2002:DS24

EXAMENSARBETE

Vägen till COBOL
Programmering med engelskliknande språkelement

Monica Vilhelmsdotter

2002-09-11

Högskolan Trollhättan/Uddevalla
Institutionen för informatik och matematik
Box 957, 461 29 Trollhättan
Tel: 0520-47 53 30 Fax: 0520-47 53 99

EXAMENSARBETE

Vägen till COBOL
Programmering med engelskliknande språkelement

Sammanfattning

Programmeringsspråket COmmon Business-Oriented Language (COBOL) skapades 1959. Det har engelskliknande språkelement, så som också dess föregångare FLOW-MATIC har. Avsikten med studien är att återge förhistorien till och arbetet i Committee on Data Systems Languages (CODASYL) med COBOL 60, på ett historiskt och vetenskapligt korrekt sätt.

Studiens mål är att praktiskt förstå inriktningens teori, metod, resultat och tillämpning. Att praktiskt visa övergången från användning av maskinkällkod till en tidig kompilator och därifrån till mer sofistikerade programmeringsspråk. Arbetet är en litteraturstudie. Det baseras helt på litteratursökning och personliga besök på institutioner, samt egna arbeten.

Varför pratar datorer engelska? Frågan besvaras genom att gå tillbaka till en tidig dator, The Analytical Enginge från 1834. För att därifrån göra ett hopp drygt 100 år fram i tiden. Fram till programmering i maskinkällkod av den första i en serie av Harvard Mark ifrån 1939. För att därefter fortsätta med en tidig kompilator beskriven 1952 för den första av flera generationers UNIVersal Automatic Computer (UNIVAC) från 1951 till användningen av B-0/FLOW-MATIC från 1955. För att slutligen komma fram till det maskinoberoende språket COBOL 60.

Programmering med engelskliknande språkelement användes som en väg, för att utöka kretsen av programmerare.

Nyckelord: COBOL, B-0, FLOW-MATIC, FACT, Commercial Translator, AIMACO, CODASYL, Charles Babbage, Augusta Ada Lovelace, Luigi Federico Menabrea, Alan Turing, Dick Clipping, Howard Aiken, Roy Goldfinger, Jack Jones, Grace Hopper, Jean Sammet, The Analytical Engine, The Difference Engine, Harvard Mark I, UNIVAC I, utveckling av programmeringsspråk.

Utgivare: Högskolan Trollhättan/Uddevalla
Institutionen för informatik och matematik
Box 957, 461 29 Trollhättan
Tel: 0520-47 53 30 Fax: 0520-47 53 99

Författare: Monica Vilhelmsdotter

Examinator: Universitetslektor Associerad Professor Stefan Mankefors

Handledare: Doktorand Linn Gustavsson

Poäng: 10 **Nivå:** C

Huvudämne: Datavetenskap **Inriktning:**

Språk: Svenska **Nummer**: 2002:DS24 **Datum**: 2002-09-11

DISSERTATION

The Road to COBOL
Programming with English-like Notation

Abstract

The programming language COmmon Business-Oriented Language (COBOL) was created in 1959. It has English-like notation like its forerunner FLOW-MATIC. The intention with the study is to render the prehistory to and the work within the Committee on Data Systems Languages (CODASYL) with COBOL 60 in a historical and scientifically correct way.

The purpose of the study is to practically understand the aim and direction of the theory, method, result and application. To practical show the link between the usage of source code and early compilers to more sophisticated programming languages. The work is a literary survey. Which is based entirely upon documentary research, personal visits to institutions and my own pieces of written work.

Why do computers speak English? The answer is given by reverting to an early computer, The Analytical Engine of 1834. From there a jump is made fully 100 years ahead in time, to programming in source code of the first one of several Harvard Mark of 1939. In 1952 an early compiler were described and written for the first in line of several generations of UNIVersal Automatic Computer (UNIVAC) to the use of FLOW-MATIC in 1955. At last the machine independent language COBOL 60 is reached.

Programming with English-like notation was used as one way, to increase the sphere of programmers.

Keywords: COBOL, B-0, FLOW-MATIC, FACT, Commercial Translator, AIMACO, CODASYL, Charles Babbage, Augusta Ada Lovelace, Luigi Federico Menabrea, Alan Turing, Dick Clipping, Howard Aiken, Roy Goldfinger, Jack Jones, Grace Hopper, Jean Sammet, The Analytical Engine, The Difference Engine, Harvard Mark I, UNIVAC I, programming language development.

Publisher: University of Trollhättan/Uddevalla
Department of informatics and mathematics
Box 957, S-461 29 Trollhättan, SWEDEN
Phone: + 46 520 47 53 30 Fax: + 46 520 47 53 99

Author: Monica Vilhelmsdotter

Examiner: Senior Lecturer Associated Professor Stefan Mankefors

Advisor: Candidate for the Doctorate Linn Gustavsson

Subject: Applied Computer Science

Language: Swedish **Number**: 2002:DS24 **Date**: 11 September, 2002

Innehållsförteckning

Bilageförteckning

1 Inledning

I min strävan att nå filosofie kandidatexamen i datavetenskap har jag sökt kunskap om bland annat programmering (Vilhelmsdotter, 2002 b). Man kan idag välja bland många olika typer av programmeringsspråk. Språken är utvecklade för att stödja olika processer. Det finns nu språk som är plattformsoberoende, men vägen dit har varit lång. Strävan efter att samordna kommunikationen mellan människa och datamaskin med hjälp av programmering är således inte ny. Hur börjar det egentligen? Många eller näst intill alla av dagens programmeringsspråk använder engelskliknande språkelement. Varför är det så och vad föranledde detta? Funderar man över språket, leder det också tankarna till att fundera över hur vi programmerar. Tidigare användes mest maskinkällkod medan vi idag kompilerar nästan allt. Hur ser vägen ut från att program skrivits i maskinkällkod till att skriva kompileringsbara program med lättläst källkod bestående av vanliga engelska ord? Jag började med att säga att dagens språk kan vara plattformsoberoende, men går det egentligen att göra dem maskinoberoende, fria och gemensamma?

I uppsatsen kommer jag först att ta upp programmering av en tidig dator och sedan en maskinkällkodsprogrammerad sådan. För att till sist beskriva ett kompilatorprogrammerat system och introducera programmering med engelskliknande språkelement. Studien startar förutsättningslöst, för arbeta sig fram till hållbara svar på frågeställningen (ibid).

2 Bakgrund

Idag finns en dator i de flesta svenska hem (ibid), men så har det inte alltid varit. Redan långt tillbaka i den förra delen av 1800-talet försökte Charles Baggage skapa en mekanisk ångdriven dator. Han gav den namnet The Analytical Engine. Den hade hålkortsstyrning liknande Jacquardmaskinens från samma tid.

Beräkningsmaskiner behövdes för en mängd uppgifter i den expansiva industriella miljön vid denna tid. Sjöfarten behövde tillförlitliga nautiska tabeller, livförsäkringsbolagen statistiska beräkningar över förväntad livslängd, bankerna räntetabeller o s v. Edvard och Georg Scheutz utvecklade tillsammans en kommersiell differensmaskin ur tankgångarna från Babbages The Difference Engine. Beräkningsmaskiner utvecklades vidare till att drivas elektromekaniskt via reläteknik under 1900-talets första tre årtionden.

Fortfarande under 1940- och 1950-talet, före den integrerade kretsens tillkomst och transistorteknikens intåg i datorerna, var de väldiga värmestrålande rörbestyckade kolosser. De fyllde hela rum och deras ännu mekaniska kringutrustning slamrade och förde oväsen (ibid).

Först under 1950-talet utvecklades kompilatorer för kommunikation med datorernas hårdvara. Före den tiden skedde programmering i maskinkällkod, vilken var helt specifik för den enskilda datorn som den användes på.

Under senare hälften av 1959 samarbetade dåtidens tillverkare av stordatorer och försvarsmakten i Committee on Data Systems Languages (CODASYL) för att åstadkomma just detta. De ställdes inför följande problemställning, ”how can the language be designed to include particularities of each machine and still remain universal” skriver Jean Sammet (1978, s 137) som satt med i kommittén.

3 Problemställning och syfte

Utvecklingen gick från att använda maskinkällkod till användning av kompilatorer under 1950-talet, men varför skapades programmeringsspråk som använde sig av engelska? Hur gick utvecklarna av programmeringsspråk tillväga rent praktiskt? Vad var deras visionen? På vika grunder har de olika valen gjorts under resans gång? Avsikten är att belysa och ge en inblick i ett avgörande skede under en fruktbar tidsperiod i utvecklingen

Arbetets strävan är att genom samordning av en mängd uppgifter ur många olika källor nå fram till relevanta slutsatser på dessa frågor (Vilhelmsdotter, 2002 b). Någon färdig hypotes finns inte utan primärdokumenten får tillhandahålla sin tids synsätt på utvecklingen av programmeringsspråk.

Programmeringsspråket COmmon Business-Oriented Language (COBOL) skapades 1959. Det har engelskliknande språkelement, så som också dess föregångare FLOW-MATIC har. Nästa steg i uppsatsen är därför att titta på detta. Varför skedde denna utveckling under 1950-talets senare del? Arbetet syftar till att visa på de bakomliggande orsakerna. Studien försöker att sammantaget fastställa de mest sannolika slutsatserna. Avsikten är att med studien återge förhistorien till och utvecklingsarbetet i CODASYL av programmeringsspråket COBOL 60, på ett historiskt och vetenskapligt korrekt sätt (ibid).

4 Avgränsningar

Programmeringsspråket COBOL behandlas endast i tiden för dess uppkomst, fram till COBOL 60 och i någon mån för COBOL 61. Språket beskrivs enbart med utgångspunkt från CODASYL:s arbete med att diskutera sig fram till en specifikation för grundsymboler och ingående ord. Samtliga övriga omnämnda programmeringsspråk, förutom FLOW-MATIC behandlas endast m a p deras relation till COBOL. Tidsorienteringen är historisk med tyngdpunkten på den senare delen av 1950-talet (ibid).

