Programmering 1 - Uppgift 5

Peter Holm

Augusti 2020

1 Uppgift B: Genonsnittsålder

1.1 Förutsättningar

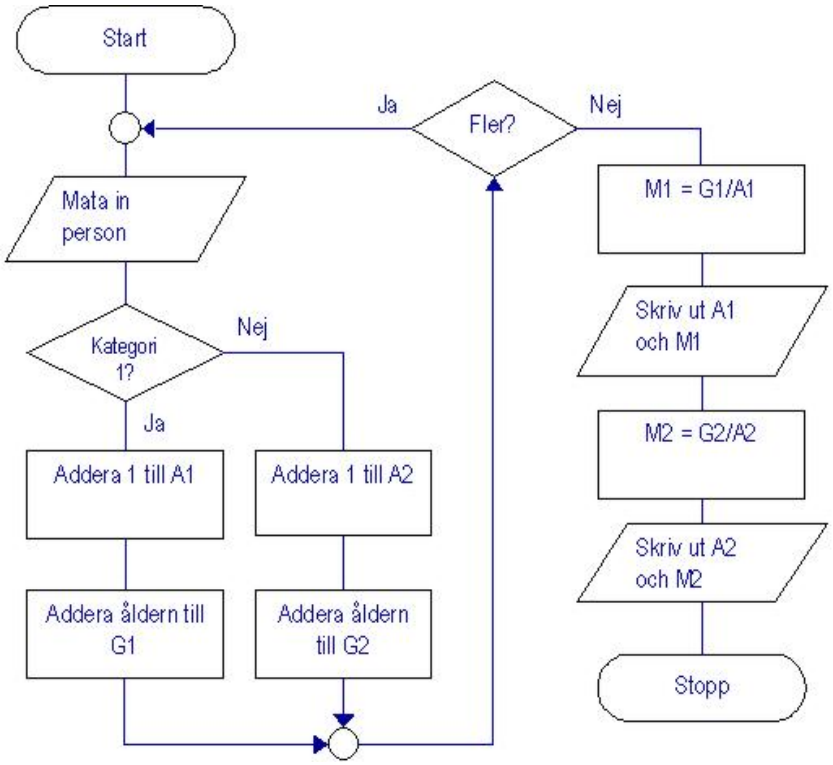
I uppgift B, skall ett program konstrueras för att kunna tilldelas uppgifter om flera personer. För att sedan beräkna genomsnittålder i respektive gruppering. Dvs, **kvinnornas** samt **männens**.

Vad vi vet,

- Är ett redan utformad strukturdiagram vi skall konstruera algoritmen efter.
- Samt specifikationerna **robusthet** och **stabilitet**. Programmet får inte visa inkorrekta resultat eller generera exekveringsfel.

1.2 Utformning

Med hjälp av det givna strukturdiagrammet,



underlättar vi konstruktionens detaljerade nivåer, genom att bryta ned diagrammet i sektioner,

1. **Indata**. Här definieras vilken information vi behöver av användaren för att kunna beräkna och få de resultat vi är ute efter.
2. **Selektion**. Här beskrivs hanterandet av sorteringen av de två grupperna.
3. **Iteration**. Här beskrivs upprepningen av ovanstående sektioner. Vill användaren lägga till fler personer i populationen? Om ja upprepa 1 och 2, annars fortsätt med 4.
4. **Kalkyl**. Hur gruppernas ålder adderas och divideras för att få genomsnittsåldern i respektive grupp.
5. **Utdata**. Presentationen av resultaten till användaren.

1.2.1 Psuedokod

Denna fas var inte vacker på något sätt. Att iterativt gå från skisser på papper och parallellt skriva källkod. När något inte var uppenbart gick jag tillbaka till penna och papper.

1.2.2 Indata

Tanken är att låta programmet tilldelas personnummer, för att ge informationen vi behöver.

Algorithm 1: Indata. Be användaren om personnummer.

Result: Tilldela programmet en sträng-variabel innehållande personnummer med formen: ÅÅÅÅMMDD-NNNN
Sträng personNummer;
Boolean korrekt = personNummer == ÅÅÅÅMMDD-NNNN ;
while *korrekt* **do**
  Print "Något gick fel, försök igen.";
end
Programmet fortsätter

Algorithm 2: Indata. Tar födelseåret ur personnumret som argument och beräknar personens ålder.

