de mogelijkheid bestaat, dat men ad infinitum het zelfde event blijft afwerken. Ook in twijfelgevallen is het altijd aan te raden om het zekere voor het onzekere te nemen. Op grond van de waarde van evnumb wordt het programma onderdeel new car of pumpfree uitgevoerd.

5.3.2 New car

De aankomsttijd van deze nieuwe auto staat in "now". We onderzoeken of deze auto door zal rijden of niet. Indien het geen doorrijder is wordt length q met 1 verhoogd en de aankomsttijd wordt in de rij genoteerd. We dienen voor later de aankomsttijd van de daaropvolgende auto te genereren en te plaatsen in et [2] (zie 1.8.2).

5.3.3 pump free

De Boolean variabele "pump free" wordt aangepast.

Gegevens voor statistieken worden bijgehouden.

Het aantal afgewerkte auto's wordt met 1 verhoogd.

5.3.4 occupy pomp

Indien de pomp vrij is (pump free = true) en tenminste één auto in de wachtrij staat (length q > 0) moet de pomp bezet worden.

Deze situatie kan ontstaan door de aankomst van een auto bij "leeg-loop" of door het vrijkomen van de pomp.

De aankomsttijd van de eerste in de wachtrij wordt genoteerd in "pump". De lengte van de rij wordt met 1 verlaagd. De rij wordt opgeschoven.

Het moment waarop de auto aan de pomp afgetankt is zal een toekomstig event zijn en wordt genoteerd in et [1] .

5.3.5 initial

De variabelen dienen zodanig geinitialiseerd te worden, dat de volgende situatie wordt weergegeven:

1. de pomp is vrij.
2. op tijd nul komt een auto aan.

5.3.6 output

Deze spreekt voor zichzelf.

5.4 De correctheid van het programma.

Door de opsplitsing van het programma in kleine onderdelen en het minimaliseren van de goto-statements is het controleren van de correctheid van het programma eenvoudig. Eigenlijk geeft de toelichting (5.3.1. t/m 5.3.6) wel voldoende vertrouwen.

5.5 Namen en hun betekenissen.

mnemonic	betekenis
i	hulpteller
readycar	aantal afgewerkte auto's
totcar	totaal aantal te verwerken auto's
evnumb	nummer van de kleinste eventtijd
lengthq	lengte van de wachtrij
pumpfree	true als pomp vrij is, anders false
dummy	hulpvariabele
now	systeemtijd
mu1	gemiddelde inter-aankomsttijd
mu2	gemiddelde helptijd
tot	wachttijd
sumtot	som van wachttijden
sumsqwt	som van kwadraten van wachttijden
inf	een zeer groot getal
et	event tijden
pump	moment van aankomst van klant in bediening bij de pomp
row	aankomsttijden van auto's
p	lengte emplacement

5.6 Het Algol-programma voor het benzinepompprobleem.

```
0  Lalgol 08163764 benzinepomp
1  begin
2  integer i,readycar,totcar,evnumb,lengthq,dummy,p;
3  real now,mu1,mu2,t,sumwt,sumsqwt,inf,frand;
4  boolean pumpfree;
5  real array et[1:2], row[1:4];
6  library SETRANDOM,RANDOM;

7  initial :
8  SETRANDOM(read);
9  totcar:=read; mu1:=read; mu2:=read; p:= read;
10 inf:=10600; sumwt:=sumsqwt:=0; lengthq:=readycar:=0;
11 et[1] :=inf; et[2]:=0; pumpfree:=true;

12 program:
13       for dummy:=0 while readycar<totcar do
14       begin
15 clock:      now:=inf;
16             for i:=1,2 do if et[i] < now then
17             begin evnumb :=i ; now:=et[i] end ;
18             et[evnumb]:= inf ;

19             if evnumb=2
20             then
21 new car:         begin if lengthq<p
22                        then begin lengthq:=lengthq + 1 ;
23                                   row[lengthq]:= now
24                             end;
25                        et[2]:= now - mu1×ln(RANDOM)
26                  end
27             else
28 pump is free :   begin pumpfree:=true;
29                        readycar:= readycar + 1
30                  end;

31             if pumpfree ∧ lengthq > 0
32             then
33 occupy pump :    begin pumpfree:=false; lengthq:=lengthq-1;
34                        t:=now - row[1];
35                        sumwt:=sumwt+t; sumsqwt:= sumsqwt + t×t;
36 push down :             for i:=1 step 1 until lengthq do
37                        row[i]:= row[i+1];
38                        et[1]:= now - mu2×ln(RANDOM)
39                  end
40       end;

41 output: NLCR;PRINTTEXT(<p=>);PRINT(p);
42         NLCR;PRINTTEXT(<mean waiting-time =>); PRINT( sumwt/totcar);
43         NLCR;PRINTTEXT(<variance of this  =>);
44              PRINT((sumsqwt-sumwt×sumwt/totcar)/(totcar-1))
45 end;
46 progend
```

6. Opgaven.

Varianten op het benzinepompprobleem zouden kunnen zijn:

6.1 Tabellen.

6.1.1 Algemene gegevens:

opgave	inter aankomst tijd	help tijd	aantal pompen	queue discipline	emplacements grootte	door rijden
6.2	2	3	2	FIFO	10	indien rijlengte = p
6.3	2	3	2	FIFO	10	indien rijlengte = p
6.4	4	3	1	FIFO	4	zie tabel 6.1.2.1
6.5	2	3	2	FIFO	10	zie tabel 6.1.2.2
6.6	2	3	2	FIFO	10	zie tabel 6.1.2.2
6.7	4	3	1	kortste helptijd	4	indien rijlengte = p

6.1.2 doorrijdkansen:

Indien er zich een wachtrij voor de pompen bevindt met lengte r, dan zal een aankomende auto met een kans P(r) het emplacement verlaten zonder te tanken. In het andere geval voegt hij zich in de wachtrij.

6.1.2.1 behorend bij 6.4

r	P(r)
0	0
1	0.1
2	0.4
3	0.6
4	1.0

6.1.2.2 behorend bij 6.5 en 6.6

r	P(r)
0	0
1	0
2	0
3	0.1
4	0.1
5	0.3
6	0.5
7	0.5
8	0.7
9	0.9
10	1.0

6.2 Indien beide pompen vrij zijn wordt door de automobilisten willekeurig een pomp gekozen. Er wordt niet gelet op de bezettingsgraad van de afzonderlijke pompen.
gevraagd: de gemiddelde wachttijd

6.3 Indien beide pompen vrij zijn wordt altijd eerst pomp 1 gekozen.
gevraagd: de gemiddelde wachttijd en de bezettingsgraad van beide pompen afzonderlijk.

6.4 Er moet rekening gehouden worden met doorrijdkansen.
gevraagd: de gemiddelde wachttijd.

6.5 Indien beide pompen vrij zijn wordt door de automobilisten willekeurig een pomp gekozen. Er wordt niet gelet op de bezettingsgraad van de afzonderlijke pompen.
Er moet rekening gehouden worden met doorrijdkansen.

6.6 Indien beide pompen vrij zijn wordt altijd eerst pomp 1 gekozen. Er moet rekening gehouden worden met doorrijdkansen.
gevraagd: de gemiddelde wachttijd en de bezettingsgraad van beide pompen afzonderlijk.

6.7 Dit probleem is op de queue discipline na identiek aan het originele probleem onder 5.1
gevraagd: de gemiddelde wachttijd.

7. Kritische beschouwing van de programmatische aanpak van het benzinepompprobleem.
In het voorbeeld hebben we gezien dat de organisatie van de wachtrij tamelijk eenvoudig was. Dit voorbeeld is echter het simpelst denkbare probleem.
In het volgende zullen we aantonen hoe beperkt de programmatische aanpak is en hoe snel men bij iets meer gecompliceerde gevallen vastloopt op programmeerbaarheid, ruimte gebruik en rekentijd.

7.1 Beperktsheid, matige efficientie
Bij de aanpak is essentieel gebruik gemaakt van de eenvoud van het voorbeeld.

7.1.1 de rij heeft een van te voren bekende maximale lengte.
Het is daardoor mogelijk een array van passende lengte te declareren. Indien de maximale rijlengte van te voren niet bekend is, moet men een zeer groot array declareren. Zo groot, dat men praktisch zeker weet dat overschrijding van de arraygrenzen niet plaatsvindt. Dit laatste noemt men majoreren.
Wanneer er meerdere wachtrijen zijn, dient men meerdere gemajoreerde 1-dimensionale array's of één gemajoreerd 2-dimensionaal array te declareren. Deze werkwijze legt een nodeloos groot beslag op de beschikbare geheugenruimte in de computer.

7.1.2. de queuediscipline is FIFO.
Het doorschuiven (zie 5.5, achterlabel "push down") is eenvoudig programmeerbaar. Bij opgave 6.7 blijkt dat een andere queuediscipline tot programmatische moeilijkheden leidt.
De efficientie van het doorschuiven is echter ook bij FIFO omgekeerd evenredig met de lengte van de wachtrij, terwijl toch de handeling zo simpel is nl.: "pak de voorste".

7.1.3 de component-notitie bestaat slechts uit één attribuut-notitie (de aankomsttijd).
Naarmate de component-notitie uit méér attribuut-notities bestaat, wordt het doorschuiven steeds inefficienter. Bij orders in een job-shop zal de component-

notitie bv. bestaan uit de routing van de orders.

7.1.4 in de lijst van events ("et") kan op ieder moment van iedere eventsoort slechts één eventnotice aanwezig zijn.
Bij opgave 6.2 kunnen we volstaan met twee eventsoorten: aankomst van een auto, het gereed komen van een pomp.
Van de laatste eventsoort kunnen er echter op een bepaald moment twee notitie's tegelijk aanwezig zijn. We dienen dan ook het array "et" te declareren als et [1:3] . Bij het bepalen van de eventsoort dient de statement:

```
evnumb:= i;
```

vervangen te worden door:

```
evnumb:= if i=1 then 1 else 2;
```

In gecompliceerdere gevallen ontstaan hierdoor zeer ondoorzichtige programma's.
De strategie waarbij de index van het array de eventsoort bepaald is verwerpelijk.
We zouden in de event-notitie niet alleen het eventtijdstip maar ook de eventsoort willen vermelden. Later zullen we zien dat het soms gewenst is nog meer parameters in de event-notitie op te nemen.

7.2 Pogingen tot efficientie-verbetering.
Terwille van de efficientie kan het doorschuiven (bij queuediscipline FIFO) o.a. op de volgende manieren vermeden worden.

7.2.1 De "lineaire" truc

We declareren het array row [1:k] met "k" zeer groot.

De wijze "wfirst" geeft de index in "row" aan van de notitie van het aankomsttijdstip van de voorste auto. De aankomsttijdstippen van de overige auto's staan hier consecutief achter.
De wijzer "wlast" wijst naar de notitie van het aankomsttijdstip van de laatst aangekomen auto.
Bij de label "occupy pump" worden statements 35, 36 en 37 vervangen door:

```
sumwt:= sumwt + now  - row wfirst ;
wfirst:= wfirst + 1;
if wfirst = wlast then wfirst:= wlast:= 0;
```

De laatste statement dient ervoor om met een waarde van k te kunnen volstaan die aanzienlijk kleiner is dan het totaal aantal gesimuleerde auto's. Bij leegloop worden de wijzers nl. weer op nul gezet. Vooral bij een lage bezettingsgraad

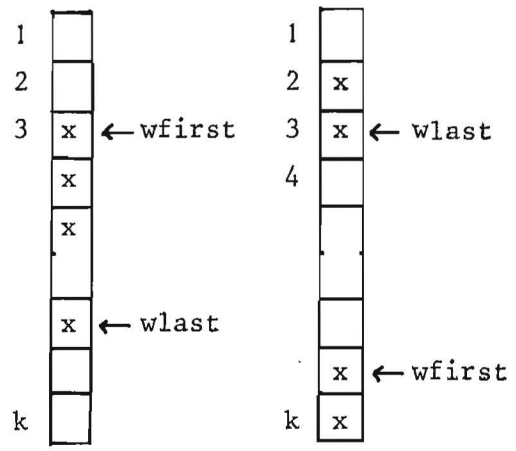
zal men met een vrij kleine waarde van k kunnen volstaan. Bij de label "new car" wordt statement 23 vervangen door: wlast:= wlast + 1;

row [wlast] := now

7.2.2 De "cyclische" truc

We declareren het array row [1:k] met "k" gemajoreerd op de maximale rijlengte. De wijzer "wfirst" geeft de index in "row" aan van de notitie van het aankomsttijdstip van de voorste auto. De wijzer "wlast" wijst naar de notitie van het aankomsttijdstip van de laatst aangekomen auto. Indien k-wfirst < lengthq dan staan er een aantal (wfirst + lengthq - k) op de plaatsen 1 t/m wlast.