Omfattningen av arbetet är selektivt och täcker enbart övergripande insatser, för att skapa programmeringsspråk med engelskliknande språkelement (ibid). Studien inriktar sig enbart på de engelskliknande språkelementens syntax. Arbetet behandlar inte den semantiska tolkningen mellan kompilator och hårdvara. Inte heller utförs någon pragmatisk undersökning om relationerna mellan de språkliga uttrycken och deras användare. Vidare avser studien inte programmeringsspråks avlusning, utprovning, dokumentation eller underhåll av detsamma. All hårdvara och kringutrustning utlämnas, likaså databehandling internt och externt. Inte heller det hierarkiska upplägget, hur poster ordnas på olika nivåer berörs.

5 Teoretisk ram och metod

För mitt arbete har jag inte sökt något material från forskningsvärldens rapporter eller avhandlingar. Många små delar från olika typer av källor har samverkar till studiens resultat. Undersökningsenheterna är de engelskspråkliknade programmeringsspråkens framväxt, variablerna de frågor som studien ställer. Variabelvärden är de svar som återfinns i källmaterialet utifrån frågeställningen (ibid). Studien har gjort ett förbehållslös avstamp och kunskapsprocessen har fått utvecklas under arbetets gång.

Studiens mål är att praktiskt förstå inriktningens teori, metod, resultat och tillämpning. Att praktiskt visa övergången från användning av maskinkällkod till tidiga kompilatorer och därifrån till mer sofistikerade programmeringsspråk. Avsikten är att göra en explorativ litteraturstudie. I form av en hard case study gränsande till hybridforskningens område (ibid). Det går inte att ingripa i ett historiskt skede, för att förändra utvecklingen av programmeringsspråk i gången tid. Men det går an att förklara hur programmering av även tidiga datorer skedde. De som föregick kompilatorernas intåg under 1950-talets första år. För att därifrån återge affärsprogrammeringsspråkens sammansmält-

ning i COBOL 60. Till viss del återges CODASYL:s arbete med COBOL:s syntax. Koden beskrivs i några fall nedbruten till de enskilda orden och hur de samverkar med varandra. Detta för att skapa förståelse för kommitténs arbete och som ett försök att tolka deras avsikt med hur de valt att utforma COBOL. Studien har ett neutralt perspektiv och medför ingen förändring utan full förståelse eftersträvas (ibid).

Studien använder den kvalitativa, induktiva upptäcktens väg som metod. Återgivning av sammanhang och struktur eftersträvas. Likaså en helhetsbild av tidens skeende med avseende på problemformuleringen (ibid).

Arbetets validitet går inte att fastställa. De funna uppgifterna har olika grad av giltighet med avseende på frågeställningen. Källmaterialet som finns för handen ger inte svar som passar som hand i handske. Det som återfinns är troligtvis framställt från en annan utgångspunkt och för andra syften. Det finns ingen möjlighet att säkerställa vad som varit de olika ingående programmeringsspråksutvecklarnas ursprungliga avsikter. Inte heller att återskapa den ursprungliga processen. De funna sambanden redovisas (ibid).

Reliabiliteten är säkrad, för alla använda källor redovisas extensivt. Däremot är det inte så att studien är fokuserad, i den mening att den ser till den statistiska representationen. Resultatet av studien sammanför en mängd uppgifter från olika källor och presenteras med kronologisk och konstruktiv disposition. Genom att argumentera för och emot framtolkas relativt säkra svar. Av dessa har utformas en teori om hur utvecklingen kan ha framskridit (ibid).

6 Material

Arbetet baseras helt på litteratursökning och personliga besök på institutioner samt egna arbeten. Avsikten har varit att främst använda primärdokument. Så som tidskriftsartiklar, hela böcker, enstaka bokkapitel, artiklar från konferenser och annat publicerat material. Även opublicerat material från min egen hand används i någon omfattning. Datainsamlingen har varit förutsättningslös (ibid).

Allt skrivet material har bearbetats var för sig och sammansmälts till en helhetsbild. Diagram och tabeller har utarbetats för att återge programmeringsgången för de tidiga datorerna. För 1950-talets affärsprogrammeringsspråk ges programmeringsexempel

som kodexempel, såväl som i diagramform. Jämförelser av de olika språkens syntax sinsemellan görs också.

För sökning av material har de databaser använts som är tillgängliga från Högskolan i Trollhättan/Uddevallas (HTU) bibliotek. Artiklar från främst tidskrifterna Institute of Electrical and Electronics Engineers (IEEE) Annals of the History of Computing och IEEE Electrical Engineering har använts. Konferensartiklar från Association for Computing Machinery (ACM) sammankomster 1952 och 1978 har återfunnits. Böcker om och tillämpning av tidiga affärsprogrammeringsspråk har eftersökts och funnits. Även några av världens största bokhandlar, så som Blackwells, Foyles och Waterstones i London har besökts. Så också de få antikvariat som finns kvar på Charing Cross Road, London i jakten på äldre litteratur. Även tillgång till British Librarys forskningsbibliotek har erhållits på plats i London. Science Museum på samma ort besöktes också, för att samla in material till detta arbete. World Wide Web har utnyttjas flitigt för att spåra användbara källor (ibid).

7 Förhistorien till och utvecklingen av COBOL

Det finns mycket skrivet om möjligheterna att använda ett helt naturligt språk som programmeringsspråk. Men ingen verkar ha kommit till skott med, att verkligen implementera ett sådant. Jag kommer nu att i kronologisk ordning försöka beskriva utvecklingen och tänkandet genom att göra sporadiska nedslag i historien från ca 1830 och fram till 1960-talet.

Sammet (1969, s 57) skriver om möjligheten att skapa programmeringsspråk utifrån vilket naturligt språk som helst:

> If there is ever to be any hope of allowing users to define their own artificial [sic!] languages, it will most likely occur through the use of formal methods of language description and processors which can accept these definitions and either translate them into running code or interpret them to produce answers directly.

Hon beskriver vidare på s 731 två möjliga vägar att gå:

> One is the approach which can be called *bottom up*, in which artificial languages are developed to become increasingly close to natural language; the opposing, or *top down* view is that concentrating on understanding English [any natural language] grammar and developing generative an recognition procedures for increasingly large subsets of it.

Ändå anser hon redan på s 716 att möjligheten att framgångsrikt använda ett naturligt språk för annat än rena frågespråk är oöverstigliga. För även om kompilatorn förstår att analysera satslösningen, skall den också kunna skilja ut kommandon, variabler, kontrollstrukturer och formler uttryckta i ord. Hon skriver:

> Even if we have the mechanism for parsing English [any natural language] sentences and understanding what to do with them, the fact that there might be several hundred of these in a single program, together with a large number of data variables, control transfer statements, complex calculations, etc., presents a problem which transcends the mere syntactic and semantic analysis that we already know is a major difficulty.

1950-talets pionjärer: Clipping, Goldfinger, Hopper, Jones och Sammet själv såg inte så på saken vid den tiden. De var de drivande personerna när det gällde att skapa programmeringsspråk med engelskliknade språkelement. Främst Grace Hopper var en föregångare när det gällde att skriva källkod på ett engelskliknande sätt. Detta som ett sätt att få användare i affärsvärlden att acceptera programmering överhuvudtaget (Sammet, 1969).

Varför pratar datorer engelska? Frågan besvaras genom att gå tillbaka en tidig dator, The Analytical Enginge från 1834 (Vilhelmsdotter, 2002 a). Därifrån görs ett hopp drygt 100 år fram i tiden, till programmering i maskinkällkod av Harvard Mark I från 1939 (Aiken och Hopper, 1946). Det var den första i serien av flera maskiner med samma namn. För att därefter fortsätta med en tidig kompilator beskriven 1952 för den första av flera generationers UNIVersal Automatic Computer (UNIVAC) från 1951 (Hopper, 1952) till användningen av B-0/FLOW-MATIC från 1955 (Sammet, 1991). För att slutligen komma fram till det maskinoberoende språket COBOL 60 (Sammet, 1978). Till att börjar med beskrivs en av de första maskinerna.

7.1 En tidig dator

Efter framgångarna med The Difference Engine presenterar Charles Babbage 1834 The Analytical Engine (Vilhelmsdotter, 2002 a).

Före datorernas intåg fick alla beräkningar skötas av mänskliga s k computers. Nu introduceras maskinen som är avsedd att användas för att snabba upp och göra felfria beräkningar för matematiska tabeller. Vilka kräver noggrannhet och upprepning. Den

behärskar de fyra räknesätten (ibid). Hårdvarumässigt består den av, källa Vilhelmsdotter (2002 a):

- inenhet – hålkortsläsare för siffror, operationer och variabler
- centralenhet – the Mill
- minne – the Store
- utenhet – hålkortsstans eller tryckande enhet

7.1.1 Programmering av The Analytical Engine

Charles Baggage skriver 1837 ett manuskript som han ger titeln *The Mathematical Powers of Calculating Engine.* Det publiceras aldrig då, men återges i Randell (1982 b). Babbage beskriver där på ett utförligt sätt maskinens funktion och hur olika datavärden flyttas mellan de ingående enheterna. Programmeringen tillåter nästlade loopar och kan göra villkorade hopp (Vilhelmsdotter, 2002 a) även kallat branching. När problemet är formulerat stansas hålkort för ingående variabler och deras värden (Randell, 1982 b). I montern som innehåller delar från The Analytical Engine på Science Museum, London ligger provexemplar på variabel- och startvärdekort. De är av olika storlek. För de beräkningsoperationer som skall utföras väljs erforderliga hålkort ut (ibid). De har olika färg för de fyra räknesätten, källa Randell (1982 b):

- addition – vitt
- subtraktion – gult
- multiplikation – blått
- division – grönt

Använder sig beräkningarna av konstanter ställs dessa in manuellt på minnesenheten (Randell, 1982 c). Den håller reda på de olika korten och de inställda siffrorna. Deras värden matas in i centralenheten för beräkning. Efter att beräkningen gjorts förs delresultaten tillbaka ner till minnet för mellanlagring. Dessa mellanlagrade värden kan i sin tur utgöra indata i ett senare skede av beräkningsoperationen för samma problem. Resultatet stansas ut på hålkort eller trycks i tabellform på papper. En kommitté från British Association besöker Babbage 1878 för att studera hans maskin. De upprättar en rapport som publiceras året efter. Den återges i faksimil i Randell (1982 c), funktionsskiss över The Analytical Engine visas i fig 7.1 i bil 1, källor Randell (1982 b, c).

