Programmering 1 - Uppgift 4

Peter Holm

Juli 2020

1

1.1 Vad är en algoritm?

En algoritm är en uppsättning instruktioner för hur man, steg för steg, löser en matematisk eller logisk uppgift.
Vid programmering, skrivs algoritmer för att instruera datorn till att utföra en viss uppgift.

Generellt sett, finns algoritmer överallt i vardagen. Att följa ett recept vid matlagning. Metoder som används vid addition och division när man räknar. Att vika ett par byxor eller en tröja. Din morgonrutin. Kan alla anses vara algoritmer.

"Algoritmer kan uttryckas på många olika sätt. Ett vanligt sätt är att algoritmen anges i psuedokod. Man kan också använda strukturdiagram."

Förklara skillnaden mellan psuedokod och strukturdiagram

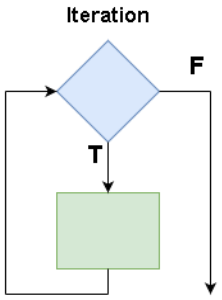
Innan en algoritm designas är det viktigt att först förstå vad problemet är.

1.1.1 Psuedokod

Programmerare använder sig utav programspråk som följer en viss syntax, för att få sina program att exekveras ordentligt.
Psuedokod är däremot inget programmeringspråk, utan en enklare metod till att beskriva instruktioner utan att behöva trassla med något specifikt syntax. Det finns ingen strikt uppsättning av standardiserad notation för psuedokod. Vilket underlättar att forma instruktioner som sedan kan bildas till programtext samt ger överblick av problemet.

1.1.2 Strukturdiagram

I strukturdiagram används figurer för illustrativ beskrivning av olika steg i program. Exempel, en diamant-figur symboliserar val i ett program.



Ett strukturdiagram är utmärkt till att överblicka program, men blir otympliga vid mer detaljerade nivåer av programmen.

1.2

"För att det skall vara möjligt att konstruera generella algoritmer, måste den beskrivningsmetod man använder ha förmåga att uttrycka tre konstruktioner, även kallade kontrollstrukturer. Dessa är sekvens, selektion och iteration. Förklara var och en av de tre kontrollstrukturerna, dvs. vad innebär sekvens, selektion resp. iteration?"

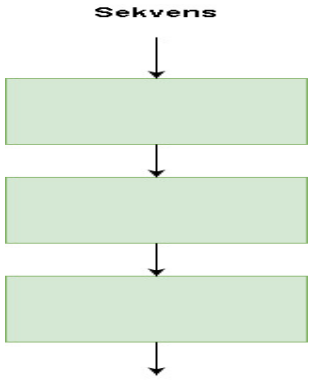
1.2.1 Hur uttrycker man sekvenser i Java? Skriv ett exempel (källkod).

```
public class Temperature {
  public static void main(String[] args) {

    String input = JOptionPane.showInputDialog
    ("Enter temperature in (F): ");

    double f = Double.parseDouble(input);
    double c = (f - 32) * 5 / 9;

    JOptionPane.showMessageDialog
    (null,
    f + " degrees F " + " equals " + c + " degrees C");
  }
}
```

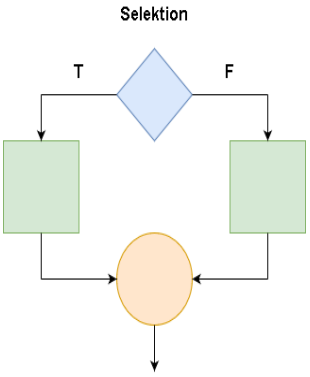


1.2.2 Hur uttrycker man selektion i Java? Skriv ett exempel (källkod).

```
public class Selection {
    public static void main(String[] args) {

        String personalSerialCode = "19930609-1337";
        int sexDigit = Character.getNumericValue
        (personalSerialCode.charAt(11));

        if (sexDigit % 2 == 0)
            System.out.println("You are female");
        else
            System.out.println("You are male");
    }
}
```

