Programmeringsparadigm

Föreläsning 1

Robert W Floyd - The paradigms of programming

1978 gav Robert Floyd en föreläsning för att fira sin Turing award. I den föreläsningen myntades begreppet programmeringsparadigm.

Floyd hade en bredare definition av paradigm som även innefattade algoritmer och algoritmidéer (dessa kommer i ADK:n inte i den här kursen) samt egenskaper hos utvecklingsmiljöer (som inte heller räknas till den här kursen).

The paradigms of programming - Robert W Floyd

“from [Thomas] Kuhn:

“The older schools gradually disappear. In part their disappearance is caused by their members' conversion to the new paradigm. But there are always some men who cling to one or another of the older views, and they are simply read out of the profession, which thereafter ignores their work.” In computing, there is no mechanism for reading such men out of the profession. I suspect they mainly become managers of software development.”

“The acquisition of new paradigms by the individual programmer may be encouraged by reading other people's programs, but this is subject to the limitation that one's associates are likely to have been chosen for their compatibility with the local paradigm set.. Evidence for this is the frequency with which our industry advertises, not for programmers, but for Fortran programmers or Cobol programmers. The rules of Fortran can be learned within a few hours; the associated paradigms take much longer, both to learn and to unlearn.”

Definition av programmeringsparadigm

Vad är ett programmeringsparadigm:

1. En eller flera egenskaper (features) som ett programmeringsspråk har.
2. En eller flera egenskaper (features igen) som ett språk *inte* har.
3. De bakomliggande antagandena bakom språkets modell.

Det finns inte en one-to-one-mappning mellan paradigmer och språk. Många av de vanligaste programmeringsspråken stödjer flera paradigm (och då släpper man på krav på vad språket inte får ha). Vissa språk stödjer svagt eller delvis ett paradigm.

Högnivåspråk och lågnivåspråk

Program som ligger nära hårdvaran kallas lågnivåspråk. Ett exempel är assembler.
Språk som försöker abstrahera bort delar av hårdvaran kallas högnivåspråk.
Vad som räknas som högnivåspråk förändras från år till år. Assembler har alltid varit ett lågnivåspråk, men på sistone har C börjat räknas dit.
Vilka språk blir lågnivå i framtiden?

Imperativ programmering

En sekvens av satser som beskriver hur datorn ska bygga upp ett state (tillstånd).

Subprogram (kallas subrutiner i assembler) anropas med hopp (jump och branch, exempel jsr, beq, bne i assembler, goto i C och BASIC) och utan parametrar. Först modifieras programmets state för att hantera anropet, resultatet av beräkningarna levereras genom bieffekter på globala variabler.

Många universitetskurser i Python och Java underkänner eller sänker betygen för studenter som är låsta i detta tankesätt. Även om det ibland löser uppgiften så visar det inte att studenter tagit till sig av mer moderna tankesätt som gör det enklare att skriva stora program.

Flödeskontroll i imperativ programmering

I koden finns labels (etiketter). För att förflytta sig till en label i assembler så används en branch (hopp).

Dessa har namn som branch if equal (beq), branch if not equal (bne), branch if less than (blt), eller liknande. Dessa kollar statusflaggor på CPU:n som sätts i samband med jämförelser.

I C och BASIC heter motsvarande instruktion goto.

Nackdelen med denna approach: spaghettikod.

Strukturerad programmering

Påminner om imperativ programmering, men…
1. Vi introducerar for- och while-satser/loopar.
2. Vi förbjuder goto...

Dijkstra, Edsger; “Goto statement considered harmful.”; Communications of the Association for Computing Machinery (CACM 1968)

...Men Donald Knuth protesterar (1974) Structured programming with goto...

3. Strukturerad programmering försöker förbjuda flera retursatser från samma funktion, men många protesterar mot det också.

Procedurell programmering

En procedur är det vi kallar funktion, procedur, metod eller subrutin i programmeringsspråk.
Procedurell programmering bygger vidare på strukturerad programmering men lägger till konceptet räckvidd (scoping) för variabler. Detta innebär att programmet glömmer bort namn när vi lämnat funktionen.
Språk som stödjer denna sorts procedurer: Fortran, Algol, COBOL, Basic

Objektorienterad programmering

Objektorienterade språk har:
Inkapsling
arv
Dynamisk bindning av metodanrop

Deklarativ programmering

Programmen beskriver *vad* som ska beräknas, inte *hur* det ska beräknas.

Om något definitivt inte är imperativt så är det deklarativt.

Deklarativ programmering undviker bieffekter.

Referenstransparens innebär att varje uttryck kan ersättas med sitt värde utan att förändra resultatet!

Deklarativa paradigmer: Funktionella, logiska,

Exempel på deklarativt språk: SQL (databasförfrågningar)

Funktionell programmering

1. Vi har immutability, rena funktioner, funktioner som första klassens medlemmar. Referenstransparens följer av (se punkt 2).
2. Vi har inte destruktiv tilldelning eller bieffekter av funktionsanrop.
3. Bakomliggande antaganden: Vi vill arbeta med uttryck och symboler, inte detaljer om hur datorn fungerar.