7.1.2 Matematiker beskriver The Analytical Engine

I Turin 1840 håller Babbage en föreläsning över sin nya skapelse. Närvarande där är Luigi Federico Menabrea (Vilhelmsdotter, 2002 a). Han beskriver senare dess arbetssätt i artikeln Notions sur la Machine Analytique de M. Charles Babbage utgiven 1842 (Randell, 1982 g). Menabrea skriver där om hålkorten (Vilhelmsdotter, 2002 a), i engelsk översättning av A. A. L. citat efter Kim och Toole (1999): ”The cards are merely a translation of algebraical formulae, or, to express it better, another form of analytical notation.”

Den blygsamma översätterskans initialer skall utläsas Augusta Ada Lovelace (Randell, 1982 g). När artikeln publiceras i Storbritannien 1843 lägger hon till egna anteckningar. Hon förklarar och åskådliggör maskinens funktion. Hon förutser dessutom fler användningsområden för tekniken såsom artificiell intelligens, datormusik förutom matematiska tillämpningar (Vilhelmsdotter, 2002 a). Maskinens arbete beskrivs som, citat efter Toole (1996, s 9) : ”The Analytical Engine weaves algebraical patterns just as the Jacquard-loom weaves flowers and leaves.”

När Alan Turing föreläser om artificiell intelligens ca 100 år senare (Vilhelmsdotter, 2002 a), använder han sig av ett annat av hennes citat efter Toole (1996, s 10): ”The Analytical Engine has no pretensions whatever to originate anything. It can do whatever we know how to order it to perform.” Vilket än idag är en nyckel med avseende på hur datorer fungerar.

7.1.3 The Analytical Engines epilog

Maskinen färdigställdes aldrig, en ångdriven koloss om ca sex meters längd hade det blivit. Charles Babbage började så smått att bygga på central- och den tryckande utenheten. Den färdigställs efter hans död av sonen Henry, som 1910 sammanställer en kortfattad beskrivning av Babbage’s Analytical Engine. Den publiceras samma år i tidskriften *Monthly Notices of the Royal Astronomical Soc.* Randell (1982 a) återger den i faksimil. Idag återfinns The Mill som utställningsföremål på datoravdelningen på Science Museum, London. Tillsammans med delar av The Difference Engine och en prototyp av Edvard och Georg Scheutz kommersiellt använda differensmaskin.

7.2 Maskinspråkberoende källkodsprogrammering

Efter det att en tidig dator beskrivits, kanske den första görs ett hopp fram till mellankrigstidens troligen mest kända beräkningsmaskin. Vid The Computation Laboratory of Harvard University utvecklades 1939 The Automatic Sequence Controlled Calculator. Den maskinen kallas idag Harvard Mark I. Tillsammans med Howard Aiken skriver Grace Hopper artikelserien The Automatic Sequence Controlled Calculator del I, II och III. Artiklarna publiceras under 1946 av IEEE:s tidskrift *Electrical Engineering*. De återges i faksimil av Randell (1982 d, e, f).

Behovet av maskinen beskrivs enligt följande citat efter Randall (1982 d, s 205):

> The intensive development of mathematical and physical sciences in recent years has included the definition of many new and useful functions, nearly all of which are defined by infinite series or other infinite processes. Most of these are tabulated inadequately and their application to scientific problems is retarded thereby.

Huvudprogrammeringen utgörs av en hålremsa där informationen är indelad i tre fält A, B och C. Fält A representerar överföring av utdatavärden mellan maskinens olika enheter via bussen. Fält B representerar indatavärden från maskinens enheter via bussen. Fält C representerar vilken operation som skall utföras på siffervärdena i fält A och B (Randell, 1982 d).

Maskinen kan utföra fem olika operationer, förutom de fyra räknesätten kan den dessutom lagra referenser till tidigare utförda beräkningar (ibid). Funktioner lagras också på hålremsa och läses in av huvudenheten vid behov (Randell, 1982 e). Samma sak gäller för slumpvärden och konstanter (Randell, 1982 f). De sistnämnda kan även ställas in på någon av de 60 omkopplarna. Antalet decimaler som skall användas i resultatet av en multiplikation ställs in med hjälp av proppar på ett kopplingsbord. Också hålkort kan användas för att lagra data på eller läsa in data från. Programmet startas upp med en särskild starthålremsa.

Matematikern är den som skriver program för maskinen. Programmeringen genomförs med hjälp av beräkningar och genom användande av tabeller, formelsamling och manual. För att köra ett program har operatören skrivna instruktioner till sin hjälp. Där står steg för steg hur maskinen skall ställas in, startas upp och köras (ibid). Funktionsskiss för programmering av Harvard Mark I visas i fig 7.3 i bil 2, källor Randell (1982 d, e, f). Harvard

Mark fortsätter att utvecklas i flera generationer och den sista tas i drift så sent som 1952 skriver O'Connor och Robertson (1999). Delar av den finns bevarad hos Smithsonian Museum of American History Science Center, Washington D. C. (Ferguson, 1998) och vid Computer History Museum, Mountain View, Kalifornien.

7.3 Maskinspråksberoende kompilator

Källkodsprogrammering är nu etablerad och i det tidiga 1950-talet experimenteras det med kompilatorer. Vid Eckert-Mauchly Computer Corporation (EMCC) utvecklas UNIVAC-datorerna för affärsdatatillämpningar. Vid en av ACM:s allra första konferenser deltar Grace Hopper i egenskap av dess chefsprogrammerare (Hopper, 1952). Hon beskriver där hur en dator, UNIVAC I utbildas. T ex skall kommandon kunna ges, exv "'go to operation k', and the compiling routine does the rest" skriver Hopper (1952, s 246). Redan i sitt inledningsanförande på s 244 fäller hon det sedermera berömda uttrycket: "The programmer may return to being a mathematician." Genom användning av kompilator skall matematiker frigöras från programmeringssysslan. Detta för att i stället helhjärtat kunna ägna sig åt matematisk problemlösning (Hopper, 1952).

7.3.1 Programmering av UNIVAC I

UNIVAC lär sig att använda subrutiner av olika slag. Dessa lagras av maskinen för att vid senare tillfälle när behovet påkallas hämtas och används. Med flera subrutiner tillsammans konstrueras hela program. Hela bibliotek med underkataloger för olika grupper skapas av subrutiner med s k "call-numbers" (Hopper, 1952). De inleds med en bokstav som kan antyda dess uppgift skriver Hopper (1952, s 247).

a	arithmetic
b	transfer of data
c	counters
h	hyperbolic functions
i	input routines
l	logarithmic functions
o	output routines
p	polynomials
r	roots and fractional exponents
t	trigonometric functions
u	control transfers
w	storage routines
x	exponential functions
y	editing routines

För att undvika felkällor skrivs subrutinerna i förväg och provkörs. Programmeraren ses som en integrerad del av maskinen. Informationen översätts till ett för maskinen passande språk, någon maskinkod behöver inte längre användas (Hopper, 1952).

7.3.2 Kompilering av UNIVAC

Kompilatorn delar in rutinerna i typ A och typ B, följ funktionsskissen för programmering av UNIVAC I i fig 7.4 i bil 3, källa Hopper (1952).

Kompileringen av typ A-rutiner sköter om alla de tjänster som är nödvändiga för att färdigställa det slutliga programmet. De ansvarar för att rätt subrutin kallas in. Läser av call-numbers, hämtar sedan rätt rutin och indatavärden från rätt band i bandbiblioteket. Vidare allokeras temporära förvaringsplatser för beräknande operationer och program. Slutligen sköter den kontrollen av all dataöverföring till och från alla in- och utgångar (ibid).

Kompilering av typ B tillhandahåller formler som den vidarebefordrar till en typ A-rutin (ibid).

UNIVAC I beskrivs som en andraårsstudent på matematikerprogrammet som snart är redo för sitt examensarbete. Ytterligare utveckling av fler och mer avancerade rutiner förutspås. Framförallt för tillämpning inom affärsvärlden (ibid).

7.4 FLOW-MATIC – det naturliga språket

Det finns en uttalad avsikt med att använda verb som inledande programkod i en sats, så som FLOW-MATIC gör. Det var att dessa inleder de flesta s k naturliga språks meningsbyggnad i imperativ form. Tanken bakom FLOW-MATIC är att koden skulle skrivas i programmerarens eget modersmål enligt Sammet (1978). Generellt om användandet av naturliga språk kan sägas att det förmodas att användaren är en icke professionell programmerare, men skicklig i sin yrkesverksamhet. P g a det förväntas de att lägga större vikt på själva problemlösningen. De anses inte ha den erfarne programmerarens behov av att studera innehållet i datorns minnen och register fortsätter Sammet (1969).

Grace Hopper och arbetsgruppen runt henne har 1955 de preliminära specifikationerna klara för FLOW-MATIC. Det nya programmeringsspråket skall vara väl lämpat för datoriserad affärsdataanvändning, men fortfarande vara enkelt att använda (ibid).

Programmeringsspråket FLOW-MATIC grundlägger mönstret för hur ett affärsprogrammeringsspråk skall se ut. Det etablerar konceptet med ett engelskliknande naturligt språk. Ord används både för operationerna som utförs och för de dataelement som de utförs på. Ordlängden begränsas till 12 tecken med anledning av begränsningar i maskinvaran, som kompilatorn för FLOW-MATIC implementeras på (Sammet, 1978).

FLOW-MATIC (även kallad B-0) är också grunden för ytterligare programmeringsspråk med engelskspråkliknade element som AIr MAterial COmmand (AIMACO) och Commercial Translator samt Fully Automatic Compiling Technique (FACT) (Sammet, 1991). De kommer alla att påverka COBOL 60, förutom FACT som först slår igenom senare (Sammet, 1978).

7.5 COBOL – det maskinoberoende språket

Behovet av ett mer lättbegripligt programmeringsspråk uttrycktes bl a på följande sätt: ”We need to broaden the base of those who can state problems to computers. The need is for a programming language that is easier to use, even if somewhat less powerful” (Sammet, 1978, s 124).

Projektet för att skapa ett gemensamt programmeringsspråk för affärsvärlden startade maj 1959. Det sponsrades av Försvarsdepartementet därför att hårdvarutillverkarna skulle vara neutrala (Sammet, 1978).