Schematisch:



Van de statements in 7.2.1 worden de laatste twee onder de label "occupy pump" vervangen door:

wfirst:= wfirst + k * k + 1;

Van de statements in 7.2.1. wordt de eerste onder de label "new car" vervangen door:

wlast:= wlast - wlast + k * k + 1;

De programmeur dient in zijn programma nog te testen of "wlast" niet "wfirst" inhaalt, m.a.w. of de rijlengte niet toch groter is dan k. Bij 7.2.1 zorgt het Algol-systeem door "bound-check" voor de bewaking.

8. Liststructuren.

8.1 Algemene liststructuren.

We denken bij rijen aan lijsten. Een lijst bestaat uit elementen, ieder element heeft directe voorgangers en directe opvolgers.

Bij deze lijsten moet het mogelijk zijn op eenvoudige wijze elementen te verwijderen of toe te voegen.

De elementen van een lijst kunnen bestaan uit een aantal attribuut-notities. Deze attribuut-notities kunnen van type verschillen: boolean, integer, real, array etc.
Bij een groot aantal praktische problemen is het gewenst, dat er in het programma op eenvoudige wijze gemanipuleerd kan worden met lijststructuren. Bv. bij wachtrijproblemen, netwerkproblemen, bestandsorganisatie, bevolkingsregistratie etc.
Het efficient implementeren van lijststructuren in hun volle omvang is gecompliceerd. De behoefte aan het manipuleren met lijststructuren is pas de laatste jaren tot uiting gekomen. Dit is de reden dat we in de oudere talen geen of gebrekkige en pas bij de recentere talen uitgebreide voorzieningen aantreffen. Anderzijds is dit een van de redenen waarom efficiente vertalers voor deze talen niet of nauwelijks aanwezig zijn. Het is zelfs de vraag of efficiente vertaling op de huidige computers mogelijk is.
Overzicht:

taal	voorzieningen voor lijststructuren
ALGOL 60	geen
FORTRAN	geen
COBOL	gebrekkig
ALGOL 68	uitgebreid
PL I	redelijk
SIMULA	uitgebreid

In ons geval kunnen we volstaan met een eenvoudige lijststructuur.

8.2 Enkelvoudige lijststructuur.
Bij een wachtrij heeft ieder element ten hoogste één opvolger en ten hoogste één voorganger. We zullen ons tot dit eenvoudige geval beperken. De structuur van een netwerk bv. kan veel gecompliceerder zijn.
We noemen de bijbehorende lijststructuur een enkelvoudige lijststructuur of ketting (chain).
De elementen van deze ketting noemen we schakels (links). Deze schakels bestaan uit een vast aantal systeemparameters (wijzers naar voorganger en opvolger etc.) en een aantal attribuut-notities. We beperken ons tot het geval waarbij per probleem het maximum aantal attribuut-notities tevoren bekend is, m.a.w. tot schakels van vaste lengte. Deze lengte = het aantal systeem parameters + het

maximum aantal attribuut-noties. In gevallen waarbij het aantal attribuut-notities zeer sterk uiteenloopt legt dit een nodeloos beslag op de beschikbare geheugenruimte. Echter in verreweg de meeste gevallen is het geheugengebruik zeer redelijk.
We gaan er van uit dat alle attribuut-notities van hetzelfde type zijn.

9. Schakels, kettingen, de kettingkast.

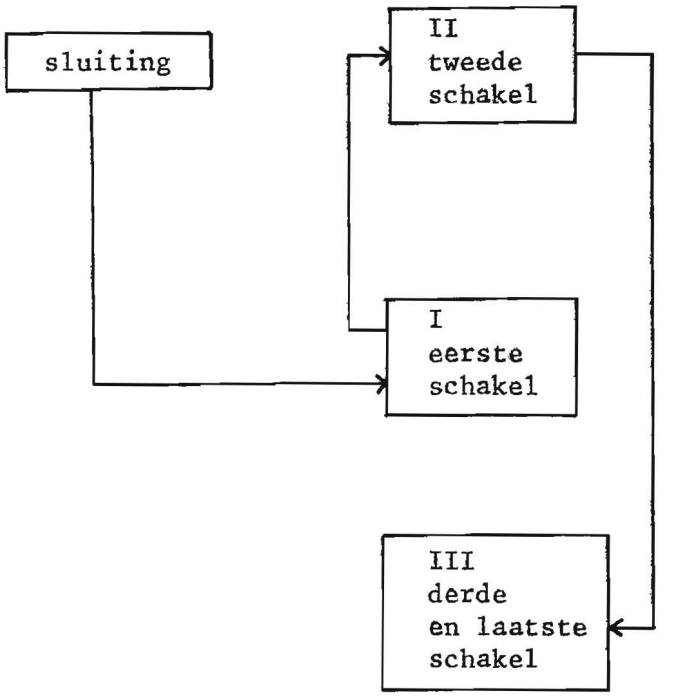
We geven iedere wachtrij weer als een ketting. Dit kunnen we ook doen voor de lijst van toekomstige events, dit is immers ook een wachtrij.
Het stelsel van kettingen wordt ondergebracht in de "kettingkast" (chain case) een groot 1-dimensionaal array "cc".
Een ketting bestaat uit schakels.
Een schakel bestaat uit een aantal consecutieve elementen van "cc".
Twee schakels die elkaars voorganger en opvolger zijn in een ketting, behoeven in het array "cc" niet consecutief te staan. Het opvolger resp. voorganger zijn wordt bepaald door de waarde van wijzers in de schakel.
De plaats van een schakel in "cc" bepaalt in het geheel niet het rangnummer in de rij (ketting). Zo kan het best voorkomen dat de eerste schakel van de eerste ketting achteraan in "cc" staat.

Bij iedere ketting behoort een "sluiting". Deze sluiting:

1. is een "fictieve" schakel, die niet tot de ketting behoort.
2. representeert niet een component of een event en heeft dus ook geen attribuut- of eventnotities.
3. bestaat slechts uit de volgende drie systeemparameters
   - de wijzer naar de eerste schakel van de ketting
   - de wijzer naar de laatste schakel van de ketting
   - het aantal schakels van de ketting
4. staat op een vaste plaats in "cc".

Via de sluiting van een ketting heeft men toegang tot de eerste en de laatste schakel en via elk van deze tot de overige schakels.

schematisch:



Terwille van de overzichtelijkheid van het schema zijn slechts de pijlen naar opvolgers getekend. In het systeem heeft elke schakel een wijzer naar zijn voorganger en een wijzer naar zijn opvolger. Het nut van deze dubbele verwijzing wordt in 9.1 besproken.

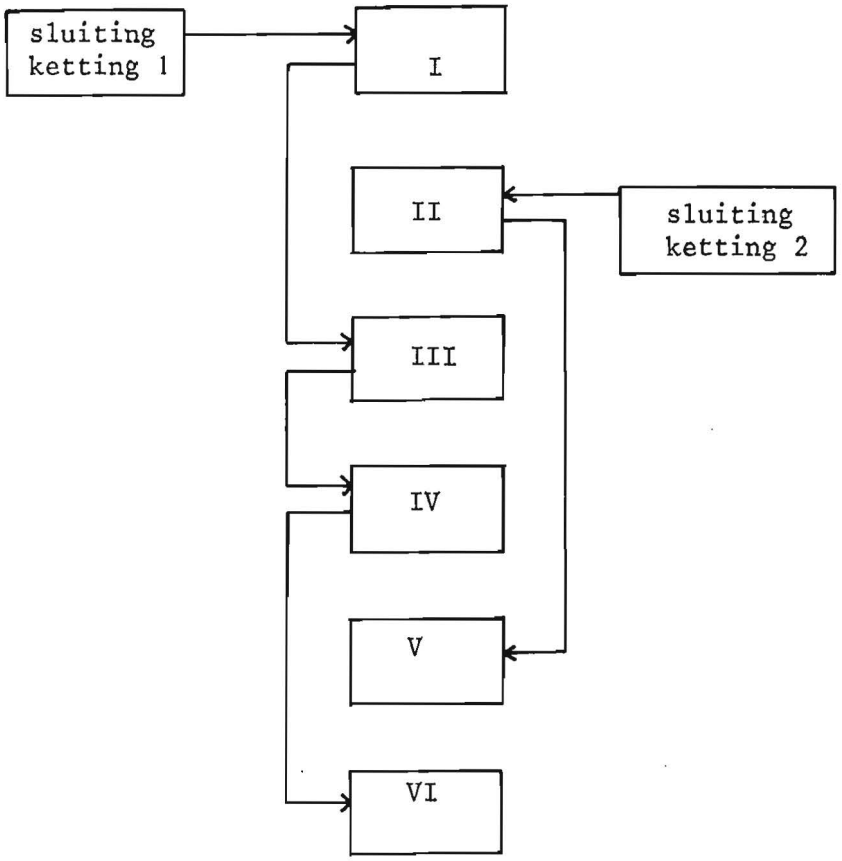
9.1 Het manipuleren met schakels in kettingen.

Tijdens een simulatie zullen we dikwijls een schakel van de ene ketting naar een andere ketting willen overbrengen. Dit immers geeft de overgang van de klant van de ene wachtrij naar de andere weer. Hiervoor is slechts de verandering van enkele wijzers nodig, de informatie zelf blijft op dezelfde plaats staan. De hoeveelheid werk is onafhankelijk van de grootte van het blok te verhuizen informatie.

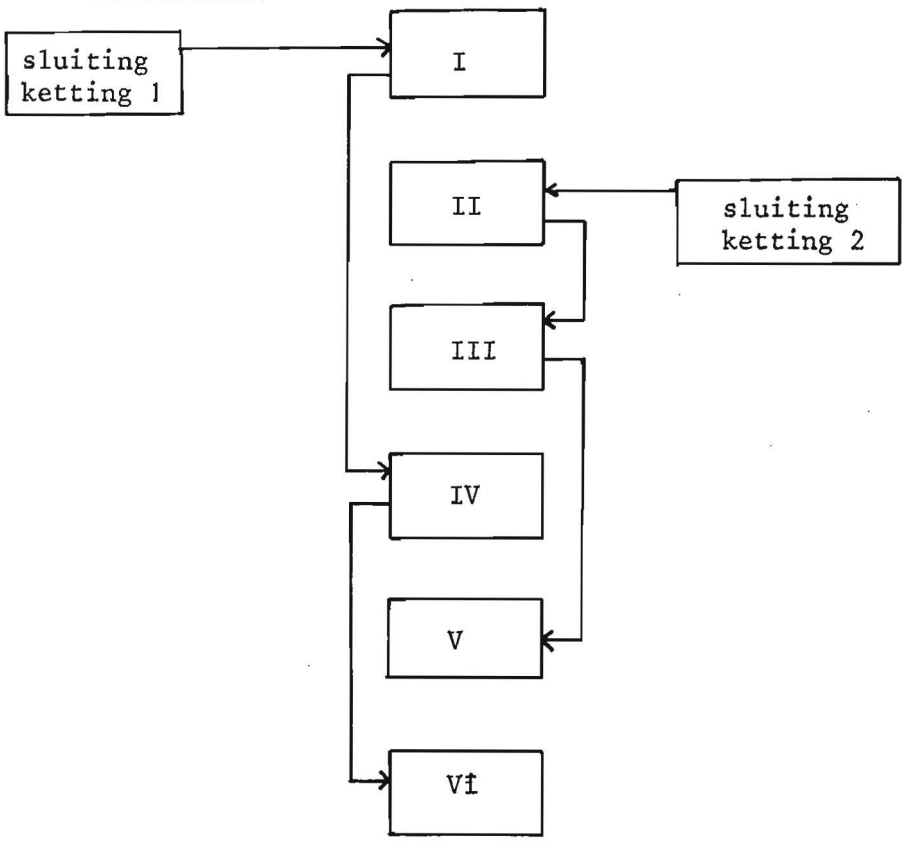
schematisch:

Stel dat de tweede schakel (III) van ketting 1 moet komen tussen de eerste schakel (II) en de tweede schakel (V) van ketting 2.

Oude situatie:



Nieuwe situatie



In de schakels staan zowel voorganger als opvolger vermeldt. Dit kost extra geheugenruimte. Het heeft echter de volgende voordelen:

1. we kunnen de ketting van voor naar achteren of in omgekeerde richting doorlopen
2. stel dat we met een enkele verwijzing zouden werken bv. naar de opvolger. Indien we nu een willekeurige schakel uit de ketting willen verwijderendan dienen we de wijzer bij zijn voorganger te veranderen. Deze voorganger is echter niet direkt bereikbaar. We kunnen deze voorganger alleen vinden door de ketting vanaf het begin af te lopen. Bij dubbele verwijzing is dit geen probleem.

9.2 De vergaarbak.

Tijdens een simulatie moeten we,
- enerzijds beschikken over een "nieuwe" schakel (aankomst van een klant)
- anderzijds schakels kunnen "weggooien" (een klant verlaat het systeem).

In werkelijkheid kunnen schakels niet gecreëerd of weggegooid worden. We dienen te beschikken over een vergaarbak. Deze vergaarbak bestaat uit de "vrije ketting" + de "vrije ruimte".