De funktionella språken heter Lisp, Erlang, Haskell och Clojure. Ett språk som är fullt ut funktionellt men också tillåter flera imperativa paradigm är Scala. Funktionellt inspirerad syntax och språkkonstruktioner finns i C++, C#, Java, Javascript, Python med flera.

Logisk programmering och villkorsprogrammering

Villkorsprogrammering (Constraint Programming) går ut på att speca villkoren för en beräkning och sedan låta språket räkna ut svaret själv.
Logisk programmering är ett deklarativt paradigm.
Främsta (och enda exemplet i kursen) är Prolog som kan testas med swipl:

?- ['kings_and_queens.pl']

?- parent(silvia, Who).

Who = walther ;

Who = alice.

Prologsyntax

Ett faktum i prolog kallas atom (Nej, inte en fysikalisk atom, inte editorn, inte programmeringsspråket Atom, inte CPU:n från Intel och inte “atomär” som i parallellprogrammering heller)

Bästa sättet att lära sig Prolog till extralabbarna är manualen till SWI-prolog.

Labbarna för högre betyg X1 och X2

Konsten att lära sig ett programmeringsspråket X (för någon sträng X):
Alla programmeringsspråk har bra Wikipediasidor. Kolla dem och se till att kunna alla paradigm, kategorier och språkegenskaper som står där.
Sök på “X tutorial” i din sökmotor (till exempel Google).
Sök på “common X errors”, “X gotchas”.
Sök på “X api”.
Installera språket på din egen dator.
Ibland är manualen till kompilatorn den bästa guiden (till exempel SWI-prolog)
Labba gärna i par för att ha någon att fråga.

För att bemästra ett språk (överkurs, kommer inte på KS:en)

Läs den bästa/de bästa böckerna om språket.
Många stora språk är standardiserade med en spec. Läs specen. Det går bäst med Python eller Go, är halvtungt med Java Language Specification och extremt tungt med C++17-specen (drygt 1600 sidor).
Skriv ett större projekt i språket till exempel ett spel. Gör Inet-labben!

Programmeringsparadigm

Del 2: Matematiska modeller bakom datorer och programspråk

Lambdakalkyl

Lambdakalkylen (lambda calculus) består av tre regler. Fascinerande nog är detta en universell beräkningsmodell!

Lambdakalkyl	Namn	Exempel i Haskell
x	variabel	let x = 23
(**λ**x.M)	abstraktion	let f = (\x->x*x)
(M N)	applikation	head [1,2,3]

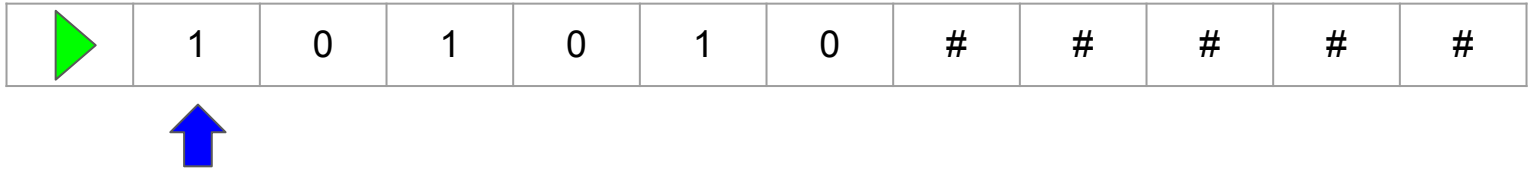
Lisp använder applikationssyntaxen från Lambdakalkyl.
På KS:en behöver ni inte kunna utföra beräkningar med lambdakalkyl, men ni behöver känna till vad de tre reglerna heter och vad begreppen abstraktion och applikation betyder.

Konceptet abstraktion

Abstraktion innebär att vi tar bort detaljer och ersätter dessa med ingenting eller placeholders. Några exempel följer:

1. En bit i datorns minne är spänningsnivån i en vippa. När vi programmerar ignorerar vi den exakta spänninsnivån och säger 1 eller 0.
2. En klass är en abstraktion från ett objekt, där fältens värden har lämnats tomma.
3. En funktion är en abstraktion över kod, där en bit kod ersatts med ett namn för att enkelt kunna återanvändas.
4. Superklasser är en abstraktion från vanligt förekommande objekt till regler för hur de skapas, läses, uppdateras och raderas (CRUD).

Turingmaskinen: Alan Turing (1936)



current state	read	write	move	next state
start	>	>	R	start
start	1	#	R	start
start	0	#	R	start
start	#	#	L	halt

Turingmaskinens egenskaper

En Turingmaskin består av:

1. En minnespekare och ett “nuvarande tillstånd” (pilarna).
2. Ett oändligt band med 4 symboler: “>”, “0”, “1”, “#”. Den första kallas start, den sista kallas blanksteg.
3. Ett ändligt antal states (tillstånd) som maskinen kan befinna sig i. Tillståndet bryr sig inte om andra positioner på bandet än den som pekas ut.