De olika företagen som var aktiva i datorbranschen uppmuntrades att själva också delta i utvecklingsarbetet. Deltog gjorde också slutanvändare och konsulter. Vid uppstartsträffen diskuterade mötet de önskvärda egenskaperna hos ett nytt programmeringsspråk skriver Sammet (1978). Flertalet förordade användning av enkel engelska, men några förslog att matematiska symboler skulle användas i stället. Ett litet antal deltagare menade att ett problemorienterat programmeringsspråk inte var något att satsa på. Användning av engelska ansågs inte vara inte någon universallösning. För det gick inte att manipulera på samma sätt som algebraiska uttryck. Ett programmeringsspråk som var lättare att använda även om det var mindre kraftfullt, var också ett önskemål. Likaså att bredda basen för rekrytering av fler programmerare. Vid utveckling av det nya språket skulle man inte hindras av de förutfattade meningar som fanns p g a problemen med de dåvarande kompilatorerna hävdade mötesdeltagarna (ibid). Sammet (1969) förtydligar att kompilatorer vid den här tiden var långsamma och producerade undermålig objektkod.

CODASYL:s uppgift under de första tre månaderna var, skriver Sammet (1978) att utforska de tre befintliga affärsprogrammeringsspråken. De var FLOW-MATIC skapat av Remington Rand Company UNIVAC Division, 1955-8 och AIMACO skapat av Grace Hopper och Jack Jones vid Amerikanska Flygvapnet, 1959 samt Commercial Translator skapat av IBM, 1959. De två förstnämnda var i bruk, medan det sistnämnda fortfarande var under utveckling. I den första COBOL-standarden från 1960 finns AIMACO omnämnt bland de språk som bidragit till COBOL. Det blev aldrig copyrightskyddat och slutade användas i tidigt 60-tal.

Andra programmeringsspråk som det också kastades ett getöga på var, källa Sammet (1978):

- FORTRAN
- Autocoder
- SURGE
- RCA 501 Assembler
- Report Generator
- APG-1

CODASYL kände till ALGOrithmic Language (ALGOL) 58, men inte i någon högre grad. Förutom ordföranden i en av arbetsgrupperna, för han satt med i dess kommitté. Debatten kring vilka språk som skulle studeras mer ingående påverkades inte av ALGOL 58 (ibid).

Efter tre månader beskrivs det nya språkets uppgifter. Det skall t ex tillhandahålla en ordnad och fast bas av termer. Vidare skall enkelheten skall vara av sådant slag att även novisen till programmerare skall kunna skriva samman sitt program. Det skall vara lättläst och svara mot ledningens behov att uttrycka sin programmeringsproblem. Vanligt affärsfolk skall kunna läsa det utan att ha några som helst programmeringskunskaper. System som tillhandahåller språket skall kunna tillämpa det. Programmeringen skall utföras på det för systemet mest praktiska sättet (Sammet, 1978). Det skall inte ge stöd för vetenskapsrelaterade frågeställningar som uppkommer i samband med planering av ekonomistyrningssystem (Sammet, 1969).

Vid mötets diskussion framkom, att en del tyckte att det hade varit för många kompromisser. Andra däremot menade att det nya språket var bättre än de som fanns tillgängliga. Några tyckte att språket var allt för komplext, andra menade att det var för enkelt (Sammet, 1978).

Vad man dock kunde enas om var, att det nya programmeringsspråket skulle ha tre olika typer av input (ibid). Beskrivningen av innehållet i de olika delarna är delvis hämtad från Lysgård (1966).

1. **Enviroment/Kringutrustning**
   Maskinberoende information som hämtas eller lämnas till eller från yttre enheter.

2. **Description/Data**
   Filers innehåll, posters hierarki, dataelements storlek, klass, skalfaktor, arbetsutrymme eller konstanter.

3. **Procedur**
   Instruktioner inleds i form av verb, med vilka man beordrar ett moment i procedurdelen att utföras. Dessa kan sättas samman till satser som opererar på de storheter som angivits i datadelen. Som beskrivning av processen för databehandlingen, exempelvis kan satserna sammanfogas till paragrafer och vidare till sektioner.

När väl hårdvarutillverkarna hade kommit en bit på väg med implementationen, blev de allt mer ovilliga att göra sena ändringar i COBOL:s specifikationer (Sammet, 1978).

Med hänsyn tagen till skillnader i kringutrustning var avsikten att ett program skrivet i COBOL skulle kunna exekveras på vilken annan dator som helst som hade en COBOL-kompilator implementerad. När så COBOL:s preliminära specifikationer offentliggjordes i februari 1960 beskrivs tillverkarnas ansvar enligt Sammet (1978 s 131) som: ”COBOL as recommended …would be accepted as the minimum that would be required from all computer manufacturers.” D v s den tillverkare som hävdar att hans maskin använder COBOL, han skall tillhandahålla en kompilator som behandlar vart element som är definierat i COBOL. I december 1960 kom de första kompilatorer som körde program skrivna i COBOL i bruk. De var framtagna av två olika hårdvarutillverkare och utgjorde beviset för att maskinoberoende hade uppnåtts (Sammet, 1978).

7.5.1 Design av COBOL

Jean Sammet (1969) framhåller att det som påverkar språkdesign mest är personlig tycke och smak, snarare än varken vetenskaplighet eller ekonomi, hon skriver på s 49: ”Language design is an art, not a science.”

COBOL:s mål var, källa Sammet (1978):

- Naturlighet
- Lätt att överföra till annan media
- Att vara strukturerat
- Lätt att implementera

När stridigheter uppstod kring hur syntaxen skulle se ut, slutade det oftast med att det som ansågs mest lättläst antogs. Dock inte i fallet med aritmetiska operatorer. För dess vidkommande bestämdes detta skrivsätt (ibid). Exemplen visar reserverade kommandoord understrukna och variabelnamn och dataelement i skrivs med små bokstäver enligt gängse modell för COBOL:s metaspråk för att beskriva objekten, källa Sammet (1978):

DIVIDE y INTO x

istället för det mer naturliga:

DIVIDE x BY y

Här fick läsbarheten ge vika för att vidmakthålla principen att ”having the receiving field as the last stated variable for the four arithmetic verbs” skriver Sammet (1978, s 139). D v s den sista variabeln skall vara det mottagande fältet vid aritmetiska operationer.

Språket var avsett att användas för affärsändamål. Därför ansågs det att inga matematiska funktioner eller reell aritmetik med flytande decimalkomma behövdes (Sammet, 1978). Utan enbart operatorer som behövdes för bokföring, avlöningssystem, lagerhantering osv.

Antal siffror för ett fält sattes till maximalt 18 stycken. Hos hårdvarutillverkarna fanns inte någon sådan implementation färdig. Därigenom hade ingen fördel framför den andre under utvecklingsprocessen (ibid).

Variabelnamn får innehålla maximalt 30 tecken, av dem skall minst ett tecken vara en bokstav. Det framfördes åsikter i kommittén om att tillåta helt numeriska variabelnamn. Detta med anledning av att då kunde artikelnummer användas som de var. Men risken för att förväxling skall ske och att variabelnamnet kan komma att betraktas som ett tal ansågs vara allt för stor (ibid).

CODASYL valde att begränsa COBOL:s teckenuppsättning till den som fanns för hålkort och skrivmaskiner (ibid). Sammet (1969) anger att hålkortstansarna klarade 47 alternativ 48 tecken. Lysegård (1966) anger antalet grundsymboler för COBOL 61 till 51 st. De är först och främst siffrorna 1–9 och bokstäverna A–Z. Det förefaller naturligt att använda + – X / som aritmetiska operatorer, samt att sätta in parenteser ( ) som interpunktion i beräkningar. Likaså används relationsoperatorerna = < > på sedvanligt sätt (ibid). Eftersom COBOL är amerikanskt är $-tecknet väl valt för att representera valuta. Punkt avslutar en mening precis som i vårt naturliga språk (ibid). Citationstecken används när meddelande skall skrivas ut till operatören, källa Lysegård (1966) t ex:

STOP "Paper jam"

Kommatecken används vid uppräkning, som vid vanligt språkbruk, men det kan också utelämnas och ersättas av blanksteg eller med ordet AND här exempel på fyra olika giltiga skrivsätt, källa Lysegård (1966):

ADD x y TO z

ADD x, y TO z

ADD x AND y TO z

ADD x, AND y TO z

Semikolon används godtyckligt för att skilja satser och bisatser åt, här två alternativa men ekvivalenta skrivsätt, källa Lysegård (1966):

ADD 1 TO x GO TO y.

ADD 1 TO x; GO TO y.

I vissa fall kunde såväl symboler som engelska ord användas (Sammet, 1978). T ex lika väl + – X / som det reserverade ordet COMPUTE följt av ADD, SUBTRACT, MULTIPLY, DIVIDE. Kommittén var av den bestämda uppfattning att kommandoraden skall inledas med ett verb. Därav användningen av det inledande ordet COMPUTE.

Språket utgörs av ett fåtal verb, med många tillvalsmöjligheter. När kommittén arbetade med att utarbeta syntaxen använde den metaspråk för att strukturera de olika valmöjligheterna. En sådan möjlighet är att använda s k nonsensord som utfyllnad för att

få en mer lättläst programmeringskod (ibid). Lysegård (1966) anger för COBOL 61 att det har 284 reserverade ord vilka visas i tab 7.1 i bil 4, källa Lysegård (1966, s 132 f).

För flödeskontroll, skriver Sammet (1978) finns det inte några nästlade loopar som i ALGOL 60, men väl en hierarki beskriven av Lysegård (1966) bestående av:

- **Statement/Satser**
  De inleds med ett kommando följt av den data som kommandot skall utföras på, exempel:

  ADD 1 TO x

- **Sentence/Mening**
  Består av en eller flera satser och avslutas med punkt följt av ett mellanslag, exempel:

  ADD 1 TO x GO TO y**.**

- **Paragraf**
  Dessa är namngivna och avslutas med punkt och mellanslag. Därefter kommer meningarna som hör till den paragrafen, ex:

  addera**.** ADD 1 TO x**.**
  GO TO y**.**

- **Sektion**
  Flera paragrafer kan underordnas i en sektion, ex:

  summering SECTION**.**
  addera**.** ADD 1 TO x**.**
  GO TO y**.**

Med hjälp av det reserverade ordet DEFINE kan nya reserverade ord skapas. Detta är ett av de tidigaste exemplen på användning av makroprogrammering i ett högnivåspråk. Det tas senare bort av CODASYL 68 på grundval av att ingen hårdvarutillverkare valt att implementera det (Sammet. 1978).