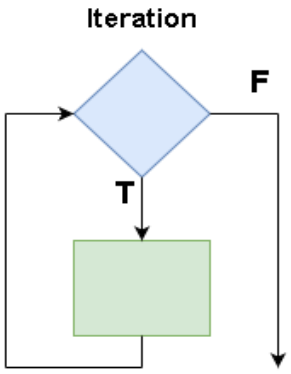


1.2.3 Hur uttrycker man iteration i Java? Skriv ett exempel (källkod).

```java
public class Iteration {
    public static void main(String[] args) {
        /* Fibonacci */
        int n = 10, t1 = 0, t2 = 1;
        System.out.print("First " + n + " terms: ");

        for (int i = 1; i <= n; ++i)
        {
            System.out.print(t1 + " + ");

            int sum = t1 + t2;
            t1 = t2;
            t2 = sum;
        }
    }
}
```



1.3 En mycket viktig teknik vid programkonstruktion är ”stegvis förfining”. Förklara denna teknik med hjälp av ett exempel. (OBS! inte källkod).

Exempel: Morgonrutin.

1 Vakna

1.1 Slå av larmet

1.1.1 Lokalisera ljudkällan.

1.1.2 Swipea av larmet.

2 Stig upp

2.1 Om sov på soffan, fortsätt till sektion 'Hygien'.

3 Bädda

3.1 Om inte renbäddat på länge eller sängen är ockuperad, skriv upp på 'att göra-lista'. Annars;

3.1.1 Jämna ut lakan

3.1.2 Puffa kuddarna

3.1.3 Lägg över täcket

4 Hygien

4.1 Duscha

4.1.1 Vrid på kranen.

4.1.2 Justera temperatur.

4.2 Borsta tänderna

4.2.1 Applicera tandkräm.

4.2.2 Borsta tills varje tand är borstad

2 C-uppgift

Det kan finnas tre olika typer av fel i ett program:

- Kompileringsfel
- Logiska fel
- Exekveringsfel

Förklara dessa typer med hjälp av tre olika exempel (källkod). OBS! Detta är en C-uppgift. Du behöver förklara din källkod i detalj (Utförligt).

2.1 Kompileringsfel

Varje programmeringsspråk har sina egna uppsättningar av regler för att kunna formulera sina texter. Programmerare måste följa respektive skrivregler för att datorn skall kunna förstå programmen vid kompilering. Att lyckas följa dessa regler fullt ut är inte alltid lätt, och nästintill överraskande om man lyckas på första försöket. Misslyckas man får man snart reda på vad som inte stämmer genom kompileringsfelet som dyker upp i terminalfönstret. Dessa fel kallas kompileringsfel och uppstår vid kompileringen av program.
Därmed är det lätt att finna och rätta dessa fel.

Nedan följer ett exempel på hur en mängd kompileringsfel kan se ut vid kompilering av ett program.

```
#include <stdio.h>

int main ()
{
    int value;

    printf("Assign value: ";

    scanf("%d", value);

    Double root = sqrt(value)

    printf("Squareroot of %d = %f\n", value root);

}
```

Vilket ger kompileringsfelen:

```
square.c: In function 'main':
square.c:6:28: error: expected ')' before ';' token
     printf("Assign value: ";
                            ^
s.c:12:1: error: expected ';' before '}' token
 }
 ^
```

- Som visar att filen vi precis kompilerade (filen square.c) har en funktion/klass som är namngiven "main" och att inuti denna funktion befinner sig ett eller fler kompileringsfel.

  ```
  square.c: In function 'main':
  ```

- Sedan en riktning, genom vilken rad och vid vilket tecken problemet har uppstått i. Följt av ett felmeddelande på hur problemet ser ut.

  ```
  square.c:6:28: error: expected ')' before ';' token
       printf("Assign value: ";
                              ^
  ```

Där ifrån kan man sedan se i källkoden att på rad 6 vid tecken 28 skall det vara en parentes ')' innan semikolon (;).

```
printf("Assign value: "; // *Ledsen-emoji*
printf("Assign value: "); // *Glad-emoji*
```

Fortsätt sedan med nästa kompileringsfel i listan tills programmet lyckas kompileras och får utseendet:

```
#include <stdio.h>
#include <math.h>

int main ()
{
    int value;

    printf("Assign value: ");
    scanf("%d", value);

    double root = sqrt(value);

    printf("Squareroot of %d = %f\n", value root);
}
```

Med utmatningen:

```
Assign value: 2
Squareroot of 2 = 1.414214
```

Referens till programtext finner du **här**

2.2 Logiska fel

Logiska fel uppstår när man tänkt fel vid konstruktionen av programmet. Dessa fel är svåra att hitta, då programmet lyckas kompileras och exekveras utan några felmeddelanden. Men vid test av programmet visar sig resultatet inkorrekt.