De vrije ruimte bestaat uit een consecutief stuk van het array "cc" dat op dat moment niet in gebruik is en loopt vanaf de kleinst mogelijke index tot de bovengrens van "cc".
De vrije ketting bestaat uit niet in gebruik zijnde schakels in "cc" die staan tussen wel in gebruik zijnde schakels. M.a.w. de schakels die niet in de vrije ruimte zijn onder te brengen.
De organisatie van de vrije ketting kan eenvoudiger zijn dan die van de andere kettingen. Immers de volgorde van de schakels in de vrije ketting doet niet ter zake. We kunnen volstaan met een enkele verwijzing naar het beginadres van de volgende schakel.
Vrijgegeven schakels worden, indien ze niet aan de vrije ruimte toegevoegd kunnen worden, vooraan in de vrije ketting gezet.
Bij het creëren van een "nieuwe" schakel wordt indien de vrije ketting niet leeg is, de eerste schakel genomen anders een stuk van de vrije ruimte.

9.3 Het begrip "nill".

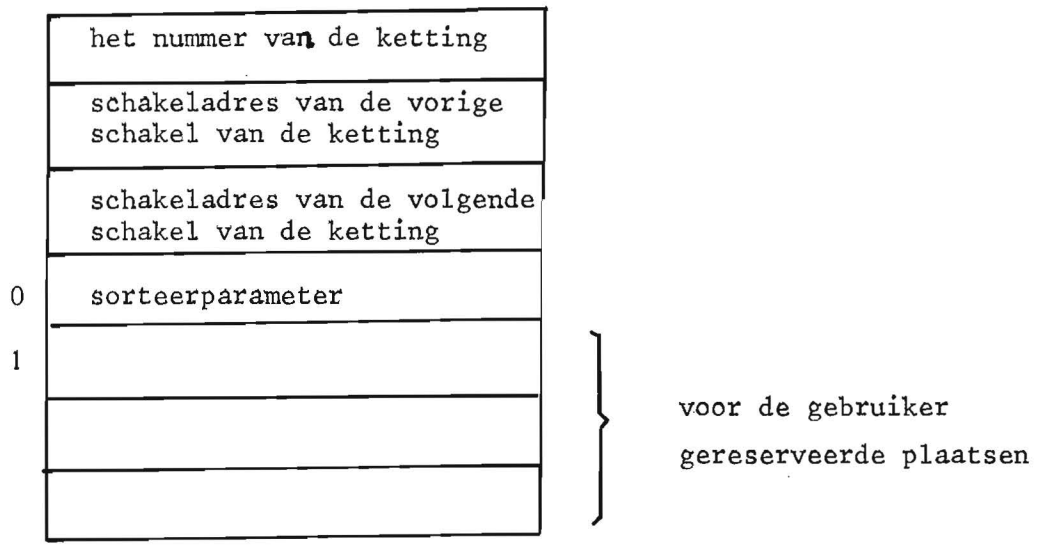
De eerste schakel van een ketting heeft géén voorganger, de laatste géén opvolger. We gaan een begrip introduceren voor "er is er geen". Dit begrip noemen we "nill" een samentrekking van nil link.

Een wijzer moet de waarde "nill" kunnen hebben. We representeren deze waarde door een getal dat als wijzer nooit voor kan komen (bv. een getal < 0). In het programma kunnen we testen of de wijzer de waarde "nill" heeft. Om vergissingen te voorkomen voeren we de variabele "nill" in, die aan het begin van het programma eenmalig gesteld wordt.
Indien ketting j leeg is hebben de beide wijzers in sluiting j de waarde "nill".

9.4 Indeling van de schakels.

Iedere schakel bestaat uit minstens 4 systeemparameters. De index van array "cc" die de vierde systeemparameter van een schakel aangeeft zullen we het schakeladres (linkaddress = la) van die schakel noemen.
De eerste systeemparameter geeft aan tot welke ketting de schakel behoort. De tweede systeemparameter geeft het schakeladres aan van de schakel die vooraf gaat aan de beschouwde schakel.
De derde systeemparameter geeft het schakeladres aan van de schakel die volgt op de beschouwde schakel.
De vierde systeemparameter is de sorteerparameter. Indien men met een andere queuediscipline dan FIFO wenst te werken kan men gebruik maken van deze sorteerparameter. Er is een procedure die op grond van de waarde van de sorteerparameter schakels in een rij sorteert (zie 10.1.7).
Het aantal overige elementen wordt door de gebruiker via de input bepaald. Deze elementen en de sorteerparameter zijn vrijelijk ter beschikking van de gebruiker.
De bewaking (9. 5) zorgt ervoor, dat de gebruiker de eerste drie systeemparameters niet kan bereiken.

schematisch:



9.5 De bewaking.

Door een programmeerfout zou de gebruiker op de plaats van een wijzer bv. een helptijd kunnen plaatsen. Het is mogelijk dat ondanks deze fout, het programma nog geruime tijd normaal doorrekend. Op het moment dat eventueel het Algol-systeem een fout aanwijzing geeft, is de oorzaak van de fout niet meer te achterhalen.

Er is een set procedures ontworpen waarmee dergelijke fouten gedetecteerd kunnen worden. Het programma van de gebruiker wordt bij een dergelijke fout onmiddellijk afgebroken. De procedures leveren een uitgebreide output voor foutdetectie.
De gebruiker behoeft zich het bestaan van de bewaking nauwelijks te realiseren.

9.6 "cc".

9.6.1 De indeling van "cc".

Aan het begin van het programma dient "cc" gedeclareerd te worden als real of integer array cc [-5: ubcc]; waarbij de integer ubcc zo groot mogelijk wordt genomen.
Het real of integer hangt af van de soort informatie die de gebruiker in de schakels wil opbergen. Bij onze voorbeelden gaan we uit van een real array cc.
Het array is als volgt ingedeeld:

cc[-5] = ubcc: Door gebruiker op te geven bovengrens van array cc
cc[-4] = ll : De voor de gebruiker te reserveren plaatsen per schakel
cc[-3] = p : Door gebruiker op te geven aantal wachtrijen
cc[-2] = q : Start index van de ketting van vrijgekomen schakels
cc[-1] = r : Begin index van de vrije ruimte in cc
cc[0] : la van achterste schakel in ketting 0 } nill als ketting
cc[1] : la van voorste schakel in ketting 0 } 0 leeg is
cc[2] : aantal schakels in ketting 0 (0 als ketting 0 leeg is)
cc[3] : }
cc[4] : } als 0, 1 en 2 maar nu voor ketting 1
cc[5] : }
⋮

```
cc[ 3 x p     ] ┐
cc[ 3 x p + 1 ] } als 0, 1 en 2 maar nu voor ketting (p-1)
cc[ 3 x p + 2 ] ┘
cc[ 3 x p + 3 ] ┐
                │
                } plaatsruimte voor schakels
                │
cc[ ubcc]       ┘
```

9.6.2 Startwaarden van "cc".

Na de declaratie van "cc" worden de kettingkast procedures gedeclareerd. Het hoofdprogramma begint met de aanroep van procedure initial die de in 9.6.1 beschreven elementen de juiste startwaarden geeft. Hierbij wordt

q = r = 3 x p;

cc [0] tot en met cc[3 x p - 1] worden zodanig opgeleverd dat alle kettingen leeg zijn.

Zolang geldt dat q = r is de vrije ketting leeg. Wordt er in dit geval een nieuwe schakel gevraagd, dan wordt de nodige ruimte hiervoor (11 + 4 plaatsen) van de vrije ruimte gehaald, waarna q zowel als r met (11 + 4) worden verhoogd.

10. Het procedurepakket "ASP".

Het procedurepakket bestaat uit:

- procedures voor het manipuleren met schakels en kettingen: de "kettingkast" procedures
- procedures die zorgen voor de bewaking: de "bewakings" procedures
- procedures voor het berekenen van gemiddelden en varianties en het maken van histogrammen: de "histogram" procedures.

Er bestaan twee versies van het pakket:

- de bewaakte versie.
  Deze versie wordt hier behandeld. Hierin zijn de bewakingsprocedures opgenomen. In de kettingkast procedures worden indien nodig de bewakingsprocedures aangeroepen
- de onbewaakte versie
  Hierin ontbreken de bewakingsprocedures en uiteraard ook de aanroepen daarvan in de kettingkastprocedures.

Voor de ongeoefende gebruiker is het raadzaam altijd de bewaakte versie te gebruiken. De geoefende gebruiker zal tijdens de testfase de bewaakte versie en tijdens de produktiefase de onbewaakte versie gebruiken. De onbewaakte versie is ongeveer twee keer zo snel als de bewaakte versie.

10.1 De kettingkast procedures.

10.1.1 procedure initial (lengte van "cc", aantal voor de gebruiker te reserveren plaatsen in de schakel, aantal kettingen)

Zorgt voor de initialisering van de kettingkast.

10.1.2 integer procedure fromf; (from free)

Haalt een schakel uit de vergaarbak en levert het schakeladres van deze schakel af. Deze schakel staat nu los van iedere ketting en is door de gebruiker verder te besturen m.b.v. het schakeladres.

10.1.3 procedure tof(la); (to free).

Plaatst de schakel met schakeladres la in de vergaarbak. Hierbij wordt verondersteld dat deze schakel los staat.

10.1.4 procedure fromch (la,j); (from chain)

Haalt de schakeladres la van ketting j.
De schakel staat nu los.

10.1.5 procedure to head (la,j); (to head of chain)

Plaatst de losstaande schakel met adres la vooraan ketting j.

10.1.6 procedure totail (la,j); (to tail of chain)

PLaatst de losstaande schakel met adres la achteraan ketting j.

10.1.7 procedure tosort (la,j); (to sort in chain)

Plaatst de losstaande schakel met adres la in ketting j volgens de sorteerparameter op la, zodanig dat deze sorteerparameters een niet dalende rij vormen.

10.1.8 integer procedure lafirst(j); (link address of first link of chain)
Levert het schakeladres op van de voorste schakel van ketting j. Indien ketting j leeg is levert een aanroep van lafirst de waarde nill op.

10.1.9 integer procedure lalast(j); (link address of last link of chain)
Levert het schakeladres op van de achterste schakel van ketting j. Indien ketting j leeg is levert een aanroep van lalast de waarde nill op.

10.1.10 integer procedure lasucc (la); (link address of successor)
Levert het schakeladres op van de schakel, die volgt op de schakel met schakeladres la. Is de schakel met schakeladres la de achterste van de betreffende ketting, dan levert een aanroep van lasucc de waarde nill op.

10.1.11 integer procedure lapred (la); (link address of predecessor)
Levert het schakeladres op van de schakel die voorafgaat aan de schakel met adres la. Is de schakel met schakeladres la de voorste van de betreffende ketting, dan levert een aanroep van lapred de waarde nill op.

10.1.12 procedure datato (la, i, data); (data to link)
Plaatst de waarde van data in de voor de gebruiker gereserveerde i-de plaats van de schakel met schakeladres la.

10.1.13 real procedure datafrom (la, i); (data from link)
Levert de informatie die staat in de i^{de} gereserveerde plaats in de schakel met schakeladres la.

10.1.14 integer procedure length (j); (length of queue)
Levert het aantal schakels van ketting j af.

10.2 De bewakingsprocedures.

10.2.1 boolean procedure testla (la); (test on link address).
Onderzocht wordt of la een schakeladres kan zijn.
Zo nee, dan volgt de foutmelding "incorrect link address", waarna de waarde van la wordt afgedrukt.

10.2.2 boolean procedure testq (j); (test on queue-number)
Onderzocht wordt of j refereert naar een bestaande ketting.
Zo nee, dan volgt de foutmelding "incorrect number of queue", waarna de waarde van j wordt afgedrukt.

10.2.3 boolean procedure testc (i); (test on cell-number)
Onderzocht wordt of $0 \leq i \leq$ aantal gereserveerde plaatsen.
Zo nee, dan volgt de foutmelding "incorrect number of cell", waarna de waarde van i wordt afgedrukt.

10.2.4 boolean procedure testlaq (la,j); (test on link address and queue-number)
Onderzocht wordt of la een schakeladres kan zijn, of j het nummer van een bestaande ketting is en of de schakel met adres la behoort tot ketting j.
Zo nee, dan volgt de foutmelding "link address does not correspond with given number of queue", waarna j wordt afgedrukt en het nummer van de ketting waartoe la behoort.

10.2.5 procedure errordata;
Indien één der boolean testprocedures de waarde true aflevert, wordt errordata aangeroepen.
Errordata drukt de vaste gegevens van array cc af, voor elke ketting het aantal schakels en controleert of dit aantal juist is.

10.2.6 procedure jumpout;
In deze procedure wordt met opzet een aanroep van een array element gedaan dat niet bestaat. Gevolg is dat het T.H.E. Algol-systeem voor de EL-X8 een foutmelding geeft. Deze foutmelding kan van groot nut zijn bij het opsporen van fouten in het hoofdprogramma.