Det fanns två sätt att beskriva logiska uttryck med relationsoperatorer, källa Sammet (1978), antingen som:

x = 2 OR x = y OR x = z

eller

x = 2**,** y OR z

Vidare skapades en ännu mer kraftfull struktur än ALGOL 60:s **if-then-else-**sats, källa Sammet (1978), exempel:

IF x = y MOVE a TO b**;**
IF GREATER ADD a TO z**;**
OTHERWISE MOVE c TO d**.**

För ytterligare exempel på komplexa programsatser, se fig 7.5 i bil 5, källa: Sammet (1969).

Idag betraktas GO TO som kätteri vid flödeskontroll, men var vid den här tiden ett normalt skrivsätt. Några av hårdvarutillverkarna fruktade att COBOL:s arbete på filer skulle störa kringutrustningens samverkan med I/O-enheterna. Därför förorsakade enkla och självklara kommandoord som OPEN, CLOSE, READ, WRITE betydande debatt. (Sammet, 1978).

7.5.2 Andra programmeringsspråks påverkan på COBOL

Ett språks utveckling beror inte enbart på de som gör själva arbetet. Det påverkas i lika hög grad av den organisation som är beställare av det (ibid). Språket COBOL är kraftigt influerat av FLOW-MATIC, men Sammet (1991) framhåller att språkutvecklarna, inte kände till ALGOL. I sin stora sammanställning över COBOL:s tillblivelse framför Sammet (1978) ett annat budskap. Där beskriver hon hur olika språk togs i beaktande. Med avseende på deras olika för- och nackdelar. Tab 7.2 i bil 6 visar vilka ord som återanvänts från dessa språk i COBOL, källor Lysegård (1966) och Sammet (1969).

7.5.2.1 FLOW-MATIC

Programmeringsspråket FLOW-MATIC skapades av Grace Hopper och implementerades av Sperry Rand Company UNIVAC Division i deras stordatorer och användes för affärsmässigt bruk av flera stora företag från 1958. Detta programmeringsspråk utgjorde en naturlig grund för vidareutveckling (Sammet, 1978).

FLOW-MATIC:s bidrag till COBOL var användning av variabelnamn. Också att de reserverade orden, operanderna skrivs ut som hela ord t ex ADD, COMPARE. Dessutom att skilja beskrivningen av filinnehåll, posters hierarki, dataelement, arbetsutrymme och konstanter från procedurdelens operander. Det sistnämnda ställde de begrepp som gällt fram till dess på huvudet (ibid).

Ett av syftena med FLOW-MATIC var att det med lätthet skulle kunna skrivas om till andra språk. Därför inleddes vart uttryck med ett verb. Detta grundade sig på att i andra naturliga språk (än engelskan) inleds en befallande mening på detta sätt. Detta synsätt att utforma syntaxen på överfördes till COBOL, trots att det enbart utvecklades för att täcka den amerikanska affärsmarknadens behov (ibid). Tab 7.3 i bil 7 visar de reserverade orden i FLOW-MATIC, källa Sammet (1969).

7.5.2.2 Commercial Translator

I FLOW-MATIC:s kölvatten utvecklade Roy Goldfinger på IBM Commercial Translator 1959 (ibid). IBM strävade efter att överföra så mycket som möjligt av sin syntax till COBOL (Sammet, 1978), tab 7.4 i bil 7 visar de reserverade orden, källa Sammet (1969). Programmeringsspråket implementerades aldrig och hade därför heller inte några användare (Sammet, 1978).

Dess bidrag till COBOL var att villkor kan användas direkt av variabler. T ex IF MINOR i stället för IF EQUALS MINOR. Också att olika nivåer kan användas i programmeringen. Dittills användes monolitisk programmering i ett enda block. Det bidrog också med användning av villkorssatser typ IF…THEN (ibid).

Dessutom tillfördes användning av suffix. Det innebar att ett fält i en post kunde hämtas trots att samma namn förekom i flera olika filer. I praktiken gick det till så, att när datainnehållet i fältet skulle hämtas hängdes filnamnet på efter fältnamnet (ibid).

PICTURE-satsen överfördes i stort sett som den var till COBOL (ibid), exempel, källa Lysegård (1966):

namn PICTURE IS teckensträng

namn är ett grundelement, PICTURE ett reserverat ord, IS ett utfyllnadsord, som kan användas för att göra programkoden mer lättläst, strängen om maximalt 30 tecken beskriver grundelementens storlek och kategori samt antal decimaler (ibid).

7.5.2.3 FACT

Honeywell satt med i CODASYL, men utvecklade i hemlighet ett eget programmeringsspråk, FACT under ledning av Dick Clippinger. När det presenterades under 1959 ställde Honeywell krav på att COBOL skulle ta över FACT:s syntax (Sammet, 1978).

En av hörnstenarna för COBOL är att det skall vara oberoende av hårdvarutillverkare och gemensamt. När FACT lanserades 1960 ansågs det vara ett tekniskt överlägset, riktigt bra och avancerat språk. Dock hade det en akilleshäl, det var maskinberoende. Fr o m COBOL 61 finns det med som ett av de copyrightskyddade språk ur vilka COBOL skapats (ibid).

7.5.2.4 FORTRAN

Det var ett medvetet val att göra COBOL annorlunda från IBM:s FORTRAN. Det var visserligen en av CODASYL:s förelagda uppgifter att även titta på FORTRAN, för att se vad det kunde bidra till COBOL med. Men, skriver Sammet (1978) inom kommittén var det hos flera av ledamöterna (inklusive henne själv), starka negativa känslor gentemot IBM. Det bidrog delvis till att verbet PERFORM användes istället för FORTRAN:s DO. En annan var att de skiljde sig något åt i vad de utförde (ibid).

7.5.2.5 ALGOL 58

CODASYL hade verkligen både kännedom och kunskap om ALGOL 58 (ibid). Sammet är oense med sig själv om dess inflytande på COBOL. Hon skriver ”metalanguage description of ALGOL was not known” (Sammet, 1991, s 35). I en tidigare artikel skriver hon: “I do not believe IAL [later renamed ALGOL] had any influence on our deliberations” (Sammet, 1978, s 126). Några sidor längre fram i samma artikel skriver hon vidare att, “in fact, it …provided a significant influence on the choice of the COBOL character set” (Sammet, 1978, s 134).

I kringutrustningens teckenuppsättning som användes vid den här tiden saknades en del av de tecken som ALGOL 58 använde i sitt metaspråk, för att beskriva syntaxen. Kommittén slogs av hur opraktiskt det var att använda tecken som inte fanns. Den höll benhårt på att de tecken som skulle infogas i COBOL:s metaspråk skulle vara läs- och skrivbara med dåtidens kringutrustning (Sammet, 1978).

7.5.3 Epilog till CODASYL:s arbete

COBOL togs fram av CODASYL på sex månader. Sammet (1978, s 132) skriver: ”In those days all of us had yet to learn how long it takes to define and/or clarify a language!” Enligt Sammet (1991) är en tidsperiod om ett till tre år mer rimligt för att skapa ett nytt programmeringsspråk.

8 Resultat

COBOL 60 är en sammansmältning av de tre likartade programmeringsspråken FLOW-MATIC och AIMACO samt Commercial Translator. CODASYL:s uppgift var att ta det bästa från befintliga programmeringsspråk och baka ihop dem. Den utmaningen antogs och genomfördes, trots tjuv och rackarspel hårdvarutillverkarna emellan (Sammet, 1978). Jean Sammet (1969, s 324) gör följande reflektion: ”The question of whether people prefer to write EQUALS or EQUAL TO instead of = is one that requires solution by a psychologist rather than by programmers.”

Möjligheten till ett mångfacetterat skrivsätt vid kodning är en stor tillgång. På så sätt kan personliga preferenser tillgodoses och inga utbrytare behöver skapa egna dialekter. Ur ett i grunden, ett och samma programmeringsspråk. Vi har alla olika tycke och smak. Låt oss slå vakt om denna egenskap.

COBOL var enbart avsett att användas i USA skriver Sammet (1978). Därför var det inte aktuellt att göra någon omskrivning av det från engelskan till något annat språk. Vilket i och för sig hade varit enkelt att göra under utvecklingsarbetets gång (ibid).

Det ifrågasätts hur läsbarheten fungerar i t ex Sverige. För oss som använder COBOL med de reserverade orden på engelska tillsammans med svenska namn på variabler, filer, poster och deras element. Det primära målet för COBOL var enligt CODASYL att det skulle vara naturligt, i meningen ett levande språk (ibid). För mig förefaller ingenting vara mer onaturligt än den svengelska som ofta används i programmeringskod. Läsbarheten fungerar därför att jag har aldrig ens tänkt på möjligheten att det kunde vara annorlunda.

9 Diskussion

The Analytical Engines programmeringsmetod har ett säreget sätt att uttrycka problemlösning med algebra på. Genom olika färg och storlek på hålkorten skiljs de ingående elementen åt. Ett av syftena med att använda engelskliknande språkelement vid programmeringsspråkens utvecklingen på 1950-talet var att åstadkomma ett mer lättförståligt skrivsätt. Baggage visar redan, drygt 100 år tidigare på, att förståelse kan uppnås genom att använda färg och form istället.

Grace Hopper intog en helt central ställning för utvecklingen av att använda engelskliknande språkelement i programmerings-

språk. Jag inte kunnat få fram några uppgifter om hon har skrivit eller publicerat något i detta ämne. Trots detta anser jag, att jag har kunnat skapa förståelse för det nya sätt att programmera som växte fram under några händelserika år i slutet av 1950-talet.

Det hade varit intressant att utvidga frågeställningen och sätta in den i ett vidare perspektiv. Dels en genomgång av assemblerprogrammering med sin trestaviga kortkod, dels programmeringsspråk som använder exv symboler som skrivsätt. Detta får anstå till ett senare tillfälle. Tiden medger f n inte att även denna sidan av saken belyses, vilket jag beklagar.

10 Slutsats

Programmering med engelskliknande språkelement användes som en väg, för att utöka kretsen av programmerare.

Kanske återuppstår möjligheten till programmering i naturligt språk på ett just nu oförutsägbart sätt. I Första Moseboken, kapitel 11, vers 1 läser vi: ”Hela jorden hade samma språk och samma ord.”

Referenser

Computer History Museum. Artifacts.
[Elektronisk] tillgänglig:
< http://www.computerhistory.org/collections/artifacts/database/X00006_1975.tdb >
[hämtad och läst 2002-05-24]

Ferguson, C. (1998). Howard Aiken: Makin' a Computer Wonder.
[Elektronisk] tillgänglig:
< http://www.news.harvard.edu/net_news/5_27/story9.html >
[hämtad och läst 2002-05-24]

Hopper, G. (1952). The Education of a Computer.
Proceedings of Association for Computing Machinery 1952 Conference,
2-3 maj; Pittsburgh, USA., ss. 243-9.

Kim, E. E., Toole, B. A. (1999). Ada and the first computer.
Scientific American, vol. 280, no. 5, ss. 66-71.

Lysegård, A. (1966). *COBOL*.
Lund: Studentlitteratur.

O'Connor, J. J., Robertson, E. F. (1999) Howard Hathaway Aiken.
[Elektronisk] tillgänglig:
< http://www-history.mcs.st-andrews.ac.uk/history/Mathematicians/Aiken.html >
[hämtad och läst 2002-05-23]

Randell, B., (1982 a). Babbage's Analytical Engine.
The Origins of Digital Computers Selected Papers.
3:dje uppl. Berlin: Springer-Verlag.

Randell, B., (1982 b). On the Mathematical Powers of Calculating Engine.
The Origins of Digital Computers Selected Papers.
3:dje uppl. Berlin: Springer-Verlag.

Randell, B., (1982 c). Report of the Committee, consisting of Professor Cayley, Dr. Farr, Mr. J. W. L. Glaisher, Dr. Pole, Professor Fuller, Professor A. B. W. Kennedy, Professor Clifford and Mr. C. W. Merrifield, appointed to consider the advisability and to estimate the expense of constructing Mr. Babbage's Analytical Machine, and of printing tables by its means.
The Origins of Digital Computers Selected Papers.
3:dje uppl. Berlin: Springer-Verlag.

Randell, B., (1982 d). The Automatic Sequence Controlled Calculator del I.
The Origins of Digital Computers Selected Papers.
3:dje uppl. Berlin: Springer-Verlag.

Randell, B., (1982 e). The Automatic Sequence Controlled Calculator del II.
The Origins of Digital Computers Selected Papers.
3:dje uppl. Berlin: Springer-Verlag.

Randell, B., (1982 f). The Automatic Sequence Controlled Calculator del III.
The Origins of Digital Computers Selected Papers.
3:dje uppl. Berlin: Springer-Verlag.

Randell, B., (1982 g). *The Origins of Digital Computers Selected Papers*.
3:dje uppl. Berlin: Springer-Verlag.

Sammet, J. E. (1969).
Programming Languages: History and Fundamentals.
Englewood Cliffs: Prentice-Hall.

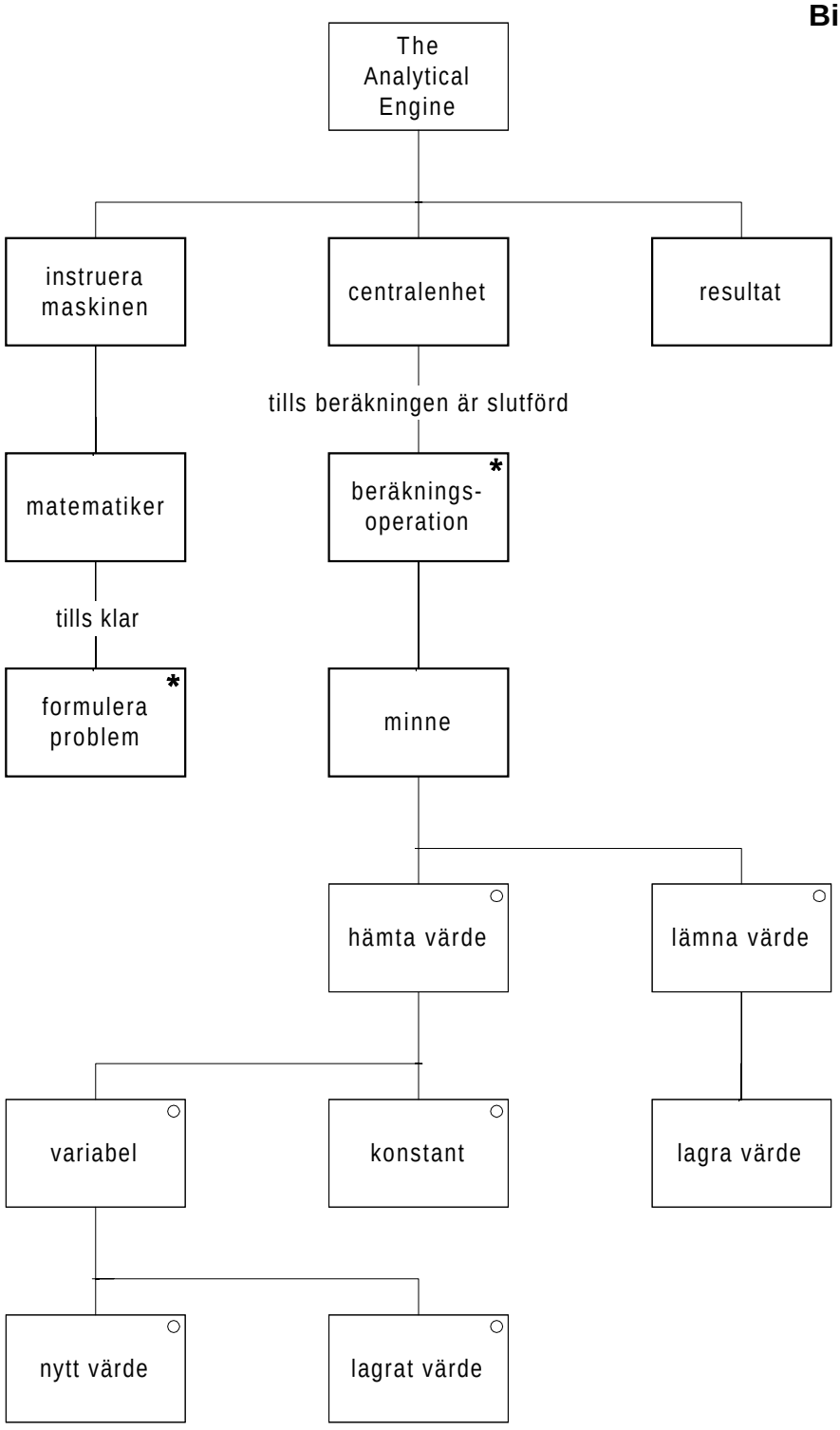
Sammet, J. E. (1978). The Early History of COBOL.
Proceedings of the ACM SIGPLAN History of Programming Languages June 1978 Conference, ss. 121-61.

Sammet, J. E. (1991). Some Approaches to, and Illustrations of, Programming Language History.
IEEE Annals of the History of Computing, vol. 13 no. 1, ss. 33-50.

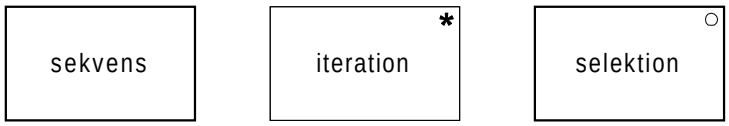
Toole, B. A. (1996).
Ada Byron, Lady Lovelace, An Analyst and Metaphysician.
IEEE Annals of the History of Computing, vol. 18, no. 3, ss. 4-12.

Vilhelmsdotter, M. (2002 a). *Ada Lovelace*. Opublicerat manuskript.
Trollhättan: Högskolan i Trollhättan/Uddevalla,
Institutionen för informatik och matematik.

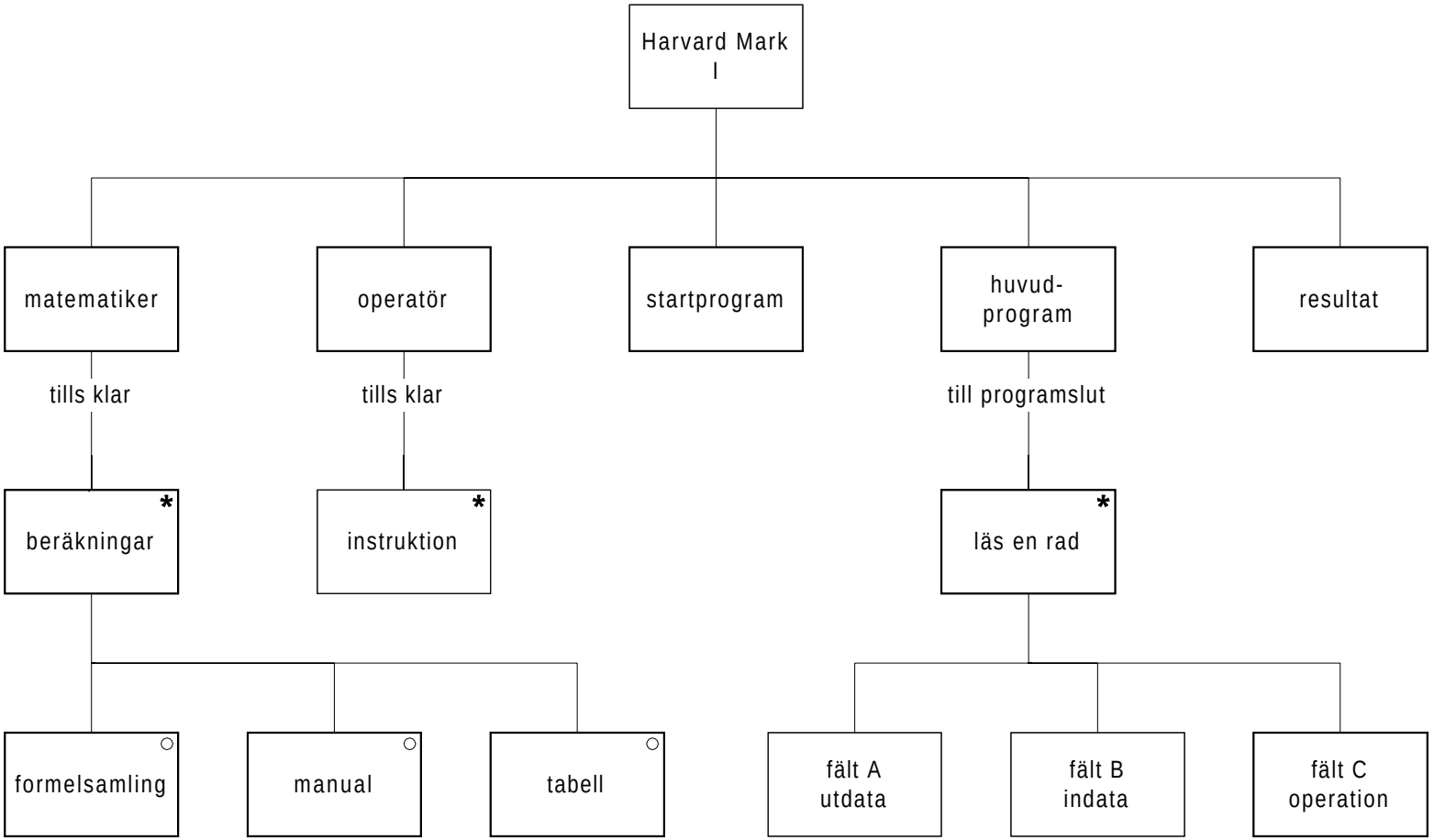
Vilhelmsdotter, M. (2002 b).
Planeringsrapport. Opublicerat manuskript.
Trollhättan: Högskolan i Trollhättan/Uddevalla,
Institutionen för informatik och matematik.