Exemplet nedan är ett program som skall skriva ut alla primtal mellan 1 till n.

```
#include <stdio.h>
int main()
{
  int n, counts = 0;

  printf("Enter value n: ");
  scanf("%d", &n);

  for (int i=1; i<=n; i++) {
  // Check if prime.
  _Bool is_prime = 0;
  for (int k = 2; k<i; k++)
    if (i % k == 0)
      is_prime = 1;
    if (is_prime) {
      counts++;
      printf(" %d", i);
      if (counts % 10 == 0)
        printf("\n");
    }
  }
}
```

Programmet gör inte som det var tänkt. Då programmet skriver ut alla tal förutom primtalen? Med utskriften:

```
Enter value n: 100
  4   6   8   9   10  12  14  15  16  18
  20  21  22  24  25  26  27  28  30  32
  33  34  35  36  38  39  40  42  44  45
  46  48  49  50  51  52  54  55  56  57
  58  60  62  63  64  65  66  68  69  70
  72  74  75  76  77  78  80  81  82  84
  85  86  87  88  90  91  92  93  94  95
  96  98  99  100
```

Zoomar vi in på undersökningen av primtalen,

```
for (int i=1; i<=n; i++) {
  // Check if prime.
  _Bool is_prime = 0; // ger ut 'falskt' vid primtal
  for (int k = 2; k<i; k++)
    if (i % k == 0)
      is_prime = 1; // 'sant' vid delbara tal.
    if (is_prime) { // <----------
      counts++;
      printf(" %d", i);
      if (counts % 10 == 0)
        printf("\n");
    }
  }
```

ser vi att det blivit fel vid de logiska sannings-värdena.

Genom att ändra de logiska uttrycken till att istället ge ut 'sant' vid primtalen

```
_Bool is_prime = 1; // 'sant' vid primtal
.....
.....
if (i % k == 0)
  is_prime = 0; // 'falskt' vid delbara tal
if (is_prime){  // 'sant' vid primtal
....
}
```

Ger sedan korrekta utskriften:

```
Enter value n: 100
  1  2  3  5  7   11
  13  17  19  23  29
  31  37  41  43  47
  53  59  61  67  71
  73  79  83  89  97
```

Referens till programtext finner du **här**

2.3 Exekveringsfel

Detta fel uppstår under exekveringen av program. Programmet följer dess språkregler och kompileras felfritt, men innehåller fel, som vid mänsklig inmatning gör att programmet genererar en felsignal, och om detta scenario inte hanteras i programmet, avbryts exekveringen med en felutskrift.

Följande är ett program som läser in en vald textfil, och skriver ut dess innehåll i terminal-fönstret.

```
#include <stdio.h>

int main() {

    char file_name[100];

    printf("Enter file-name: ");
    scanf("%s", file_name);

    FILE *input_file = fopen(file_name, "r");

    char s[100];
    while (fgets(s, 100, input_file) != NULL)
        printf("%s", s);
}
```

Ett exekveringsfel som kan uppstå vid körning av detta program, är att filen som skall öppnas inte existerar i arbetskatalogen, där programtexten ligger.