10.3 Het maken van histogrammen.
We gaan er hiervan uit dat de gebruiker van te voren opgeeft in welk projekt de waarnemingen zullen vallen en wat een zinvolle klassebreedte is. In verreweg de meeste praktische gevallen is dit zeer goed mogelijk.
ASP beschikt ook over een pakket waarbij de waarnemingen tevens op een magneetband geschreven worden. Indien nu het histogram niet naar wens is kan men vanaf deze magneetband later een nieuw histogram maken zonder dat de simulatie opnieuw uitgevoerd behoeft te worden. Dit laatste echter zal hier niet verder worden besproken.

10.3.1 De histogram arrays

De gebruiker dient via de input op te geven hoeveel histogrammen hij wenst te maken. Dit aantal wordt aangetekend in de integer variabele "nhist".
In een binnenblok volgt dan de declaratie van een 2-dimensionaal array:
real array histdata[1 : nhist, 1:4] ;

10.3.1.1 Het array "histdata"

Dit array dient via de input gevuld te worden met de volgende karakteristieke gegevens:

histdata [i,1] = akl; het aantal klassen van histogram i.
histdata [i,2] = low; de ondergrens van de eerste klasse van histogram i
histdata [i,3] = klbr; de klassebreedte der klassen van histogram i
histdata [i,4] = state; de wijze waarop de in te voeren waarden geturfd moeten worden.
state = 0; indien de te turven waarden integers zijn
state = 1; indien de te turven waarden reals zijn

Op grond van de gegevens in "histdata" kan nu in een binnen blok een array van passende grootte worden gedeclareerd, waarin de histogrammen zelf kunnen worden opgebouwd (zie 10.3.1.2).

10.3.1.2 Het array "hist".

Het aantal plaatsen dat in het array "hist" voor histogram i wordt gereserveerd = histdata [i,1] + 8 .

nl.:

1 : entries (het aantal keren dat een waarde is ingebracht) } tellingen
2 : minimum entry
3 : maximum entry
4 : som van entries
5 : kwadraatsom van entries
6 : frekwenties van entries < low } turfstaat
7 : frekwentie van entries in klasse 1
8 : frekwentie van entries in klasse 2
9 :
6+akl : frekwentie van entries in klasse (akl)
7+akl : frekwentie van entries > bovengrens klasse (akl)
8+akl : zero entries: het aantal keren dat een waarneming ligt tussen -10^{-10} en 10^{-10}

We zien hieruit dat het aantal plaatsen dat in het array "hist" moet worden gereserveerd voor de histogrammen = $\sum_{i}^{nhist}$ (histdata [i,1] + 8).
Deze waarde wordt in het buitenblok van dit blok toegekend aan de variabele ubhist op de plaatsen hist[-1] t/m hist [-nhist] . Staan de wijzers naar de plaatsen waar de diverse histogrammen beginnen, dwz. op hist [-j] staat de wijzer naar de begin plaats van het j^e histogram.
De procedure "histset" verzorgd de initialisatie van het array "hist",

10.3.2 De histogram procedures

10.3.2.1 procedure histdef;
Bij een aanroep van histdef worden voor alle histogrammen de door de gebruiker opgegeven karakteristieke waarden ingelezen en in het array histdata opgeborgen. Tevens wordt"ubhist" bepaald.

10.3.2.2 procedure histset(nhist);
Een aanroep van histset initialiseert het array hist.

10.3.2.3 procedure histput (j, x, num x);
Een aanroep van histput werkt histogram j bij met "numx" waarnemingen, van de waarde "x".

10.3.2.4 procedure histmv (j, m, v);
Levert het gemiddelde van histogram j af in m, de variantie in v

10.3.2.5 procedure histprint (string, j);
Een aanroep van "histprint" levert de volgende output:
- de in de aanroep van histprint meegegeven "string", en het nummer van het histogram
- het gemiddelde
- de variantie
- het minimum
- het maximum
- het aantal entries
- het aantal "nul" entries (zie 10.3.1.2)
- het histogram
- het cumulatieve histogram

Overgang op een nieuwe pagina of een nieuwe regel wordt door de procedure verzorgd.

10.4 Algol-tekst van de kettingkast procedures.

```
procedure initial(l, n, p);
value l, n, p; integer l, n, p;
begin
      integer i;
      cc[-5]:= l;  cc[-4]:= n;  cc[-3]:= p;
      cc[-2]:= cc[-1]:= 3×p;
      for i:=0 step 3 until cc[-2]-1 do
      begin cc[i]:= cc[i+1]:= nill;
            cc[i+2]:= 0
      end;
end initial of cc;


integer procedure fromf;
begin
      integer i, p;
      if cc[-2] ≠ cc[-1]
      then begin p:= cc[-2];  cc[-2]:= cc[p] end
      else begin p:= cc[-1];  cc[-2]:= cc[-1]:= p+cc[-4]+4;
                 if cc[-1]>cc[-5]+1 then begin NLCR; PRINTTEXT(‹array cc too small›);
                                               errordata;  jumpout
                                         end
           end;
      cc[p]:= nill;
      fromf:= p+3;
end from free;


procedure tof(la);
value la; integer la;
begin
      integer p;
      if testla(la) then begin errordata;  jumpout end;
      if cc[la-3]≠nill then begin NLCR; PRINTTEXT(‹incorrect call of tof:  link is member of chain›); FIXT(3, 0, cc[la-3]);
                                  errordata; jumpout
                            end;
      p:= la-3;  cc[p]:= cc[-2];  cc[-2]:= p
end to free;
```

```
procedure fromch(la, j);
value la, j; integer la, j;
begin
      integer p, pred, succ, lengthj;
      if testlaq(la, j) then begin errordata;  jumpout end;
      p:= 3×j;  lengthj:= cc[p+2];
      if lengthj < 1 then begin NLCR; PRINTTEXT(‹link from empty chain›); PRINT(lengthj);
                                errordata;  jumpout
                          end;
      pred:= cc[la-2];  succ:= cc[la-1];
      if pred=nill then cc[p+1]:= succ else cc[pred-1]:= succ;
      if succ=nill then cc[p]:= pred else cc[succ-2]:= pred;
      cc[p+2]:= cc[p+2]-1;
      cc[la-3]:= nill;
end from chain;



procedure tohead(la, j);
value la, j; integer la, j;
begin
      integer p, lafirst; boolean error;
      error:= testla(la) ∨ testq(j);
      if cc[la-3]≠nill then begin NLCR; PRINTTEXT(‹incorrect call of tohead:  link is member of chain›); FIXT(3, 0, cc[la-3]);
                                error:= true
                          end;
      if error then begin errordata;  jumpout end;
      p:= 3×j+1;  lafirst:= cc[p];
      if lafirst=nill
      then begin cc[p]:= cc[p-1]:= la;  cc[la-1]:= cc[la-2]:= nill end
      else begin cc[p]:= cc[lafirst-2]:= la;  cc[la-1]:= lafirst;  cc[la-2]:= nill end;
      cc[la-3]:= j;
      cc[p+1]:= cc[p+1]+1
end to head of chain;
```

```
procedure totail(la, j);
value la, j; integer la, j;
begin
      integer p, lalast;  boolean error;
      error:= testla(la) ∨ testq(j);
      if cc[la-3]≠nill then begin NLCR; PRINTTEXT(‹incorrect call of totail:  link is member of chain›); FIXT(3, 0, cc[la-3]);
                                  error:= true
                           end;
      if error then begin errordata;  jumpout end;
      p:= 3×j;  lalast:= cc[p];
      if lalast=nill
      then begin cc[p]:= cc[p+1]:= la;  cc[la-1]:= cc[la-2]:= nill end
      else begin cc[p]:= cc[lalast-1]:= la;  cc[la-2]:= lalast;  cc[la-1]:= nill end;
      cc[la-3]:= j;
      cc[p+2]:= cc[p+2]+1
end to tail of chain;



procedure tosort(la, j);
value la, j;  integer la, j;
begin
      integer p, pred, succ, lalow, lahigh;  real sp, low, high;  boolean error;
      error:= testla(la) ∨ testq(j);
      if cc[la-3]≠nill then begin NLCR; PRINTTEXT(‹incorrect call of tosort:  link is member of chain›); FIXT(3, 0, cc[la-3]);
                                  error:= true
                           end;
      if error then begin errordata;  jumpout end;
      p:= 3×j+1;  lalow:= cc[p];  lahigh:= cc[p-1];
      sp:= cc[la];
      if lalow=nill
      then
empty:      begin cc[p]:= cc[p-1]:= la;
                  cc[la-1]:= cc[la-2]:= nill
            end
      else
```

```
          begin low:= cc[lalow];  high:= cc[lahigh];
                if sp<low
                then
head:                begin cc[lalow-2]:= cc[p]:= la;
                           cc[la-1]:= lalow;  cc[la-2]:= nill
                     end
                else
                     begin if high<sp
                           then
tail:                           begin cc[lahigh-1]:= cc[p-1]:= la;
                                      cc[la-1]:= nill;  cc[la-2]:= lahigh
                                end
                           else
                                begin if (sp-low) < (high-sp)
                                      then
sort forward:                              begin succ:= lalow;
                                                 for dummy:=0 while sp>cc[succ] do succ:= cc[succ-1];
                                                 pred:= cc[succ-2]
                                           end
                                      else
sort backward:                             begin pred:= lahigh;
                                                 for dummy:=0 while sp<cc[pred] do pred:= cc[pred-2];
                                                 succ:= cc[pred-1]
                                           end;
                                      cc[succ-2]:= cc[pred-1]:= la;
                                      cc[la-1]:= succ;  cc[la-2]:= pred
                                end
                     end
          end;
     cc[la-3]:= j;
     cc[p+1]:= cc[p+1]+1
end sort in chain;
```

```
integer procedure lafirst(j); value j; integer j;
begin
      if testq(j) then begin errordata;  jumpout end;
      lafirst:= cc[3×j+1]
end la first;


integer procedure lalast(j); value j; integer j;
begin
      if testq(j) then begin errordata;  jumpout end;
      lalast:= cc[3×j]
end la last;


integer procedure lasucc(la); value la; integer la;
begin
      if testla(la) then begin errordata;  jumpout end;
      lasucc:= cc[la-1]
end la successor;


integer procedure lapred(la); value la; integer la;
begin
      if testla(la) then begin errordata;  jumpout end;
      lapred:= cc[la-2]
end la predecessor;
```

```
real procedure datafrom(la, i); value la, i; integer la, i;
begin
      if testla(la) ∨ testc(i) then begin errordata;  jumpout end;
      datafrom:= cc[la+i]
end data from;


procedure datato(la, i, data); value la, i, data; integer la, i; real data;
begin
      if testla(la) ∨ testc(i) then begin errordata;  jumpout end;
      cc[la+i]:= data
end data to;


integer procedure length(j); value j; integer j;
begin
      if testq(j) then begin errordata;  jumpout end;
      length:= cc[3×j+2]
end length of queue;
```

10.5 Algol tekst van de bewakingsprocedures.

```
boolean procedure testla(la);  value la;  integer la;
begin
      integer p, ll, bla;
      p:= cc[-3];  ll:= cc[-4]+4;  testla:= false;
      bla:= la-p×3-3;
      if la<0 ∨ la>cc[-1] ∨ (bla:ll×ll≠bla)
      then begin NLCR; PRINTTEXT(‹incorrect link address›); PRINT(la);
                 testla:= true
           end
end test la;


boolean procedure testq(j);  value j; integer j;
begin
      integer p;
      testq:= false;
      p:= cc[-3];
      if j<0 ∨ j≥p
      then begin NLCR; PRINTTEXT(‹incorrect number of queue›); PRINT(j);
                 testq:= true
           end
end test queue;


boolean procedure testc(i);  value i;  integer i;
begin
      testc:= false;
      if i<0 ∨ i>cc[-4]
      then begin NLCR; PRINTTEXT(‹incorrect number of cell›); PRINT(i);
                 testc:= true
           end
end test of cell;
```

```
boolean procedure testlaq(la, j);  value la, j;  integer la, j;
begin
      boolean error;
      testlaq:= false;
      eror:= false; if testla(la) then eror:= true;  if testq(j) then error:= true;
      if ⌝error
      then begin
                  if j≠cc[la-3]
                  then begin NLCR; PRINTTEXT(<link address does not correspond with given number of queue>);
                             NLCR; PRINTTEXT(<link is member of queue>); PRINT(cc[la-3]);
                             NLCR; PRINTTEXT(<given number of queue is>); PRINT(j);
                             testlaq:= true
                       end
           end;
      if error  then testlaq:= true
end test la and queue;


procedure errordata;
begin
      integer j, lengthj, l, la, p;
      CARRIAGE(2);  PRINTTEXT(<data of array cc>);
      NLCR; PRINTTEXT(<  i    array[i]>);
      for j:=-5 step 1 until -1 do begin NLCR; FIXT(2,0, j);  PRINT(cc[j]) end;
      p:= cc[-3]; NLCR;
      NLCR; PRINTTEXT(<lengths of queues>);
      NLCR; PRINTTEXT(<  i    queue[i]>);
      for j:=0 step 1 until p-1 do
      begin
            NLCR; ABSFIXT(2,0,j); lengthj:= length(j); ABSFIXT(5, 0, lengthj);
            l:= 0;  la:= lafirst(j);
            if la≠nill then
            begin l:=1;
                  for la:= lasucc(la) while la≠nill do l:= l+1;
            end;
            if lengthj≠l then begin PRINTTEXT(<erroneous      length =>); FIXT(5,0, l)  end;
      end
end errordata;
```

```
procedure jumpout;
begin
      boolean array error[1:1];  error[2]:= true
end jump out of bounds;
```

