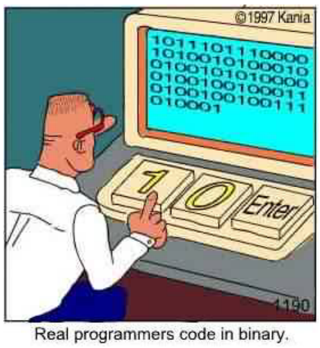
Software Visualization



Lecture
WS 02/03

Static
Program
Visualization

Static Program Visualization

- Text / Pretty Printing
- Program as Publication
- Jackson Diagrams
- Control Flow Graph (Flow charts)
- Nassi-Shneiderman Diagrams

Text / Pretty Printing

- Program Text = sequence of characters
- Pretty Printing:
  - originally use of indentation, line breaks to make the structure of a program more explicit.
  - With advance of technology also fonts, font face and colors are used, e.g.
    - bold face → keywords, italics for comments.
    - font size → nesting
    - tabbing → declarations
    - spacing → operator precedence

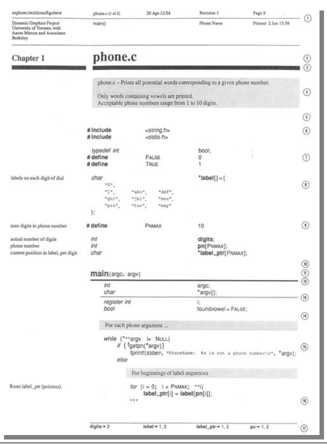
```
int i,c; while(i<100) if (i%2==0) c++; i++;
```

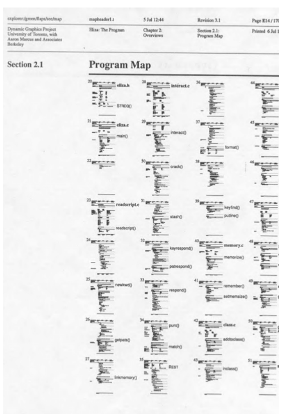
```
int i, c;

while (i<100)
  if (i%2 == 0) c++;
i++;
```


Program as Publication

- Program book [see Baker&Marcus:98]
  - Frontmatter
    - Cover page, title page
    - Abstract, program history
    - Authors
    - Table of contents
  - Chapter 1: User Documentation,
    - e.g. tutorial on how to use the program
  - Chapter 2: Overview
    - Program map (thumbnails of each program code page with major function emphasized).
    - Call hierarchy
  - One chapter per file
    - Pretty printed program code
    - Comments in margins
  - Chapter: Programmer Documentation
    - Installation and maintenance guides
  - Indices
    - Cross-reference
    - Caller index
    - Callee index
  - Back cover page
    - Highlights, summary

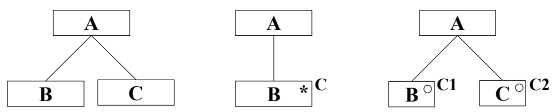
[see also „literate programming“, Knuth]





Jackson Diagrams [see Jackson:75]

- Basic elements are actions. Actions are decomposed into subactions.



Sequence:
A is C after B

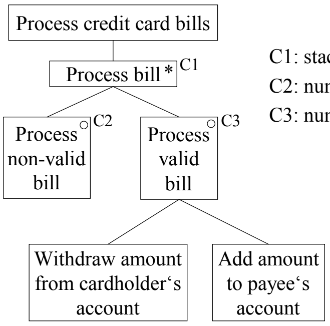
Iteration:
A is multiple repetitions of B

C = iteration condition

Alternative:
A is either B or C

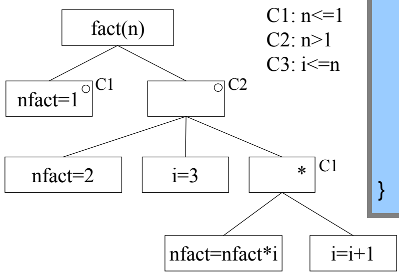
C1,C2 are conditions

Jackson Diagrams



C1: stack of bills is not empty
C2: number of credit card is not valid
C3: number of credit card is valid

Jackson Diagrams



C1: n<=1
C2: n>1
C3: i<=n

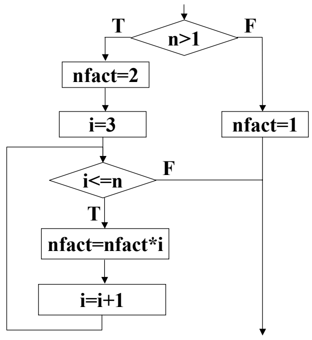
```
int fact(n)
 { if (n>1)
    { nfact=2;
      for(int i=3;i<=n;i++)
         nfact=nfact*i;
    }
    else
     { nfact=1;
     }
    return nfact;
}
```

- Jackson diagrams are more usable for design than for visualizing actual program code.