Result: Returnerar personens ålder som heltal
Heltal födelseÅr, ålder;
födelseÅr = ÅÅÅÅ;
ålder = nuvarandeÅr - födelseÅr;

Algorithm 3: Indata. Tar tredje siffran ur löpnumret(NNNN) från personnumret som argument och beräknar personens kön.

Result: Returnerar personens kön som boolean
Heltal könSiffra;
Boolean kön;
könSiffra = NNNN;
kön = Sant om värdet i 'könSiffra' är jämn, annars falskt om udda;

1.2.3 Selektion

Algorithm 4: Selektion. Tar som argument ålder och kön från `Indata 2` & `Indata 3`

Result: Lagrar personens ålder i en av två fältlistor beroende av könet.
If kön = sant;
Fält kvinna.addera(ålder);
else;
Fält man.addera(ålder);
fI

1.2.4 Iteration

Algorithm 5: Iteration.

Result: Låter användaren lägga till personer i populationen.
Iterate:
`Indata`;
`Selektion`;
While (Done)

1.2.5 Kalkyl

Algorithm 6: Iteration. Beräkna vardera grupps genomsnittsålder. Tar de två fälten, deklarerade i `selektion`, som argument.

Result: Kvinnliga och manliga genomsnittsåldern.
for (`Gruppens population`);
Gruppens totala ålder += åldervärdet ur fält;
rof;
Gruppens totala ålder / Gruppens population = genomsnittsålder

1.2.6 Utdata

Algorithm 7: Utdata. Tar resultatet av `kalkyl` som argument.

Result: Ger en utskrift av resultatet från **kalkyl**.
Print "Genomsnitts åldern är: `kalkyl`

2 Uppgift A: Inkomstskatt

2.1 Förutsättningar

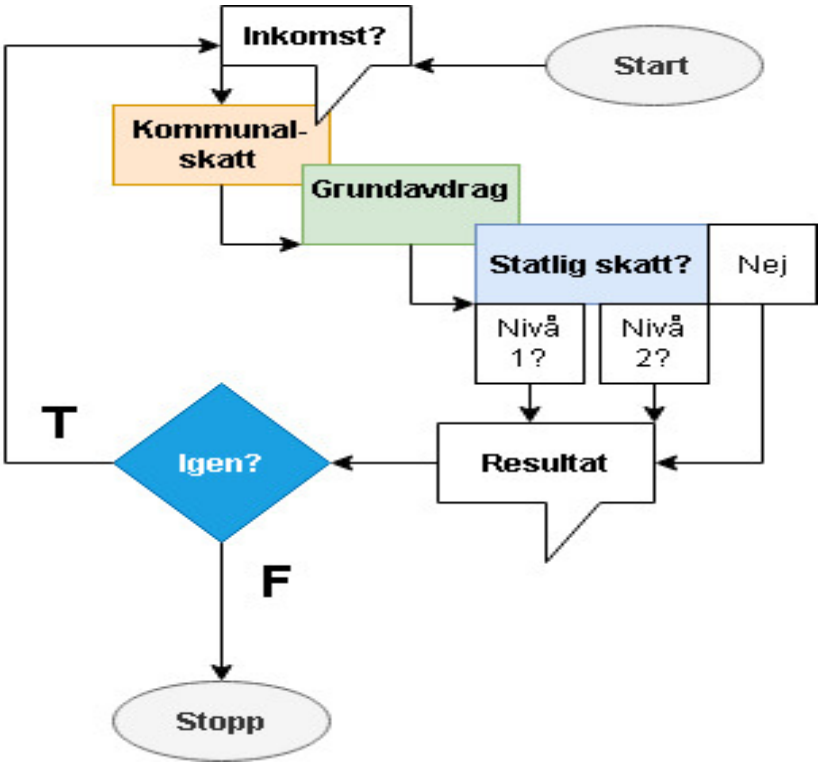
I uppgift A, skall ett program konstrueras för att kunna tilldelas en persons årliga inkomst. För att sedan beräkna inkomstskatten.

Vad vi vet,

- En beskrivning på hur programmet skall instrueras samt hänvisning till information från skatteverket.
- Samt specifikationerna **robusthet** och **stabilitet**. Programmet får inte visa inkorrekta resultat eller generera exekveringsfel.

2.2 Utformning

Efter att läst på och bekantat mig med skatter, ritades ett strukturdiagram



Sedan underlättar vi konstruktionens detaljerade nivåer, genom att bryta ned diagrammet i sektioner,

1. **Indata**. Här tilldelas programmet användarens årliga bruttoinkomst.
2. **Sekvens**. Beräknar kommunalskatt, grundavdraget och statlig skatt.
3. **Selektion**. Här beskrivs valet av skiktgränsen till den statliga skatten. Tillhör den beskattningsbara inkomsten nedre eller övre skiktgränsen.
4. **Utdata**. Presentationen av resultaten till användaren.
5. **Iteration.** Låter användaren mata in ett nytt värde.

2.2.1 Psuedokod

Denna fas var inte vacker på något sätt. Att iterativt gå från skisser på papper och parallellt skriva källkod. När något inte var uppenbart gick jag tillbaka till penna och papper.

Algorithm 8: Indata. Be användaren om bruttoinkomst.

Result: Tilldela programmet ett flyttal innehållande årlig bruttoinkomst i valutan SEK
Sträng inkomst? = **flyttal** inkomst;

Algorithm 9: Sekvens beräkning. Kommunalskatt, baserad på den genomsnittliga procenten i landet.

Result: Ger kommunalskatten
DEFINE KOMMUNALSKATT 0.32%;
Return inkomst * kommunalskatt;

Algorithm 10: Sekvens Sätter grundavdraget som konstant. I det här fallet 13400SEK

Result: Grundavdraget
DEFINE GRUNDAVDRAG = 13400 ;

Algorithm 11: Sekvens Sätter skiktgränsvärdena som konstanter, baserat på inkomståret 2019

Result: Statlig skatt
DEFINE NEDRE_SKIKTGRÄNS = 490700 ;
DEFINE ÖVRE_SKIKTGRÄNS = 689300 ;
DEFINE NEDRE_SKIKTGRÄNS_PROCENT = 0.20 ;
DEFINE ÖVRE_SKIKTGRÄNS_PROCENT = 0.25 ;

Algorithm 12: Selektion. Tar `inkomst` som argument och väljer om personen behöver betala skatt, eller i vilken skiktgräns den beskattningsbara inkomsten skall placeras.

Result: Ger statlig skatt
if inkomst > NEDRE_SKIKTGRÄNS;
Return inkomst ∗ NEDRE_SKIKTGRÄNS_PROCENT;
else if inkomst > ÖVRE_SKITGRÄNS;
Return inkomst ∗ ÖVRE_SKIKTGRÄNS_PROCENT;
fi;

2.2.2 Utdata

Algorithm 13: Utdata. Tar `inkomst`, resultatet av `kommunalskatt grundavdraget` och `statlig skatt` som argument.

Result: Ger en utskrift av resultatet.
Print "Skatten är": `kalkyler`

2.2.3 Iteration

Algorithm 14: Iteration.

Result: Låter användaren tilldela en annan inkomst.
Iterate:
`Indata`;
`Sekvens`;
`Selektion`;
`Utdata`;
While (Done)

2.3 Test & Felsökning

- Riktpunkt var att först och främst få programmet att fungera. Trassla igenom de triviala kompileringsfelen och få tecken på liv.
- Därefter lösa de logiska felen, genom deklarera logg-utskrifter till konsolen samt sätta brytpunkter i avlusar-verktyget som är integrerad i program-utvecklingsmiljön.
- Att hantera felsignaler för att undvika exekveringsfel lät jag programmet tilldelas alla typer av värden för att simulera olika scenarion som kan ske, vid användning av programmet.
  Vid upptäckt av krascher deklarerades undantag för att fånga felsignalerna som uppstod vid test.

2.4 Vidareutveckling

Möjligheterna till att göra programmen i dessa projekt mer användbara, är oändliga t.ex. koppla skatteverkets öppna API till databaser i skatteprogrammet för att göra det mer dynamiskt, med skattetabeller, inkomstårens olika procentsatser för mer korrekta kalkyler.
Eller låta genomsnittsprogrammet löpa igenom en databas fylld med tusentals personnummer för att få ut genomsnittsåldern hos dina användare.

I vilket fall har jag förstått att planeringen, design och framförallt hålla sig till kraven som är satta, är de mest essentiella delarna för att lyckas få robusta och pålitliga program.
Hade själv svårt att hålla mig ifrån tangentbordet innan jag riktigt förstod tillvägagångssättet, vilket vid vissa tillfällen fick mig att måla in mig i ett hörn, som man därefter fick radera sig ur och ge sig tillbaks till planeringen.

Generellt kan dessa program säkerligen förenklas, bli mer självdokumenterad samt testas ytterligare. Men tid är dyrbart, likväl att från första början skriva välkonstruerade programtexter.