10.6 Algol tekst van de histogram procedures.

```
procedure histdef(nhist); value nhist; integer nhist;
begin integer i, j;
      ubhist:= 0;  PRINTTEXT(‹nr. histogram,   akl,   low,   klbr,   state›);
      for i:=1 step 1 until nhist do
      begin NLCR;  ABSFIXT(5,0,i);
            for j:=1 step 1 until 4 do PRINT((histdata[i,j]:= read));
            ubhist:= ubhist+histdata[i,1]+8;
      end;
end histdef;

procedure histset(nhist);value nhist; integer nhist;
begin
      integer i, p;
      for i:=1 step 1 until ubhist do hist[i]:= 0;
      p:= 1;
      for i:=1 step 1 until nhist do
      begin hist[-i]:= p;
            hist[p+1]:= ₁₀600;  hist[p+2]:= -₁₀600;
            p:= p+histdata[i,1]+8;
      end
    ; hist[0]:= nhist;
end histset;
```

```
procedure histput(j, x, nox);
value j, x, nox; integer j, nox; real x;
begin
      integer p, cell, nc;
      real low, cw, sumx;
      procedure jumpout; begin boolean array error[1:1]; error[2]:= true end jumpout;
      if nox<0 then goto ready;
      if j<1 ∨ j>hist[0] then begin PRINTTEXT(⟨incorrect number of histogram⟩); PRINT(j);
                                    jumpout;
                              end;
      nc:= histdata[j,1];  low:= histdata[j,2];  cw:= histdata[j,3];
      p:= hist[-j];
      if abs(x)<10↑-10 then hist[p+7+nc]:= hist[p+7+nc]+1;
      if x<low
      then cell:= -1
      else begin cell:= entier( (x-low)/cw );
                 if cell>nc then cell:= nc
           end;
      cell:= p+6+cell;
fill: hist[cell]:= hist[cell]+nox;  hist[p]:= hist[p]+nox;
      if x<hist[p+1] then hist[p+1]:= x
    ; if x>hist[p+2] then hist[p+2]:= x;
      if x≠0 then begin sumx:= nox×x;
                        hist[p+3]:= hist[p+3]+sumx;
                        hist[p+4]:= hist[p+4]+sumx×x
                  end;
ready:
end histput;


procedure histmv(j, m, v);
value j; integer j; real m,v;
begin
      real sum, sumsq;
      integer p, n;
      p:= hist[-j];
      n:= hist[p];  sum:= hist[p+3];  sumsq:= hist[p+4];
      if n=0 then m:= v:= nill
             else m:= sum/n;  v:= if n>1 then (sumsq-sum×sum/n)/(n-1)  else 0;
end histmv;
```

```
procedure histprint(string, j);
value j; integer j; string string;
begin
      boolean state;
      integer nc, n, i, dec, dec1, p
              , iperc, inperc, icumperc, incumperc, entries, nentries, max
                , k, cument
;
      real low, cw, cbegin, cend, m, v
           , rperc, rnperc, rcumperc,rncumperc
;
      integer array line[0:100]
;
integer procedure decimals(v);
      value v; real v;
      begin
            integer dec;
            dec:= -1;
            v:= abs(v);  for dec:=dec+1 while v-entier(v) > 0.0000001×v do v:= v×10;
            decimals:= dec
      end decimals
;
      p:= histdata[j,4]; state:= p=0 ∨ p=2;
      p:= hist[-j];
      n:= hist[p];  NLCR;
      if LINENUMBER+histdata[j,1]>48 then NEWPAGE else CARRIAGE(4);
      PRINTTEXT(string); ABSFIXT(2,0,j);  SPACE(10);
      if n=0 then begin NLCR; PRINTTEXT(<histogram empty>); goto ready end;
      if n<0 then begin NLCR; PRINTTEXT(<incorrect histogram>) end;
      cument:= 0;
      nc:= histdata[j,1];  low:= histdata[j,2];  cw:= histdata[j,3];
      dec:= decimals(cw);  dec1:= decimals(low);  if dec1>dec then dec:= dec1;
      histmv(j, m, v);
      NLCR; PRINTTEXT(<mean    >); FIXT(8, dec+1, m);
      NLCR; PRINTTEXT(<variance>); FIXT(8, dec+1, v);
      NLCR; PRINTTEXT(<minimum >); FIXT(8, dec, hist[p+1]);
      NLCR; PRINTTEXT(<maximum >); FIXT(8, dec, hist[p+2])
;     NLCR; PRINTTEXT(<entries >); ABSFIXT(8,0,n); PRINTTEXT(<    ( zero entries>); A
                                   ABSFIXT(5,0,hist[p+7+nc]); PRINTTEXT(<)>);
     CARRIAGE(2);
     if dec>5 then dec:= 5;
```

```
      PRINTTEXT(‹ frequency                    cum.›); NLCR;
      PRINTTEXT(‹  classes   entries  perc.   perc.   ›);
      PRINTTEXT(‹      10|      20|      30|      40|      50|      60|      70|      80|      90|      100|›);
      NLCR;      cbegin:= low-cw;
      nentries := hist[p+5]; rnperc:= 100×nentries/n; inperc:= rnperc; rncumperc:= rnperc; incumperc:= rncumperc;
      SPACE(38);  for i:=0 step 1 until inperc do PRCHAR(126);
      for i:=-1 step 1 until nc do
      begin
            NLCR;  cbegin:=cbegin+cw;
            entries:= nentries; rperc:= rnperc;  iperc:= inperc;  rcumperc:= rncumperc;  icumperc:= incumperc;
            if state ∧ i>-1 ∧ i<nc
            then begin SPACE(1);  if i=0 then cbegin:= low end
                                  else if i<nc then PRINTTEXT(‹<›) else PRINTTEXT(‹>›);
            FIXT(4, dec, if i<nc then cbegin else cbegin-cw);
            SPACE(if dec=0 then 6 else 5-dec);
            ABSFIXT(4,0,entries); ABSFIXT(5,2,rperc); ABSFIXT(3,2,rcumperc); SPACE(1);
            cument:= cument+nentries;
            if cument=n then begin NLCR;  PRINTTEXT(‹remainder empty›);  goto ready end;
            if i=nc then goto L;
            nentries := hist[p+7+i];
            rnperc:= 100×nentries/n;  inperc:= rnperc;  rncumperc:= rncumperc+rnperc;  incumperc:= rncumperc;
        L:  max:= if iperc>inperc then iperc else inperc;
            for k:=1 step 1 until max do line[k]:= 126;
            for k:= max+1 step 1 until icumperc do line[k]:= 93;
            for k:= icumperc+1 step 1 until incumperc do line[k]:= 126;
            line[0]:= line[iperc]:= line[icumperc]:= 255;
            for k:=0 step 1 until incumperc do PRCHAR(line[k]);
      end;
ready:
NLCR;
end histprint;
```

10.7 Magneetband.

Het procedurepakket "ASP" zou, indien op ponsband, een behoorlijk stuk ponsband vullen. Het manipuleren ermee geeft nogal eens aanleiding tot moeilijkheden (lees-fouten, scheuren in de band e.d.). Om de gebruiker te gerieven zijn de procedures op een magneetband gezet. Iedere procedure staat in één groep [6] . D.m.v. een eenvoudige aanroep kan de gebruiker over de procedures beschikken (zie 10.8).

Voor de bewaakte versie (zie 9.5 en 10) de aanroep mtape 234, 11-21, 30-61; (zie 10.8.1 regel 14).

Door deze aanroep wordt op de plaats van de aanroep de tekst van de ketting-kast procedures (groepen 30-45), de bewakings procedures(groepen 51-53) en de histogramprocedures (groepen 11-14) met uitzondering van "histdef" (groep 10), die in een buitenblok middels de aanroep:

mtape 234, 10;

geinsereerd wordt (zie 10.8.1 regel 8).

Voor de onbewaakte versie is de aanroep:

mtape 234, 11-21, 70-87;

De aanroep van histdef is hetzelfde als bij de bewaakte versie.

10.8 Het simulatieprogramma.

10.8.1 De structuur van het programma.

```
0  Lalgol 0816 ..... simulatie,0,0,1

1  begin
2       integer ubcc, nhist;
3       upper bound of array cc: ubcc:= read;
4       number of histograms : nhist:= read;

5       begin
6            real array cc [-5:ubcc], histdata [1:nhist, 1:4];
7            integer, ubhist;
8            mtape 234, 10; histdef(nhist);
9            histdef(nhist);

10           begin
11                real array hist [-nhist:ubhist];
12                integer la, nil1, dummy;
13                real frand;
14                mtape 234, 11-21, 30-61;

15                 nil1:= -282317;
16                initialize chain case: initial (ubcc, read, read);
17                initialize hist      : histset (nhist);
18                initialize random    : SETRANDOM(read);

19                begin  }
                         } het hoofdprogramma om de simulatie uit te voeren
                  end    }
           end
      end
end
progend
```

De regelnummering behoort niet tot het programma en dient slechts om in de tekst op eenvoudige wijze naar bepaalde regels te kunnen verwijzen.
De hier gebruikte "commentaar-labels" kunnen worden weggelaten.

10.8.2 de input

Voor het onder 10.8.1 beschreven programma is "op ponsband" tenminste de volgende input nodig, in de hier genoemde volgorde:

	regel in 10.8.1
- bovengrens van "cc"	3
- het aantal gewenste histogrammen	4
- de histogramgegevens , d.w.z. per histogram:	9
- aantal klassen	
- de ondergrens van de eerste klasse	
- de klassebreedte	
- "state", d.w.z. indien men in het histogram alleen geheeltallige waarden wenst te turven dan dient men "0" op de band te staan, onder een "1".	
- aantal voor de gebruiker te reserveren plaatsen in de schakel	16
- aantal kettingen	16
- de startwaarde voor de procedure RANDOM	18

Hierna volgt de input zoals de gebruiker die eventueel in zijn hoofdprogramma nodig heeft.

11. Het gebruik van ASP, voorbeelden en opgaven.

Bij gebruik van het procedurepakket ASP dienen we te beginnen met het ontwerpen van een schakelindeling. Deze schakelindeling kan door de gebruiker op vele manieren gekozen worden. De door ons gekozen indeling in de voorbeelden geven slechts een mogelijkheid. De gekozen indeling en het programma dienen met elkaar in overeenstemming te zijn. Onder schakelindeling wordt in het vervolg verstaan: de indeling van het voor de gebruiker gereserveerde stuk, de indeling van het overige stuk is immers vast en wordt door het pakket bepaald.
Ook de rij van events kan gerepresenteerd worden als een ketting. In ASP is géén apart mechanisme voor het maken van een gesorteerde lijst van events, immers de kettingkastprocedures bieden hiertoe al het benodigde gereedschap. Om de eventrij te representeren gebruiken we altijd ketting "0".

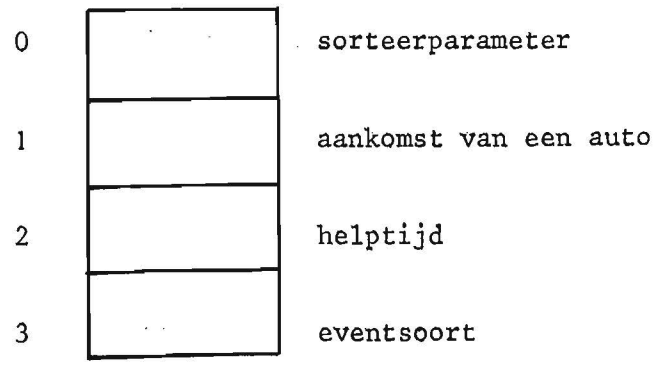
De volgende vereenvoudiging is mogelijk. We gebruiken de schakel voor twee doeleinden:
- soms representeert de schakel een auto die zich in de wachtrij bevindt,
- soms representeert de schakel, een auto die in behandeling is of nog in het systeem moet komen. De schakel staat dan in de eventrij en representeert dan tevens een event.

Bij de schakelindeling dient men hiermee rekening te houden.

Men zou ook aparte eventschakels kunnen maken. In gecompliceerde gevallen bv. wanneer de order als een netwerk door het systeem gaat is deze vereenvoudiging niet meer mogelijk (Meerdere onderdelen van de order zijn tegelijkertijd in bewerking).