Control Flow Graph

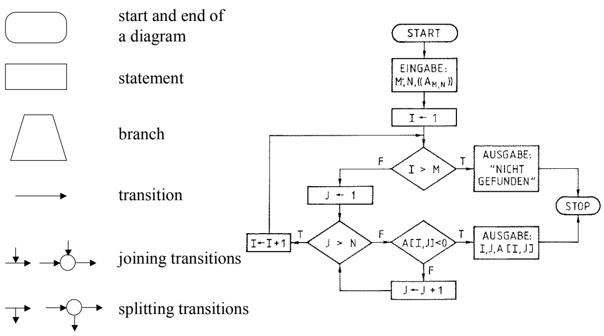
- Goldstine and von Neumann: 1946/47
  - Node: event/activity/process/function/statement
  - Diamond: branch condition (several exits)
  - Arrow: temporal order, transition → flow of control

Control Flow Diagrams

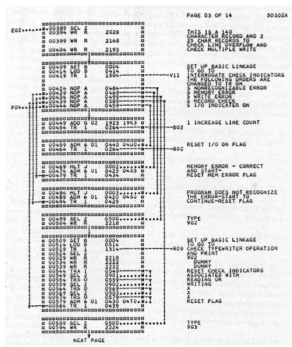



```
int fact(n)
 { if (n>1)
    { nfact=2;
      for(int i=3;i<=n;i++)
         nfact=nfact*i;
    }
    else
     { nfact=1;
     }
    return nfact;
}
```


Flowcharts: DIN 66001



Automatic Generation of CFGs



- A.E. Scott:58

Automatic Computation of CFG

- Syntax of a sample programming language
- Computation of CFGs
- A simple layout algorithm for CFGs

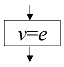
Syntax of a simple programming language

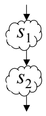
$$G = \{\ S \longrightarrow V\texttt{=}E \mid S\texttt{;}S \mid \texttt{if (}E\texttt{) \{}S\texttt{\} else \{}S\texttt{\}} \mid \texttt{while (}E\texttt{) \{}S\texttt{\}}$$
$$V \longrightarrow \text{Variablename}$$
$$E \longrightarrow \text{Expression}$$
$$\}$$

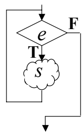
$L_G(A) = \{w \mid\ w$ is a terminal word with $A \xrightarrow{G} w\}$

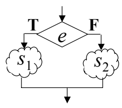
Informal Computation of CFGs

Let $s, s_i \in L_G(S), v \in L_G(V)$ and $e \in L_G(E)$

v=e 

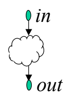
s_1;s_2 

while(e) {s} 

if(e) {s_1} else {s_2} 

Computation of a CFG

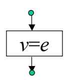
A CFG is a tuple (V, E, in, out) where



- V is a set of nodes,
- $E \subseteq V \times L \times V$ a set of edges,
- $L = \{\epsilon, \texttt{T}, \texttt{F}\}$ a set of labels,
- and $in, out \in V$ are start and end nodes.

Let Γ be the set of all control flow graphs.
We define a function cfg : $L_G(S) \longrightarrow \Gamma$ which maps programs to control flow graphs:
cfg$(w) = (V, E, in, out)$ where

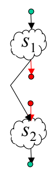
if $w = v$=e then



- $V = \{w, in, out\}$,
- $E = \{(in, \epsilon, w), (w, \epsilon, out)\}$,
- and in, out are two new nodes.

if $w = s_1$;s_2 then

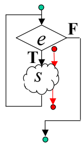
- $V = (V_1 - \{out_1\}) \cup (V_2 - \{in_2\})$,
- $E = (E_1 - \{(v, l, out_1) | v \in V_1\}) \cup (E_2 - \{(in_2, \epsilon, v) | v \in V_2\}) \cup \{(v_1, l_1, v_2) \mid (v_1, l_1, out_1) \in E_1, (in_2, \epsilon, v_2) \in E_2)\}$
- and $in = in_1, out = out_2$.