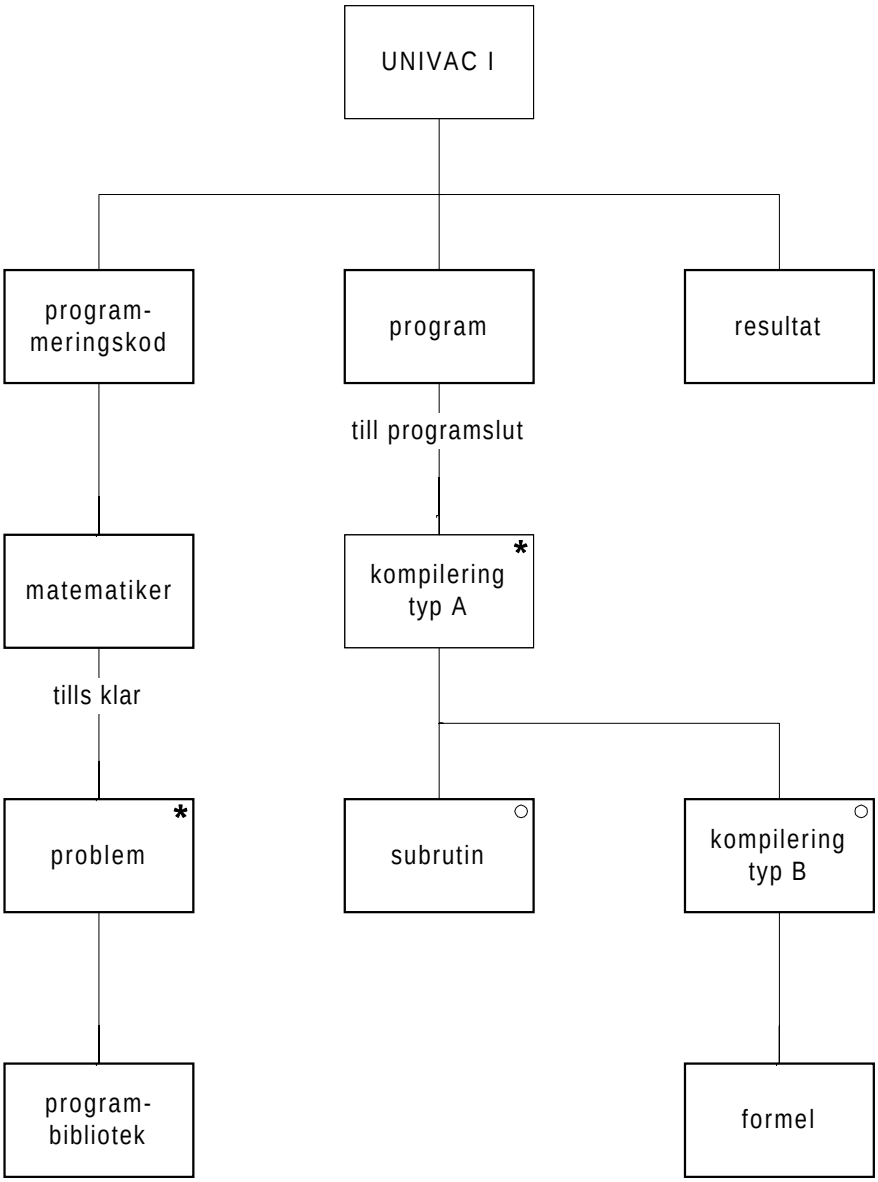
Figur 7.1 Funktionsschema för The Analytical Engine som Jackson Structured Programming (JSP). Källa: Randell (1982 b, c).



Figur 7.2 Förklaring av de olika elementens innebörd.



Figur 7.3 Funktionsschema för Harvard Mark I.
Källor: Randell (1982 d, e, f).



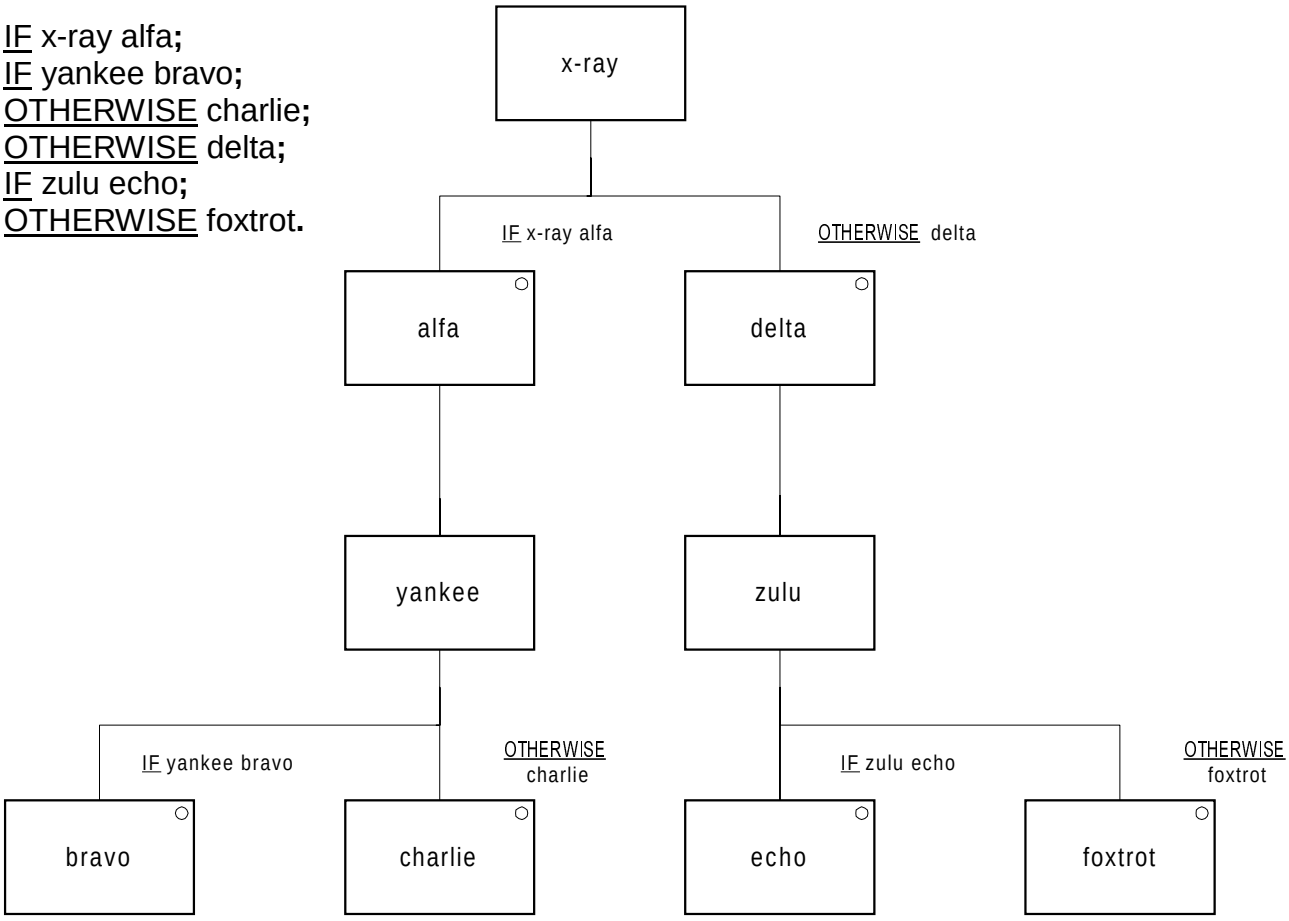
Figur 7.4 Funktionsschema för UNIVAC I
Källa: Hopper (1952).