11.1 Benzinepomp, "FIFO" (zie 5.6)

11.1.1 Schakelindeling



11.1.2 de events: zie 5.2

```
Lalgol 08163764benzinepomp,0,0,1
begin integer ubcc;
      ubcc:=read;
      begin real array cc[-5:ubcc];
            integer la, nill, dummy;
            real frand;
            mtape 234,21,30-61;
            nill:=-282317;
            initial(ubcc,3,2);
            SETRANDOM(read);
            begin integer readycar, totcar,evnumb,p;
                  real now, mu1, mu2, t, sumwt, sumsqwt;
                  boolean pumpfree;

                  procedure generate;
                  begin la:=fromf;
                        datato(la,1,now-mu1×ln(RANDOM));
                        datato(la,2,-mu2×ln(RANDOM));
                        datato(la,3,2);
                        datato(la,0,datafrom(la,1)); toort(la,0);
                  end generate;

                  totcar:=read; mu1:=read; mu2:=read; p:= read;
                   now:=0; readycar:=0; pumpfree:=true;
                   sumwt:=sumsqwt:=0;
                   generate;

program:           for dummy:=0 while readycar < totcar do
                   begin
clock:                   la:=lafirst(0); fromch(la,0);
                         now:=datafrom(la,0); evnumb:=datafrom(la,3);
                         if evnumb = 2
                         then
new car:                       begin if length(1) < p
                                     then begin datato(la,3,1);
                                                totail(la,1)
                                          end
                                     else tof(la);
                                     generate;
                               end
                         else
pump is free:                  begin pumpfree:=true;
                                     readycar:=readycar+1;
                                     tof(la);
                               end;
                         if pumpfree ∧ length(1) > 0
                         then
occupy pump:                   begin pumpfree:=false;
                                     la:=lafirst(1); fromch(la,1);
                                     t:=now-datafrom(la,1);
                                     sumwt:=sumwt+t; sumsqwt:=sumsqwt+t×t;
                                     datato(la,0,datafrom(la,2)+now);
                                     tosort(la,0);
                               end
                   end;
output:            NLCR;PRINTTEXT(« p »); PRINT(p);
                   NLCR;PRINTTEXT(« systeemtijd »); PRINT(now);
                   NLCR;PRINTTEXT(« mean waiting time = »); PRINT(sumwt/totcar);
                   NLCR;PRINTTEXT(« variance of this  = »);
                        PRINT((sumsqwt-sumwt×sumwt/totcar)/(totcar-1))
            end
      end
end
progend
```

11.2 Opgaven

11.2.1 Opgave 6.7

11.2.2 Opgave 5.5 met een histogram van de wachttijd.

11.2.3 Opgave 6.7 met een histogram van de wachttijd.

11.2.4 In een fabriek staan 2 machines. De orders gaan allemaal eerst naar machine 1 en vervolgens naar machine 2. De tussenaankomsttijden van de orders zijn 4 dagen, de bewerkingstijden bij machine 1 en 2 zijn resp, 3 en 3.5 dag. De queuedisciplines voor de machines 1 en 2 zijn resp. FIFO en Korste bewerkingstijd.
Gevraagd: gemiddelde, variantie en histogram van:
- wachttijden voor machine 1
- wachttijden voor machine 2
- doorlooptijd van de orders

11.2.5 Als 11.2.4. Er zijn nu echter 2 ordersoorten:
- Ordersoort 1 gaat eerst naar machine 1 en vervolgens naar machine 2
- ordersoort 2 gaat eerst naar machine 2 en vervolgens naar machine 1.

De tussenaankomsttijd van ordersoort 1 = 6, evenals die van ordersoort 2.

12. Reproduceerbaarheid.

Veelal is het gewenst om d.m.v. simulatie verschillende queue disciplines met elkaar te vergelijken. Met het oog op het klein houden van de variantie [8], is het gewenst, dat de verschillende simulaties van hetzelfde orderpakket gebruik maken. Dit kan men bereiken door:

- eenmalig het orderpakket tegenererenen op een magneetband te schrijven en deze als input voor de simulaties te gebruiken. Deze methode is efficient indien men een zeer groot aantal experimenten met hetzelfde zeer ingewikkelde orderpakket wenst te maken.
- door bij het ontstaan van de order alle stochastische attribuutnotities (bv. helptijden) te genereren. De randomgenerator immers levert bij dezelfde startwaarde dezelfde reeks getallen. Iedere aanroep van RANDOM op een andere plaats dan in de procedure "generate" dient vermeden te worden. Indien dit laatste niet mogelijk is kan van een andere RANDOM-generator gebruik worden gemaakt.

Indien men geen reproduceerbaarheid vereist kan men ook de stochastische attribuutnotities trekken op het moment dat men ze nodig heeft. De schakelindeling en administratieve beslommeringen worden hierdoor eenvoudiger.
We achten de reproduceerbaarheid echter zo belangrijk, dat we van deze vereenvoudigingsmogelijkheid geen gebruik hebben gemaakt.
In sommige simulaties heeft men te maken met orders die gesplitst worden in onderdelen welke later weer samengevoegd dienen te worden (motor revisie). De orders bewegen zich als een vertakt netwerk door het systeem. Indien men bij deze meer gecompliceerde gevallen reproduceerbaarheid vereist dan dient de schakel een netwerk te representeren. Men dient ook, indien nodig, de weg in het netwerk te kunnen vinden. De administratieve overlast kan onoverkomelijk worden. Wanneer men de eis tot reproduceerbaarheid laat vallen zijn ook simulaties met vertakte orders redelijk eenvoudig te programmeren.
Ook voor de simulatietalen geldt dat de eis tot reproduceerbaarheid aanleiding geeft tot gecompliceerde programma's.
In sommige gevallen is reproduceerbaarheid niet of nauwelijks te verwezenliijken bv. wanneer de keuze van de volgende machine afhangt van de rijlengte op dat moment en de gemiddelde helptijden bij de machines verschillend zijn.

13. Simulatie van een job-shop.

13.1 Probleemstelling.

Een jobshop bestaat uit 3 werkplaatsen, in het vervolg eenvoudige machines genoemd. De bewerkingstijd op machine i (i=1, 2 of 3) is negatief-exponentieel verdeeld met gemiddelde mu[i]:

mu[1]= 0.1

mu[2]= 0.2

mu[3]= 0.3

Bij ieder van de 3 ordertypen geldt dat hun tussen-aankomsttijden negatief exponentieel verdeeld zijn met gemiddelde 1.

Specificatie van de 3 ordertypen:

type 1: Deze orders ondergaan 2 bewerkingen. Eerst op machine 1, daarna een bewerking op machine 2 of 3 beide met een kans p = $\frac{1}{2}$

type 2: 50% doorloopt de machines in de volgorde 2-1-3; de andere helft in de volgorde 3-1-2.

type 3: Deze orders doorlopen de machines in de volgorde 3-2-1.

Indien een machine een order bewerkt heeft van type i en daarna een order moet gaan bewerken van type j ($i \neq j$), dan moet de machine omgeschakeld worden. De machines hebben bij het omschakelen van ordersoort "i" naar ordersoort "j" allen dezelfde omschakeltijd.

Tabel van omschakeltijden:

van ordertype \ naar ordertype	1	2	3
1	0	0.4	0.5
2	0.4	0	0.6
3	0.5	0.6	0

In deze omschakeltijden nu schuilt het probleem. De bewerkingstijden zijn klein t.o.v. de tussenaankomsttijden. De omschakeltijden zijn echter ten opzichte hiervan zo groot dat een willekeurige verwerking van ordertypes door elkaar tot zeer lange wachtrijen zou leiden. Hiermee rekening houdend en er

naar strevend de langste doorlooptijden, desnoods ten koste van het gemiddelde, zo kort mogelijk te houden, is in de jobshop de volgende queuediscipline ingevoerd:
Indien een order op een machine gereed is gekomen, wordt de order in bewerking genomen die van hetzelfde type is en het langst in het systeem is. Is er geen dergelijke order dan wordt de order in bewerking genomen die al het langst in het systeem is, ongeacht het type.

Gevraagd: gemiddelde, variantie en histogrammen van de doorlooptijden van de 3 verschillende ordertypes bij in totaal 1000 verwerkte orders.

13.2 De schakelindeling.
Bij elke order dienen de volgende parameters te worden genoteerd:
- de eventsoort (zie 11)
- het type
- het moment van ontstaan: nodig voor het berekenen van de doorlooptijd
- de routing en de helptijden bij de machines: dit zijn in totaal zes parameters
- het aantal af te werken en het aantal reeds afgewerkte machines: deze twee parameters zijn nodig om na te gaan of de orders gereed zijn en zoniet welke de eerstvolgende bewerkingsgang is.

In totaal elf parameters.
We delen de schakel als volgt in:

0	sorteerparameter	
1	eventsoort	
2	type	door gebruiker op te geven aantal plaatsen per schakel = 11
3	ontstaan	
4	af te werken	
5	afgewerkt	
6	nr. eerste machine	
7	nr. tweede machine	
8	nr. derde machine	
9	helptijd eerste machine	
10	helptijd tweede machine	
11	helptijd derde machine	

13.3 De events.

Er zijn twee soorten events:

- een van de drie machines komt gereed met een bewerking ("eventsoort" = 1)
- er komt een nieuwe order binnen ("eventsoort" = 2).

Ook in dit voorbeeld zal een schakel voor twee doeleinden gebruikt worden. Afhankelijk van de ketting waarin hij zich bevindt representeert hij een order in een wachtrij of een event (zie 11). Bij de schakelindeling is hiermee rekening gehouden. De parameter op plaats 1 geeft de eventsoort aan. Deze heeft alleen betekenis zolang de schakel een event representeert.
Het eventtijdstip wordt genoteerd op plaats "0" en fungeert hierdoor in de eventrij als sorteerparameter. Bij het gereedkomen van een machine leveren de parameters "af te werken", "afgewerkt" en de "routing" de gegevens nodig om te bepalen welke machine gereed gekomen is en wat er verder met de order moet gebeuren.
In de eventrij kan men drie tot zes eventschakels aantreffen, nl.

- drie of minder van soort 1
- drie van soort 2

13.4 De wachtrijen.

Het lijkt voor de hand liggend om voor iedere machine één wachtrij te formeren, waarin de drie ordertypen staan gesorteerd naar "aankomst in het systeem".
Als we echter naar de queue-discipline kijken, zien we, dat als een order gereed komt,er indien aanwezig, een order van hetzelfde type moet worden genomen.
In het programma kunnen we gebruik maken van de procedure "lasucc". Bv. als volgt:

```
la:= lafirst(j);
L: if la ≠ nill
   then begin if datafrom (la,2) = i
              then begin la:= lasucc(la);
                         goto L
                   end
        end;
```

waarbij j het machinenummer is en i het "preferente" order type.
In dit stukje programma blijkt, dat i.h.a. een aantal vergeefse zoekpogingen zullen worden uitgevoerd alvorens een order van het gewenste type gevonden wordt. Als we reeds geruime tijd bezig zijn met een bepaald type, zullen er

van dit type nog slechts een gering aantal in de wachtrij staan. Het aantal van de andere typen die reeds eerder in het systeem gekomen zijn neemt relatief toe. Het aantal vergeefse zoekpogingen wordt groot. Ook het gesorteerd in de ketting zetten van een schakel met behulp van "to sort", kan vrij veel rekentijd kosten. Terwille van de efficientie van het programma zullen we daarom niet werken met één maar met drie kettingen per machine. Voor ieder ordertype één ketting. Het is waarschijnlijk, dat in het reële systeem, de chef van de werkplaats ter vermijding van overmatig sorteer- en zoekwerk bij iedere machine ook met drie wachtrijen zal werken. Wanneer men wil pogen efficient te simuleren is het nuttig zich te realiseren welke administratieve vereenvoudigingen men in het reële systeem van zelfsprekend zou toepassen.

Er ontstaan zo 9 wachtrijen:

machine	ordertype	wachtrij
1	1	1
	2	2
	3	3
2	1	4
	2	5
	3	6
3	1	7
	2	8
	3	9
algemeen: i	j	3 x (i-1) + j

Het berekenen van de bij een machine behorende wachtrijnummers is iets gecompliceerder geworden. Dit nadeel is echter aanzienlijk geringer dan de behaalde voordelen bij zoek- en sorteerwerk.

Gezien de queue-discipline zullen de lengten van de 9 wachtrijen tijdens het programma sterk varieren. Iedere wachtrij zal, als het betreffende ordertype niet "preferent" is lang worden. Het in 7.1.1 genoemde systeem van een gemajoreerd 2-dimensionaal array zou hier inderdaad een nodeloos groot beslag leggen op de beschikbare geheugenruimte.

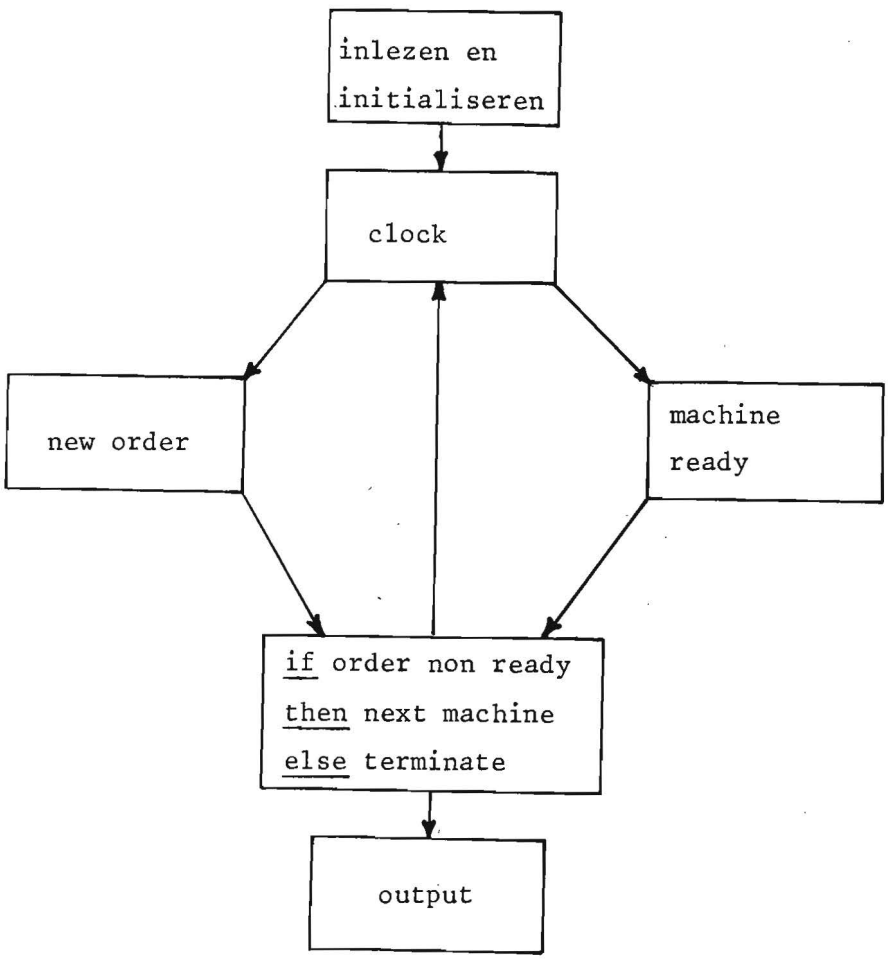
13.5 Controle op de input.

In het onder 10.8.1 besproken programma worden de systeemvariabelen ("ubcc" etc.) ingelezen. Men heeft hierbij echter achteraf geen enkele controle of de

bedoelde waarden inderdaad zijn ingelezen. In praktische toepassingen is het belangrijk op dit punt zekerheid te hebben. Men kan bv. een verkeerd bandje inleveren of de operateur bij de computer kan een fout maken. Detectie van dit soort fouten is bijna onmogelijk.
Dit is de reden dat in dit programma terstond na het inlezen van de waarde van een variabele wordt afgedrukt:
- de naam of omschrijving van de variabele d.m.v. een aanroep van "PRINTTEXT"
- de waarde van de variabele d.m.v. "PRINT".

Ook de ingelezen matrices worden terstond met een omschrijving afgedrukt. In de procedure "histdef" gebeurt dit automatisch.
Deze taktiek is zowel in het "systeemgedeelte" als in "gebruikers gedeelte" van het programma toegepast.
De in 10.8.1 gebruikte "commentaar-labels" zijn weggelaten. Deze zijn door de diverse aanroepen van "PRINTTEXT" ook als verduidelijking overbodig geworden.

13.6 Het stroomschema



13.7 Toelichting bij het stroomschema.

13.7.1 Clock.

De eerste schakel wordt van de eventry (ketting "0") gehaald. De schakel staat nu "los". De variabele "la" krijgt de waarde van het schakeladres. Afhankelijk van de eventsoort wordt het programmastuk "new order" of "machine ready" uitgevoerd.

13.7.2 New order.

De parameter "eventsoort" in de schakel krijgt de waarde "1". Immers wanneer deze schakel in de toekomst een event zal representeren zal dit steeds het gereedkomen van een order zijn. Door de aanroep van de procedure "generate" wordt in een nieuwe schakel een toekomstige order gegenereerd, die vervolgens in de eventrij wordt gezet.
Vervolgens stellen we de vraag: order non ready?, terwijl we van deze nieuwe order zeker weten, dat hij nog niet gereed is en altijd het programmastuk "next machine" zal worden uitgevoerd. We zouden deze overbodige vraagstelling kunnen vermijden door het statement: goto next machine. Terwille van de eenvoud en controleerbaarheid van het programma is dit zeer geringe efficientie verlies voor lief genomen. Bij het streven naar efficientie mag men als een goed jager de haas laten lopen als er een olifant in de buurt is.

13.7.3 machine ready

Een bewerking van een order is op een bepaalde machine klaar gekomen.De parameter "afgewerkt" wordt met 1 verhoogd.
Is er in de wachtrij voor de machine nog een order die op verwerking wacht dan wordt deze in bewerking genomen. Let wel, hier speelt de queue-discipline een rol. Om het hoofdprogramma niet al te ingewikkeld te maken is hiervoor een procedure "takenext" geschreven die uit de 3 wachtrijen voor deze machine de juiste volgende bewerking selecteert.

13.7.4 next machine.

Een order wordt geplaatst in de wachtrij van zijn volgende bewerking en daar zo mogelijk in bewerking genomen.

13.7.5 terminate

Als een order gereed is ("afgewerkt" = "af te werken") worden de statistieken bijgehouden d.m.v. een aanroep van "histput".
De schakel wordt weer vrijgegeven d.m.v. een aanroep van "tof".

13.7.6 De input voor het gebruikersgedeelte.
Achtereenvolgens worden ingelezen:
- het aantal af te werken orders (de variabele "n")
- de omschakeltijden (het array "switch")
- de gemiddelde bedieningstijden (het array "mu").

13.7.7 De initialisering.
De variabele "now", "ready" en het array "qlength" krijgen de waarde "0".

Bij het array "busy with" krijgt ieder element busywith[i], de waarde "i". Dit is een willekeurige keuze. Het starten met een leeg systeem is echter ook een willekeurige keuze. Het zou beter zijn te starten met bv. de stationaire toestand van het systeem. Deze toestand kent men echter niet. In het algemeen zal men daarom starten met een leeg systeem en bv. de eerste 100 orders niet in de statistieken betrekken in de hoop dat het systeem na deze aanloopfase van 100 orders de stationaire toestand bereikt zal hebben. Het implementeren van deze taktiek is eenvoudig. Om didactische redenen is de implementatie hiervan in deze inleiding achterwege gelaten.

Van elk ordertype wordt een order gegenereerd d.m.v. de procedure "generate" en als event in de eventrij geplaatst. In wezen start men op systeemtijd "0" met een leeg systeem.

13.7.8 De output.
Naast een overzicht van de input (zie 13.7.6) wordt afgedrukt:
- het aantal afgewerkte orders
- de systeemtijd
- per ordertype de door "histprint" verzorgde output (zie 10.3.2.5) t.a.v de doorlooptijden.

13.8 Namen en hun betekenissen.
In onderstaande lijst is de volgorde van de namen van de variabelen gelijk

aan de volgorde van declaratie in het gebruikersprogramma.

mnemonic	betekenis
i, j	hulptellers
n	aantal te verwerken orders
ready	aantal verwerkte orders
type	het type van een order
lanew	hulpvariabele voor een schakeladres
mnumb	machine nummer
numready	aantal verrichte bewerkingen aan een order
nextm	de machine waarop de eerstvolgende bewerking van een order plaats vindt
qnumb	het nummer van de wachtrij
eventsoort	de soort event
t	hulpvariabele
now	systeemtijd
qlength[1:3]	de som van de lengten der wachtrijen voor elk van de drie machines
busy with[1:3]	het ordertype dat het laatst op de verschillende machines is verwerkt
switch[1:3, 1:3]	de omschakeltijden der machines
mu[1:3]	de gemiddelde bewerkingstijden voor elk van de drie machines
generate	procedure die het genereren van een toekomstige order verzorgd
takenext	procedure die het in bewerking nemen van de volgende order op de machine verzorgd.