Då uppstår felutskrifter som,

```
Enter file-name: .bash_history
Segmentation fault (core dumped)
```

och programmet avbryts.
För att undvika att programmet avbryts och lämnar användaren i oklarheter, om vad som skett. Kan man fånga felet och skriva ut en förklaring till varför felet uppstod:

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    char file_name[100];

    printf("Enter file-name: ");
    scanf("%s", file_name);

    FILE *input_file = fopen(file_name, "r");

    if (input_file == NULL) {
        printf("Can't find any file named '%s'", file_name);
        exit(1);
    }

    char s[100];
    while (fgets(s, 100, input_file) != NULL)
        printf("%s", s);
}
```

if-satsen som lades till, kollar ifall det tilldelade filnamnet är kopplad till någon befintlig fil.
Om så inte är fallet får man utskriften;

```
Enter file-name: .bash_history
Can't find any file named .bash_history
```

Referens till programtext finner du **här**

3 Referenser

3.1 C-uppgifter

3.1.1 Kompileringsfel

```
#include <stdio.h>
#include <math.h>

int main ()
{
    int value;

    printf("Assign value: ");
    scanf("%d", value);

    double root = sqrt(value);

    printf("Squareroot of %d = %f\n", value root);
}
```

- Programmet ovan tilldelas ett heltalsvärde från användaren genom standardbiblioteket `stdio`'s funktion `scanf()`;

```
    int value; // Deklarerar heltals-variabeln, 'value'.

    printf("Assign value: "); // utrskrift.
    scanf("%d", value); // Tilldelar ett heltal till variabeln.
```

- Sedan tilldelas en flyttals-variabel, kvadratroten ur heltalsvariabeln användaren tilldelade. Genom standard funktionen `sqrt()` taget ifrån `math.h`.

```
    double root = sqrt(value);
```

- Och slutligen skriver ut svaret till användaren genom `printf()` funktionen från `stdio.h`

```
    printf("Squareroot of %d = %f\n", value root);
```

Gå till nästa uppgift, 'logiska fel' **här**

3.1.2 Logiska fel

```
#include <stdio.h>
int main()
{
  int n, counts = 0;

  printf("Enter value n: ");
  scanf("%d", &n);

  for (int i=1; i<=n; i++) {
  // Check if prime.
  _Bool is_prime = 1;
  for (int k = 2; k<i; k++)
    if (i % k == 0)
      is_prime = 0;
    if (is_prime) {
      counts++;
      printf(" %d", i);
      if (counts % 10 == 0)
        printf("\n");
    }
  }
}
```

- I programmet ovan, tilldelas heltalsvariabeln `n` av användaren. `n` sätts sedan in som vilkorsdel i en `for`-sats.

  ```
  for (int i=1; i<=n; i++)
  ```

- `for`-loopen ovan, räknar sedan från 1 till värdet i `n`. För varje varv utförs även en kapslad `for`-loop deklarerad inuti `for`-loopen, ovan.
- Den kapslade `for`-loopens uppgift är att undersöka ifall värdet är ett primtal eller inte, genom iteration, moduleras värden `i` från huvud-loopen med `k` från sub-loopen.
- Om värdet `i` är delbart med `k`, sätts *is_prime* till falskt, och sant om resten är större än noll.
- Slutligen skrivs primtalet ut och om varven(`counts`) av `for`-loopen är delbart med 10 utförs en ny rad. Främst för att utskriften skall bli nätt.

Gå till nästa uppgift, 'exekveringsfel' **här**

3.1.3 Exekveringsfel

```
#include <stdio.h>

int main() {

    char file_name[100];

    printf("Enter file-name: ");
    scanf("%s", file_name);

    FILE *input_file = fopen(file_name, "r");

    char s[100];
    while (fgets(s, 100, input_file) != NULL)
        printf("%s", s);
```

- I programmet ovan tilldelas en sträng-variabeln(tecken-fältet) `file_name` ett filnamn av användaren.
- Sedan tilldelas `input_file` av typen `FILE` till att öppna filen med namnet `file_name` som tilldelades av användaren. ”r” beskriver att `fopen()` skall läsa(read) den valda filen.

```
    FILE *input_file = fopen(file_name, "r");
```

- Sedan tilldelas tecken-fältet `s`. och en `while`-loop deklareras till att läsa de 100 första tecknen i filen och skriva ut i terminal-fönstret.

```
    char s[100];
    while (fgets(s, 100, input_file) != NULL)
        printf("%s", s);
```

Tack för att du läste. Gå tillbaks till början **här**