Tabell 7.1 Reserverade ord i COBOL 61. **Bilaga 4**

ABOUT	COBOL	ENVIRONMENT	IN	MODULES	POSITION	RH	TALLY
ACCEPT	CODE	EQUAL	INCLUDE	MOVE	POSITIVE	RIGHT	TALLYING
ACCESS	COLUMN	EQUALS	INDEX	MULTIPLE	PREPARED	ROUNDED	TAPE
ACTUAL	COMMA	ERROR	INDEXED	MULTIPLIED	PRIORITY	RUN	TERMINATE
ADD	COMPUTATIONAL	EVERY	INDICATE	MULTIPLY	PROCEDURE	SA	THAN
ADDRESS	COMPUTE	EXAMINE	INITIATE	NEGATIVE	PROCEED	SAME	THEN
ADVANCING	CONFIGURATION	EXCEEDS	INPUT	NEXT	PROCESS	SD	THROUGH
AFTER	CONSTANT	EXIT	INPUT-OUTPUT	NO	PROCESSING	SEARCH	THRU
ALL	CONTAINS	EXPONENTIATED	INSTALLATION	NOT	PROGRAM-ID	SECTION	TIMES
ALPHABETIC	CONTROL	FD	INTO	NOTE	PROTECT	SECURITY	TO
ALPHANUMERIC	CONTROLS	FILE	INVALID	NUMBER	QUOTE	SEEK	TYPE
ALTER	CONVERSION	FILE-CONTROL	IS	NUMERIC	QUOTES	SEGMENT-LIMIT	UNEQUAL
ALTERNATIVE	COPY	FILE-LIMIT	JUSTIFIED	OBJECT-COMPUTER	RANDOM	SELECT	UP
AN	CORRESPONDING	FILE-LIMITS	KEY	OBJECT-PROGRAM	RANGE	SELECTED	UPPER-BOUND
AND	CURRENCY	FILLER	KEYS	OCCURS	RD	SENTENCE	UPPER-BOUNDS
APPLY	DATA	FINAL	LABEL	OF	READ	SEQUENCED	UNIT
ARE	DATE-COMPILED	FIRST	LAST	OFF	RECORD	SEQUENTIAL	UNTIL
AREA	DATE-WRITTEN	FLOAT	LEADING	OH	RECORDING	SET	UPON
AREAS	DE	FOOTING	LEAVING	OMITTED	RECORDS	SIGN	USAGE
ASCENDING	DECIMAL-POINT	FOR	LEFT	ON	REDEFINES	SIGNED	USE
ASSIGN	DECLARATIVES	FORMAT	LESS	OPEN	REEL	SIZE	USING
AT	DEFINE	FROM	LIBRARY	OPTIONAL	RELEASE	SORT	VALUE
AUTHOR	DEPENDING	GENERATE	LIMIT	OR	REMARKS	SOURCE	VALUES
BEFORE	DESCENDING	GIVING	LIMITS	OTHERWISE	RENAMES	SOURCE-COMPUTER	VARYING
BEGINNING	DETAIL	GO	LINE	OUTPUT	RENAMING	SPACE	WHEN
BITS	DIGITS	GREATER	LINE-COUNTER	OV	REPLACING	SPACES	WITH
BLANK	DISPLAY	GROUP	LINES	OVERFLOW	REPORT	SPECIAL-NAMES	WORDS
BLOCK	DIVIDE	HASHED	LOCATION	PAGE	REPORTING	STANDARD	WORKING-STORAGE
BY	DIVIDED	HEADING	LOCK	PAGE-COUNTER	REPORTS	STATUS	WRITE
CF	DIVISION	HIGH-VALUE	LOW-VALUE	PERFORM	RERUN	STOP	ZERO
CH	DOLLAR	HIGH-VALUES	LOW-VALUES	PF	RESERVE	SUBTRACT	ZEROES
CHARACTERS	DOWN	HOLD	LOWER-BOUND	PH	RESET	SUM	ZEROS
CHECK	ELSE	I-O	LOWER-BOUNDS	PICTURE	RETURN	SUPERVISOR	
CLASS	END	I-O-CONTROL	MEMORY	PLACES	REVERSED	SUPPRESS	
CLOCK-UNITS	ENDING	IDENTIFICATION	MINUS	PLUS	REWIND	SYMBOLIC	
CLOSE	ENTER	IF	MODE	POINT	RF	SYNCHRONIZED	

Källa: Lysegård (1966, s. 132 f.) Lista över reserverade COBOL-ord.

IF x-ray alfa;
IF yankee bravo;
OTHERWISE charlie;
OTHERWISE delta;
IF zulu echo;
OTHERWISE foxtrot.



Figur 7.5 Exempel på programsatser i COBOL.
Källa: Sammet (1969).

Tabell 7.2 Sammansmältning av programmeringsspråken. **Bilaga 6**

ABOUT	COBOL	ENVIRONMENT	IN	MODULES	POSITION	RH	TALLY
ACCEPT	CODE	EQUAL	INCLUDE	MOVE	POSITIVE	RIGHT	TALLYING
ACCESS	COLUMN	EQUALS	INDEX	MULTIPLE	PREPARED	ROUNDED	TAPE
ACTUAL	COMMA	ERROR	INDEXED	MULTIPLIED	PRIORITY	RUN	TERMINATE
ADD	COMPUTATIONAL	EVERY	INDICATE	MULTIPLY	PROCEDURE	SA	THAN
ADDRESS	COMPUTE	EXAMINE	INITIATE	NEGATIVE	PROCEED	SAME	THEN
ADVANCING	CONFIGURATION	EXCEEDS	INPUT	NEXT	PROCESS	SD	THROUGH
AFTER	CONSTANT	EXIT	INPUT-OUTPUT	NO	PROCESSING	SEARCH	THRU
ALL	CONTAINS	EXPONENTIATED	INSTALLATION	NOT	PROGRAM-ID	SECTION	TIMES
ALPHABETIC	CONTROL	FD	INTO	NOTE	PROTECT	SECURITY	TO
ALPHANUMERIC	CONTROLS	FILE	INVALID	NUMBER	QUOTE	SEEK	TYPE
ALTER	CONVERSION	FILE-CONTROL	IS	NUMERIC	QUOTES	SEGMENT-LIMIT	UNEQUAL
ALTERNATIVE	COPY	FILE-LIMIT	JUSTIFIED	OBJECT-COMPUTER	RANDOM	SELECT	UP
AN	CORRESPONDING	FILE-LIMITS	KEY	OBJECT-PROGRAM	RANGE	SELECTED	UPPER-BOUND
AND	CURRENCY	FILLER	KEYS	OCCURS	RD	SENTENCE	UPPER-BOUNDS
APPLY	DATA	FINAL	LABEL	OF	READ	SEQUENCED	UNIT
ARE	DATE-COMPILED	FIRST	LAST	OFF	RECORD	SEQUENTIAL	UNTIL
AREA	DATE-WRITTEN	FLOAT	LEADING	OH	RECORDING	SET	UPON
AREAS	DE	FOOTING	LEAVING	OMITTED	RECORDS	SIGN	USAGE
ASCENDING	DECIMAL-POINT	FOR	LEFT	ON	REDEFINES	SIGNED	USE
ASSIGN	DECLARATIVES	FORMAT	LESS	OPEN	REEL	SIZE	USING
AT	DEFINE	FROM	LIBRARY	OPTIONAL	RELEASE	SORT	VALUE
AUTHOR	DEPENDING	GENERATE	LIMIT	OR	REMARKS	SOURCE	VALUES
BEFORE	DESCENDING	GIVING	LIMITS	OTHERWISE	RENAMES	SOURCE-COMPUTER	VARYING
BEGINNING	DETAIL	GO	LINE	OUTPUT	RENAMING	SPACE	WHEN
BITS	DIGITS	GREATER	LINE-COUNTER	OV	REPLACING	SPACES	WITH
BLANK	DISPLAY	GROUP	LINES	OVERFLOW	REPORT	SPECIAL-NAMES	WORDS
BLOCK	DIVIDE	HASHED	LOCATION	PAGE	REPORTING	STANDARD	WORKING-STORAGE
BY	DIVIDED	HEADING	LOCK	PAGE-COUNTER	REPORTS	STATUS	WRITE
CF	DIVISION	HIGH-VALUE	LOW-VALUE	PERFORM	RERUN	STOP	ZERO
CH	DOLLAR	HIGH-VALUES	LOW-VALUES	PF	RESERVE	SUBTRACT	ZEROES
CHARACTERS	DOWN	HOLD	LOWER-BOUND	PH	RESET	SUM	ZEROS
CHECK	ELSE	I-O	LOWER-BOUNDS	PICTURE	RETURN	SUPERVISOR	
CLASS	END	I-O-CONTROL	MEMORY	PLACES	REVERSED	SUPPRESS	
CLOCK-UNITS	ENDING	IDENTIFICATION	MINUS	PLUS	REWIND	SYMBOLIC	
CLOSE	ENTER	IF	MODE	POINT	RF	SYNCHRONIZED	

Förklaring:	COBOL	FLOW-MATIC	Commercial Translator	FACT

Källor: Lysegård (1966), Sammet (1969).

Tabell 7.3 Reserverade ord i FLOW-MATIC. **Bilaga 7**

ADD	FILL	LESS	REMAINDER	THE
AGAINST	FORCE	MOVE	REPLACE	THROUGH
AND	FROM	MULTIPLY	RERUN	TO
BLK-RELATIVE	GIVING	NUMERIC	RESET	TRANSFER
BREAKDOWN	GO	NUMERIC-TEST	REWIND	TYPE
BY	GREATER	NUMERICAL	SELECT	USE
CLOSE-OUT	HALT	OF	SELECT-LEAST	VALUE
COMPARE	HSP	ON	SET	WHEN
COUNT	IF	OPERATION	SERVO	WHERE
COUNTER	IGNORE	OTHERWISE	SERVOS	WITH
DATA	IN	OUTPUT	SPACES	WRITE-ITEM
DIVIDE	INCREMENT	OVERLAY	STOP	X-1
END	INTO	PERIOD	STORE	ZEROES
EQUAL	INPUT	PERIODS	SUBTRACT	
EQUALS	INSERT	PRESELECTION	SUM	
EXCEEDS	JUMP	PRESET	SUPPLEX	
EXECUTE	KEY	PRINT-OUT	SWITCH	
FILE	LEADING	READ-ITEM	TEST	

Källa: Sammet (1969).

Tabell 7.4 Reserverade ord i Commercial Translator.

ADD	DO	GIVING	ON	TIMES
ALL	END	GO	OPEN	TO
AT	ENTER	HERE	OVERFLOW	TRUNCATED
BEGIN	EXACTLY	IN	OVERLAP	USING
CALL	FILE	INCLUDE	RECORD	WHEN
CLOSE	FILES	LOAD	SECTION	WITH
CORRESPONDING	FOR	MOVE	SET	
DISPLAY	FROM	NOTE	STOP	

Källa: Sammet (1969).