13.9 Het Algol programma voor het jobshop probleem.

```
XXXXXXXXXXXXXXX

      125 -    0 -    190371 - 01063074 JOBSHOP

    1 BEGIN INTEGER NHIST, UBCC;
    2       PRINTTEXT(*KERBOSCH - KROEP , JOBSHOP*);NLCR;
    3       UBCC:= READ;
    4       PRINTTEXT(*UBCC                          *);PRINT(UBCC);NLCR;
    5       NHIST:= READ;
    6       PRINTTEXT(*NHIST                         *);PRINT(NHIST);NLCR;
    7       BEGIN REAL ARRAY CC[-5:UBCC], HISTDATA[1:NHIST, 1:4];
    8             INTEGER I, J, UBHIST;
    9             MIABE 234,10;
   19             HISTDEP(NHIST);

   20             BEGIN REAL ARRAY HIST[-NHIST:UBHIST];
   21                   INTEGER LA, NILL, DUMMY;
   22                   REAL FRAND;

   23                   MIABE234,11-21,30-61;

  387                   NILL:= -282317;
  388                   INITIAL(UBCC, 11, 10);
  389                   HISTSET(NHIST);
  390                   FRAND:=READ;NLCR;PRINTTEXT(*START RANDOM GENERATOR     *);PRINT(FRAND);NLCR;
  391                   SETRANDOM(FRAND);

  392                   BEGIN INTEGER N, READY,EVENTSOORT, TYPE, LANEW, MNUMB, NUMREADY, NEXTM, QNUMB;
  393                         REAL T, NOW;
  394                         INTEGER ARRAY QLENGTH, BUSYWITH[1:3];
  395                         REAL ARRAY SWITCH[1:3, 1:3], MU[1:3];
```

```
396                     PROCEDURE GENERATE(LA, TYPE);
397                     VALUE LA, TYPE; INTEGER LA, TYPE;
398                     BEGIN
399                           INTEGER FIRST, SECOND, THIRD; REAL T;
400                           T:= NOW-LN(RANDOM);
401                           DATATO(LA, 0, T); DATATO(LA,3, T);DATATO(LA,1, 2);
402                           DATATO(LA,2,TYPE); DATATO(LA,5, 0);

403                           IF TYPE=1
404                           THEN BEGIN FIRST:=1; DATATO(LA,4, 2);
405                                      SECOND:= IF RANDOM≤0.5 THEN 2 ELSE 3;
406                                END
407                           ELSE BEGIN IF TYPE=2
408                                      THEN BEGIN SECOND:= 1;
409                                                 IF RANDOM≤0.5
410                                                 THEN BEGIN FIRST:= 2; THIRD:= 3 END
411                                                 ELSE BEGIN FIRST:= 3; THIRD:= 2 END
412                                           END
413                                      ELSE
414                                           BEGIN FIRST:= 3; SECOND:= 2; THIRD:= 1 END
415                                END;
416                           DATATO(LA,6, FIRST); DATATO(LA,9, -MU[FIRST]*LN(RANDOM));
417                           DATATO(LA,7, SECOND); DATATO(LA,10, -MU[SECOND]*LN(RANDOM));
418                           IF TYPE≠1
419                           THEN BEGIN DATATO(LA,8, THIRD); DATATO(LA, 11, -MU[THIRD]*LN(RANDOM));
420                                      DATATO(LA,4, 3)
421                                END
422                     END GENERATE;
```

```
423                 PROCEDURE TAKENEXT(MNUMB);
424                 VALUE MNUMB; INTEGER MNUMB;
425                 BEGIN
426                       INTEGER LA, NEWTYPE, QNUMB, NEWQNUMB,FIRST, LAST, I;
427                       REAL T, MIN;
428                       QNUMB:= 3*(MNUMB-1)+BUSYWITH[MNUMB];
429                       IF LENGTH(QNUMB)>0
430                       THEN
431                 HETZELFDE TYPE:
432                             BEGIN LA:= LAFIRST(QNUMB); FROMCH(LA, QNUMB);
433                                   T:= NOW+DATAFROM(LA,9+DATAFROM(LA,5))
434                             END
435                       ELSE
436                 ANDER TYPE:
437                             BEGIN MIN:= 10;  NEWQNUMB:= NILL;
438                                   FIRST:= 3*(MNUMB-1)+1;  LAST:= FIRST+2;
439                                   FOR I:=FIRST STEP 1 UNTIL LAST DO
440                                   BEGIN LA:= LAFIRST(I);
441                                         IF LA#NILL
442                                         THEN BEGIN IF DATAFROM(LA, 0)<MIN
443                                                    THEN BEGIN MIN:= DATAFROM(LA, 0);
444                                                               NEWQNUMB:= I
445                                                         END
446                                              END
447                                   END;
448                                   IF NEWQNUMB#NILL
449                                   THEN BEGIN LA:= LAFIRST(NEWQNUMB);
450                                              FROMCH(LA, NEWQNUMB);  NEWTYPE:= DATAFROM(LA,2);
451                                              T:= NOW+SWITCH[BUSYWITH[MNUMB], NEWTYPE]
452                                                  +DATAFROM(LA, 9+DATAFROM(LA,5));
453                                              BUSYWITH[MNUMB]:= NEWTYPE
454                                        END
455                                   ELSE GOTO READY;
456                             END;
457                 NIEUW EVENT:
458                       DATATO(LA, 0, T); TOSORT(LA, 0);
459                 READY:
460                 END TAKE NEXT;
```

```
461 INLEES EN INITIALISEERGEDEELTE:
462                         N:= READ;PRINTTEXT(*N                                   *);PRINT(N);NLCR;
463                         NOW:= 0;
464                         READY:= 0;
465                         NLCR;PRINTTEXT(*TABEL VAN OMSCHAKELTIJDEN*);
466                         FOR I:= 1 STEP 1 UNTIL 3 DO
467                         BEGIN NLCR;
468                               FOR J:= 1 STEP 1 UNTIL 3 DO
469                               BEGIN   SWITCH[I, J]:= READ;PRINT(SWITCH[I,J])  END
470                         END;
471                         NLCR;NLCR;PRINTTEXT(*GEMIDDELDEN DER BEWERKINGSTIJDEN*);NLCR;
472                         FOR I:= 1 STEP 1 UNTIL 3 DO
473                         BEGIN  MU[I]:= READ;PRINT(MU[I])END;
474                         FOR I:= 1 STEP 1 UNTIL 3 DO BEGIN QLENGTH[I]:= 0; BUSYWITH[I]:= 1 END;
475                         FOR I:=1 STEP 1 UNTIL 3 DO
476                         BEGIN LA:= FROMF;
477                               GENERATE(LA, I);
478                               TOSORT(LA, 0)
479                         END;
```

```
480 HOOFDPROGRAMMA;
481                     EOB DUMMY:=0 WHILE READY<N DO
482                     BEGIN
483 CLOCK:                   LA:= LAFIRST(0);  NOW:= DATAFROM(LA, 0); EVENTSOORT:= DATAFROM(LA, 1);
484                          FROMCH(LA, 0);
485                          IF EVENTSOORT=2
486                          THEN
487 NEWORDER:                     BEGIN DATATO(LA, 1, 1);
488                                    TYPE:= DATAFROM(LA,2);
489                                    LANEW:=FROMF;GENERATE(LANEW, TYPE); TOSORT(LANEW, 0);
490                                    NUMREADY:= 0;
491                               END
492                          ELSE
493 MACHINE READY:               BEGIN NUMREADY:= DATAFROM(LA,5);
494                                    MNUMB:= DATAFROM(LA,6+NUMREADY);
495                                    NUMREADY:= NUMREADY+1;DATATO(LA,5, NUMREADY);
496                                    QLENGTH[MNUMB]:= QLENGTH[MNUMB]-1;
497                                    IF QLENGTH[MNUMB]#0 THEN TAKENEXT(MNUMB)
498                               END;

499                          IF NUMREADY<DATAFROM(LA, 4)
500                          THEN
501 NEXT MACHINE:               BEGIN NEXTM:= DATAFROM(LA, 6+NUMREADY);
502                                    QNUMB:= 3*(NEXTM-1)+TYPE;
503                                    DATATO(LA, 0, DATAFROM(LA,3));
504                                    TOSORT(LA, QNUMB);
505                                    QLENGTH[NEXTM]:= QLENGTH[NEXTM]+1;
506                                    IF QLENGTH[NEXTM]=1 THEN TAKENEXT(NEXTM)
507                               END
508                          ELSE
509 TERMINATE:                    BEGIN T:= NOW-DATAFROM(LA, 3); HISTPUT(TYPE, T, 1);
510                                    TOF(LA);
511                                    READY:= READY+1
512                               END
513                     END;
```

```
514 OUTPUT:                          CARRIAGE(3);
515                             PRINTTEXT(‹JOBSHOP PROBLEEM, OUTPUT›);NLCR;
516                             PRINTTEXT(‹AANTAL ORDERS›); PRINT(N);NLCR;
517                             PRINTTEXT(‹SYSTEEMTIJD  ›); PRINT(NOW);NLCR;
518                             FOR I:=1 STEP 1 UNTIL 3 DO
519                                   HISTPRINT(‹HISTOGRAM VAN DOORLOOPTIJDEN VOOR TYPE ›,I);
520                        END OF PROGRAM;

521               END
522          END
523 END

524 PROGEND
```

```
KERBOSCH - KROEP , JOBSHOP
UBCC                                    +10000
NHIST                                      +3
NR. HISTOGRAM,    AKL,    LOW,    KLBR,    STATE
     1              +30                     +0        +.5000000000000₁₀+  0              +1
     2              +30                     +0        +.5000000000000₁₀+  0              +1
     3              +30                     +0        +.5000000000000₁₀+  0              +1
START RANDOM GENERATOR                     +1
N                                       +1000

TABEL VAN OMSCHAKELTIJDEN
            +0        +.4000000000001₁₀+  0 +.5000000000000₁₀+  0
+.4000000000001₁₀+  0              +0        +.6000000000004₁₀+  0
+.5000000000000₁₀+  0 +.6000000000004₁₀+  0              +0

GEMIDDELDEN DER BEWERKINGSTIJDEN
+.1000000000000₁₀+  0 +.2000000000000₁₀+  0 +.2000000000000₁₀+  0


JOBSHOP PROBLEEM, OUTPUT
AANTAL ORDERS          +1000
SYSTEEMTIJD  +.3460904411888₁₀+  3
```

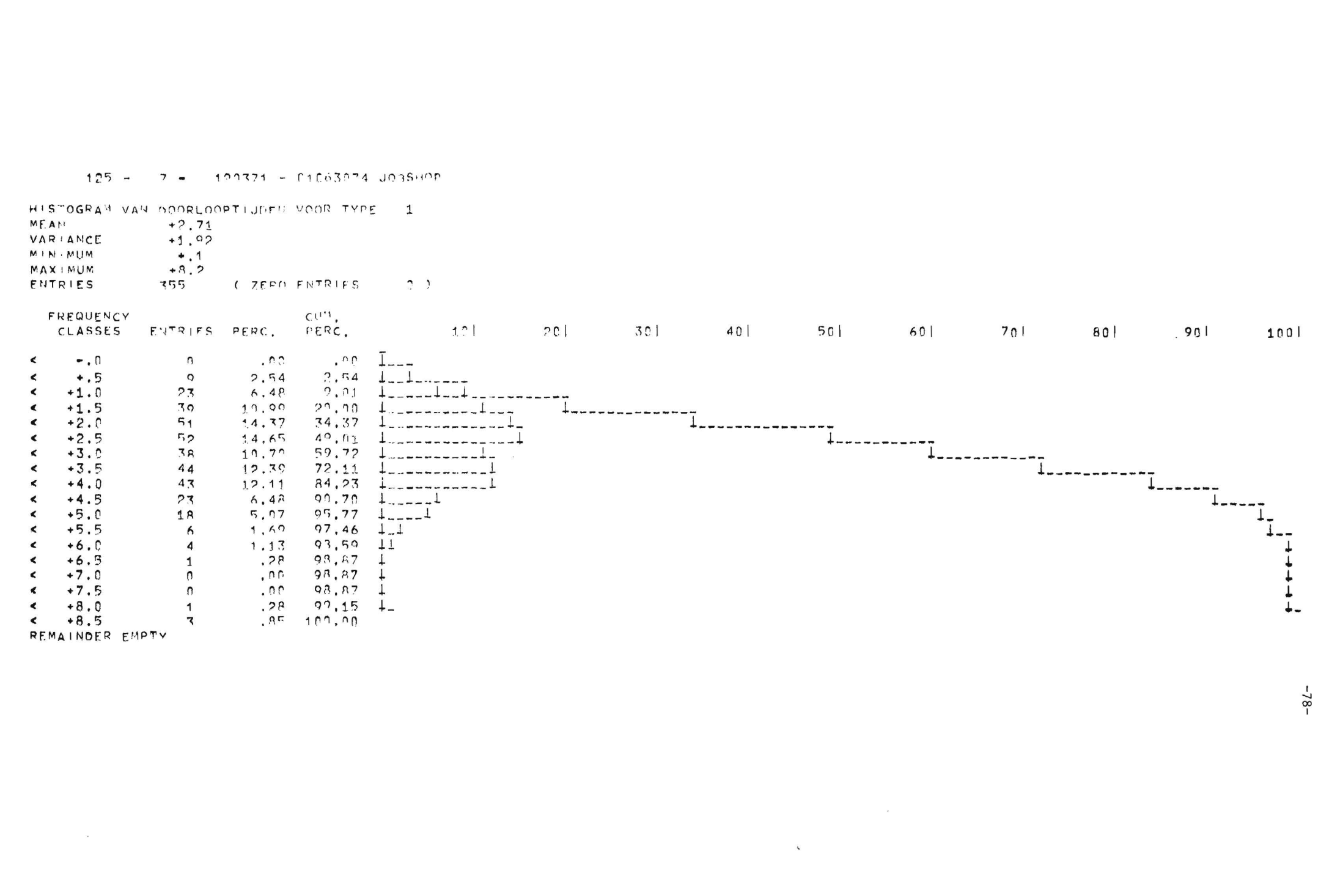
125 - 7 - 100371 - 01063074 JOBSHOP

HISTOGRAM VAN DOORLOOPTIJDEN VOOR TYPE 1

MEAN	+2.71
VARIANCE	+1.92
MINIMUM	+.1
MAXIMUM	+8.2
ENTRIES	355 (ZERO ENTRIES 0)

FREQUENCY CLASSES	ENTRIES	PERC.	CUM. PERC.
< -.0	0	.00	.00
< +.5	9	2.54	2.54
< +1.0	23	6.48	9.01
< +1.5	39	10.99	20.00
< +2.0	51	14.37	34.37
< +2.5	52	14.65	49.01
< +3.0	38	10.70	59.72
< +3.5	44	12.39	72.11
< +4.0	43	12.11	84.23
< +4.5	23	6.48	90.70
< +5.0	18	5.07	95.77
< +5.5	6	1.69	97.46
< +6.0	4	1.13	98.59
< +6.5	1	.28	98.87
< +7.0	0	.00	98.87
< +7.5	0	.00	98.87
< +8.0	1	.28	99.15
< +8.5	3	.85	100.00

REMAINDER EMPTY



125 - 8 = 190371 - 01063074 JOBSHOP

HISTOGRAM VAN DOORLOOPTIJDEN VOOR TYPE 2

MEAN	+2.64
VARIANCE	+1.53
MINIMUM	+.2
MAXIMUM	+8.0
ENTRIES	318 (ZERO ENTRIES 0)

FREQUENCY CLASSES	ENTRIES	PERC.	CUM. PERC.
< -.0	0	.00	.00
< +.5	3	.94	.94
< +1.0	16	5.03	5.97
< +1.5	44	13.84	19.81
< +2.0	52	16.35	36.16
< +2.5	48	15.09	51.26
< +3.0	32	10.06	61.32
< +3.5	43	13.52	74.84
< +4.0	39	12.26	87.11
< +4.5	14	4.40	91.51
< +5.0	17	5.35	96.86
< +5.5	5	1.57	98.43
< +6.0	2	.63	99.06
< +6.5	2	.63	99.69
< +7.0	0	.00	99.69
< +7.5	0	.00	99.69
< +8.0	1	.31	100.00

REMAINDER EMPTY

125 - 0 - 199371 - 01063074 JOBSHOP

HISTOGRAM VAN DOORLOOPTIJDEN VOOR TYPE 3

MEAN	+2.75
VARIANCE	+1.52
MINIMUM	+.3
MAXIMUM	+7.7
ENTRIES	327 (ZERO ENTRIES 0)

FREQUENCY CLASSES	ENTRIES	PERC.	CUM. PERC.
-.0	0	.00	.00
+.5	4	1.22	1.22
+1.0	19	5.81	7.03
+1.5	32	9.79	16.82
+2.0	41	12.54	29.36
+2.5	46	14.07	43.43
+3.0	45	13.76	57.19
+3.5	54	16.51	73.70
+4.0	33	10.09	83.79
+4.5	37	11.31	95.11
+5.0	6	1.83	96.94
+5.5	6	1.83	98.78
+6.0	0	.00	98.78
+6.5	0	.00	98.78
+7.0	1	.31	99.08
+7.5	2	.61	99.69
+8.0	1	.31	100.00

REMAINDER EMPTY

```
125 - 10 -   190371 - 01063074 JOBSHOP

VERTAALTIJD       102
EXECUTIETIJD      416

AANTAL OVER STROOM 1 GELEZEN GETALLEN          28
LAATST OVER STROOM 1 GELEZEN GETAL +.200000000000E+  0

AANTAL OVER STROOM 2 GELEZEN GETALLEN           0
```