Tillstånden i detalj

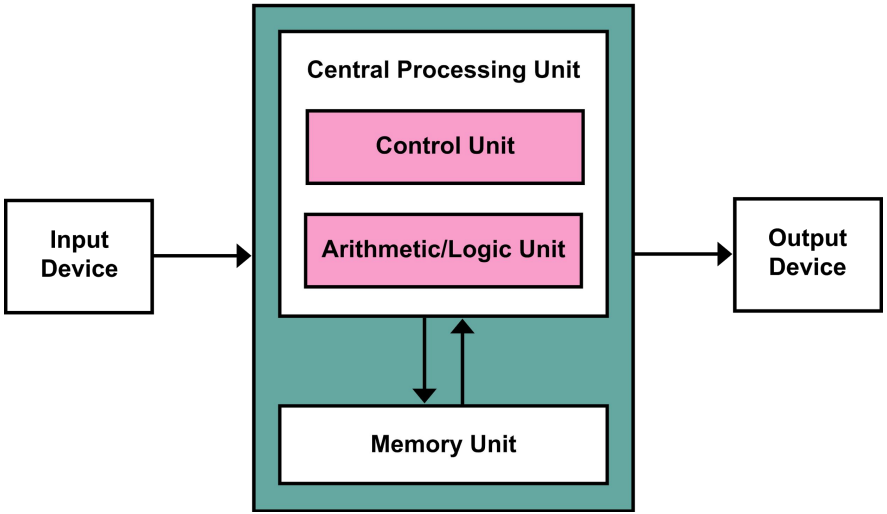
1. Början av varje rad innehåller tillstånd och vad det står på bandet vid pekaren.
2. Mitten av varje rad innehåller instruktioner om vad nuvarnade position ska ersättas med (en av 4 symboler, möjligen samma som förut) och en instruktion om vart pekaren ska flyttas (L eller R).
3. Vid slutet av varje rad läses “new state” och symbolen vid bandets pekare. Baserat på detta väljs vilken rad maskinen ska flytta till.

Von Neumann-arkitekturen

“First draft of a report on the EDVAC.” (1945)
I ursprungsversionen fanns ingen I/O

Kallas också Princeton-arkitekturen.

Von Neumann-flaskhalsen ligger mellan minnet och CPU:n.



Von Neumann-flaskhalsens lösning (ej på tentan)

Separat buss för data och instruktioner

CPU cache (håller redan använt minne och dess grannar)

Pipelining, Spekulativ exekvering

(Begränsad Stack på CPU:n)

System on a chip (SOC)

Allt detta har gjort CPU:er komplicerade vilket möjliggjort Spectre och Meltdown

Subrutiner, procedurer, funktioner och metoder

Paradigm	parametrar	lokala variabler	returvärden
imperativ, strukturerad	globala	globala	globala
procedurell, objektorienterat	by value, by reference	lokala (scoping)	lokala
Deklarativt, Funktionellt, Logiskt	Immutable	Immutable	Immutable

Från Lat evaluering till Continuation passing style

Lat evaluering, vi behöver inte köra hela funktionen utan bara de bitar vi behöver. Haskell gör detta hela tiden.

Korutiner (coroutines) innebär att vi kan köra en funktion fram till en punkt och fortsätta därifrån.

Monader fullt ut: Streams och parallella streams i Java.

Exempel på Korutin (coroutine)

Korutin i Python

```
>>> def korutin():
...  while True:
...           food = yield
...           print("Eating " +
food + ". Mooo!")
...
>>>
>>> moo = korutin()
>>> next(moo)
>>> moo.send("Hay")
Eating Hay. Mooo!
>>> moo.send("Grass")
Eating Grass. Mooo!
>>>
```

Java Streams

Java:s Streams är en monad som bygger upp delresultat efter en uppsättning constraints på vägen. Exempel:

```
List<String> names = Arrays.asList("Marcus", "Sophia",
      "Magnus", "Gunnar");
names
      .parallelStream()
      //.stream()
      /*.filter(s -> {
          System.out.println("Filtering: " + s);
          return s.startsWith("M");})
       */
      .map(s -> {
          System.out.println("toUpper: "+ s);
          return s.toUpperCase();})
      .sorted()
      .map(s -> {
          System.out.println("toLower: " + s);
          return s.toLowerCase();
      })
      .forEachOrdered(System.out::println);
```

Java Streams, primtal

Mer exempel: primtal

```
IntStream
      .range(2,1000)
      .parallel()
      .filter(x->{for(int i = 2; i<=x/2; ++i){if(x%i==0)return false;}return true;})
      .forEachOrdered(s ->System.out.print(s + " "));
```