where $(V_1, E_1, in_1, out_1) =$ cfg(s_1)
and $(V_2, E_2, in_2, out_2) =$ cfg(s_2)

if $w = \texttt{while}(e)\{s\}$ then

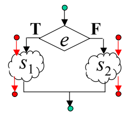
- $V = V_0 \cup \{e\}$,
- $E = (E_0 - (\{(in_0, \epsilon, v) | v \in V_0\} \cup \{(v, l, out_0) | v \in V_0\}))$
  $\cup\ \{(in_0, \epsilon, e), (e, F, out_0)\}$
  $\cup\ \{(e, T, v) | (in_0, \epsilon, v) \in E_0\}$
  $\cup\ \{(v, l, e) | (v, l, out_0) \in E_0\}$
- and $in = in_0, out = out_0$.



where $(V_0, E_0, in_0, out_0) = \text{cfg}(s)$

if $w = \texttt{if}(e)\{s_1\}\texttt{else}\{s_2\}$ then

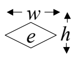
- $V = (V_1 - \{in_1, out_1\}) \cup (V_2 - \{in_2, out_2\}) \cup \{e\}$,
- $E = (E_1 - \{(in_1, \epsilon, v_1), (v_2, l, out_1) | v_1, v_2 \in V_1\}$
  $\cup\ (E_2 - \{(in_2, \epsilon, v_1), (v_2, l, out_2) | v_1, v_2 \in V_2\}$
  $\cup\ \{(in, \epsilon, e)\}$
  $\cup\ \{(e, T, v) | (in_1, \epsilon, v) \in E_1\}$
  $\cup\ \{(v, l, out) | (v, l, out_1) \in E_1\}$
  $\cup\ \{(e, F, v) | (in_2, \epsilon, v) \in E_2\}$
  $\cup\ \{(v, l, out) | (v, l, out_2) \in E_2\}$
- and in, out are two new nodes.



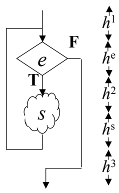
where $(V_i, E_i, in_i, out_i) = \text{cfg}(s_i)$

Simple Layout of CFGs

- We use rectangular boxes and place *in* and *out* nodes in the middle of the upper and lower borders.



The height depends on the font size and the width also on the length of the expression



Heights h^1, h^2 and h^3 are fixed.
So we can compute $H=h^1+h^2+h^3$ beforehand.
As a result we have $h = H+h^e+h^s$.

Simple Layout of CFGs

We define a function box : $L_G(S) \longrightarrow \mathcal{R} \times \mathcal{R}$ which maps programs to the sizes of the box to layout its control flow graph:

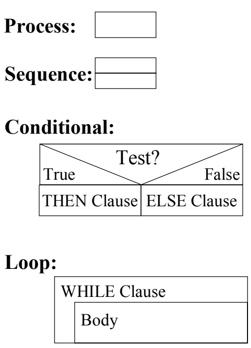
$\text{box}(e) = (w, h)$ where w, h depend on the font.

$\text{box}(s_1; s_2) = (W + \max(w_1, w_2), H + h_1 + h_2)$
where $(w_i, h_i) = \text{box}(s_i)$

$\text{box}(\texttt{while}(e)\{s\}) = (\ W + max(w_e, h_s),$
$H + h_e + h_s)$
where $(w_e, h_e) = \text{box}(e)$ and $(w_s, h_s) = \text{box}(s)$

Nassi-Shneiderman Diagrams

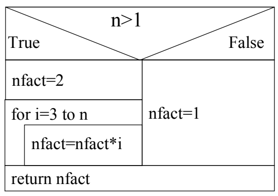
[see Nassi&Shneiderman:73]



- Also known as „structograms“
- „Not only does this notation help the programmer to think in an orderly manner, it forces him or her to do so.“
- „The absence of any representation of the GOTO or branch statement requires the programmer to work without it: a task which becomes increasingly easy with practice.“

Nassi-Shneiderman Diagrams

fact(n)




```
int fact(n)
 { if (n>1)
     { nfact=2;
        for(int i=3;i<=n;i++)
            nfact=nfact*i;
     }
     else
      { nfact=1;
      }
     return nfact;
}
```

Assignments

- We will collect the assignments at the start of next weeks lecture.

- http://www.cs.uni-sb.de/~diehl/SoftVisVorles
  - User:
  - Password: