Part II

Markup Basics

Outline of this part

Early Markup Languages

- The term **markup** has been coined by the **typesetting** community, *not* by computer scientists:
- With the advent of the printing press, writers and editors used (often marginal) notes to instruct printers to
  - select certain fonts,
  - let passages of text stand out,
  - indent a line of text, *etc.*
- Proofreaders use a special set of symbols, their special **markup language**, to identify typos, formatting glitches, and similar erroneous fragments of text.

  N.B. The markup language is designed to be easily recognizable in the actual flow of text.

Markup Languages | Early Markup

Early Markup Languages

- The term **markup** has been coined by the **typesetting** community, *not* by computer scientists:
- With the advent of the printing press, writers and editors used (often marginal) notes to instruct printers to
  - select certain fonts,
  - let passages of text stand out,
  - indent a line of text, *etc.*
- Proofreaders use a special set of symbols, their special **markup language**, to identify typos, formatting glitches, and similar erroneous fragments of text.

  N.B. The markup language is designed to be easily recognizable in the actual flow of text.

Torsten Grust (WSI) | Database-Supported XML Processors | Winter 2012/13 | 12

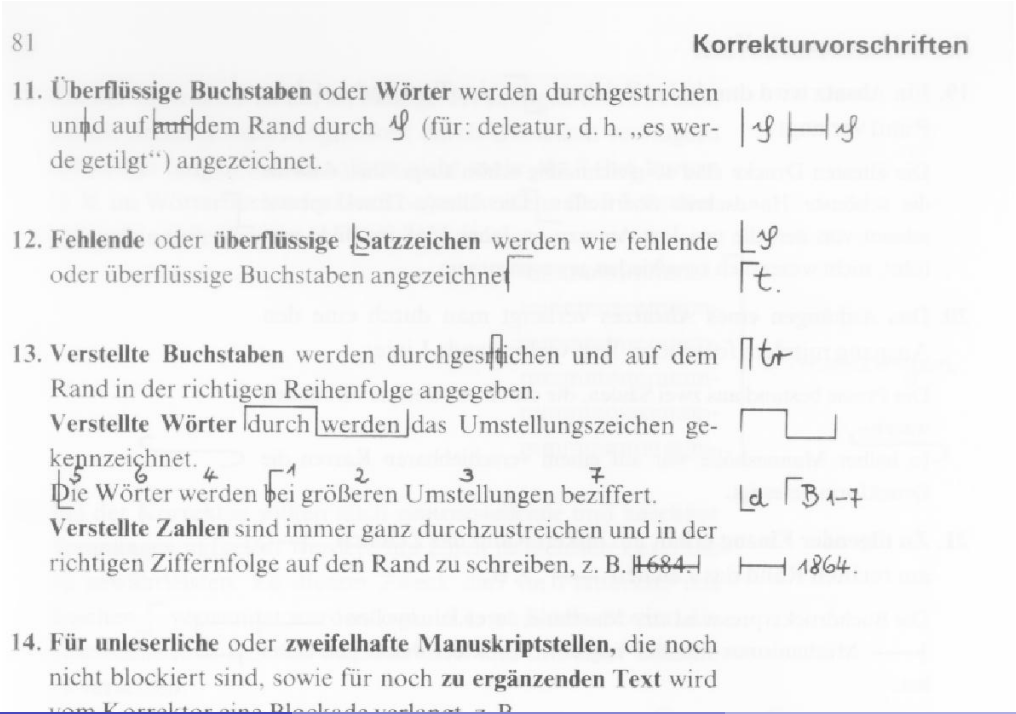
Early Markup Languages

- The term **markup** has been coined by the **typesetting** community, *not* by computer scientists:
- With the advent of the printing press, writers and editors used (often marginal) notes to instruct printers to
  - select certain fonts,
  - let passages of text stand out,
  - indent a line of text, *etc.*
- Proofreaders use a special set of symbols, their special **markup language**, to identify typos, formatting glitches, and similar erroneous fragments of text.

  N.B. The markup language is designed to be easily recognizable in the actual flow of text.

Example

Reproduced from the "Duden", 21st edition (1996), © Brockhaus AG.



81 **Korrekturvorschriften**

11. **Überflüssige Buchstaben** oder **Wörter** werden durchgestrichen und auf dem Rand durch ℘ (für: deleatur, d. h. „es werde getilgt") angezeichnet.

12. **Fehlende** oder **überflüssige Satzzeichen** werden wie fehlende oder überflüssige Buchstaben angezeichnet.

13. **Verstellte Buchstaben** werden durchgestrichen und auf dem Rand in der richtigen Reihenfolge angegeben.
Verstellte Wörter durch werden das Umstellungszeichen gekennzeichnet.
Die Wörter werden bei größeren Umstellungen beziffert.
Verstellte Zahlen sind immer ganz durchzustreichen und in der richtigen Ziffernfolge auf den Rand zu schreiben, z. B. ~~1684.~~ 1864.

14. **Für unleserliche** oder **zweifelhafte Manuskriptstellen,** die noch nicht blockiert sind, sowie für noch **zu ergänzenden Text** wird

- Computing Scientists adopted the markup idea—originally to **annotate program source code**:
  - Design the markup language such that its constructs are **easily recognizable** by a machine.
  - **Approaches**:
    1. Markup is written using a **special set of characters**, disjoint from the set of characters that form the tokens of the program.
    2. Markup occurs in places in the source file where program code may not appear (**program layout**).

- **Example** of ②: Fortran 77 fixed form source:
  - Fortran statements start in column 7 and do not exceed column 72,
  - a Fortran statement longer than 66 characters may be continued on the next line if a character $\notin$ { `0`, `!`, ␣} is placed in column 6 of the continuing line,
  - comment lines start with a `C` or $*$ in column 1,
  - numeric labels (`DO`, `FORMAT` statements) have to be placed in columns 1–5.

Fortran 77 source, fixed form, space characters made explicit (␣)

Fortran 77

```
1  C␣THIS␣PROGRAM␣CALCULATES␣THE␣CIRCUMFERENCE␣AND␣AREA␣OF␣A␣CIRCLE␣WITH
2  C␣RADIUS␣R.
3  C
4  C␣DEFINE␣VARIABLE␣NAMES:
5  C␣␣␣␣␣␣R:␣RADIUS␣OF␣CIRCLE
6  C␣␣␣␣␣␣PI:␣VALUE␣OF␣PI=3.14159
7  C␣␣␣␣␣␣CIRCUM:␣CIRCUMFERENCE␣=␣2*PI*R
8  C␣␣␣␣␣␣AREA:␣AREA␣OF␣THE␣CIRCLE␣=␣PI*R*R
9  ******************************************************************
10 C
11 ␣␣␣␣␣␣REAL␣R,CIRCUM,AREA
12 C
13 ␣␣␣␣␣␣PI␣=␣3.14159
14 C
15 C␣SET␣VALUE␣OF␣R:
16 ␣␣␣␣␣␣R␣=␣4.0
17 C
18 C␣CALCULATIONS:
19 ␣␣␣␣␣␣CIRCUM␣=␣2.*PI*R
20 ␣␣␣␣␣␣AREA␣=␣PI*R*R
21 C
22 C␣WRITE␣RESULTS:
23 ␣␣␣␣␣␣WRITE(6,*)␣␣'␣␣FOR␣A␣CIRCLE␣OF␣RADIUS',␣R,
24 ␣␣␣␣␣␣+␣␣␣␣␣␣␣␣␣␣␣␣'␣␣THE␣CIRCUMFERENCE␣IS',␣CIRCUM,
25 ␣␣␣␣␣␣+␣␣␣␣␣␣␣␣␣␣␣␣'␣␣AND␣THE␣AREA␣IS␣',␣AREA
26 C
27 ␣␣␣␣␣␣END
```

- Increased computing power and more sophisticated parsing technology made fixed form source obsolete.
- Markup, however, is still being used on different levels in today's programming languages and systems:
  - ASCII defines a set of **non-printable characters** (the C0 control characters, code range 0x00–0x1f):

code	name	description
0x01	`STX`	start of heading
0x02	`SOT`	start of text
0x04	`EOT`	end of transmission
0x0a	`LF`	line feed
0x0d	`CR`	carriage return

  - **Blocks** (containers) are defined using various form of matching **delimiters**:
    - `begin ... end`, `\begin{foo} ... \end{foo}`
    - `/* ... */`, `{ ... }`, `// ... LF`
    - `do ... done`, `if ... fi`, `case ... esac`, `$[ ... ]`

An Application of Markup: A Comic Strip Finder

Problem:

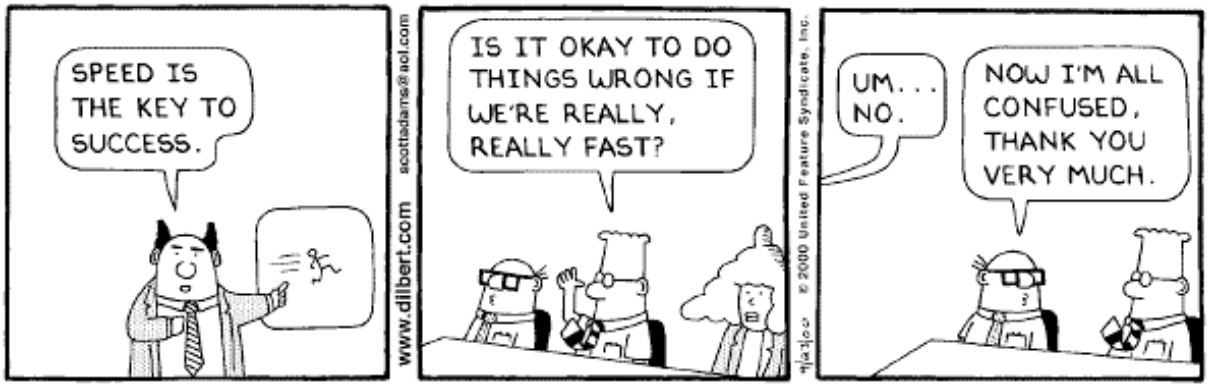
- **Query a database of comic strips by content**. We want to approach the system with queries like:
  1. *Find all strips featuring Dilbert but not Dogbert.*
  2. *Find all strips with Wally being angry with Dilbert.*
  3. *Show all strips featuring characters talking about XML.*

Approach:

- Unless we have next^n generation image recognition software available, we obviously have to **annotate** the comic strips to be able to process the queries above:

strips	bitmap	annotation
	⋮  ⋮	⋮ ...Dilbert...Dogbert Wally... ⋮

Stage 1: ASCII-Level Markup



ASCII-Level Markup

```
1  Pointy-Haired Boss: >>Speed is the key to success.<<
2  Dilbert: >>Is it okay to do things wrong if we're really, really fast?<<
3  Pointy-Haired Boss: >>Um... No.<<
4  Wally: >>Now I'm all confused.  Thank you very much.<<
```

- ASCII C0 character sequence 0x0d, 0x0a (`CR`, `LF`) divides lines,
- each line contains a character name, then a colon (`:`), then a line of speech (comic-speak: *bubble*),
- the contents of each bubble are delimited by >> and <<.

Markup Languages | Sample Markup Application

Stage 1: ASCII-Level Markup

ASCII-Level Markup

```
1  Pointy-Haired Boss: >>Speed is the key to success.<<
2  Dilbert: >>Is it okay to do things wrong if we're really, really fast?<<
3  Pointy-Haired Boss: >>Um... No.<<
4  Wally: >>Now I'm all confused.  Thank you very much.<<
```

- ASCII C0 character sequence 0x0d, 0x0a (`CR`, `LF`) divides lines,
- each line contains a character name, then a colon (`:`), then a line of speech (comic-speak: *bubble*),
- the contents of each bubble are delimited by >> and <<.

✎ Which kind of queries may we ask now?

And what kind of software do we need to complete the comic strip finder?

Stage 2: HTML-Style Physical Markup

dilbert.html

```
1  <h1>Dilbert</h1>
2  <h2>Panel 1</h2>
3  <ul>
4     <li> <b>Pointy-Haired Boss</b> <em>Speed is the key
5          to success.</em>
6  </ul>
7  <h2>Panel 2</h2>
8  <ul>
9     <li> <b>Dilbert</b> <em>Is it okay to do things wrong
10         if we're really really fast?</em>
11 </ul>
12 <h2>Panel 3</h2>
13 <ul>
14    <li> <b>Pointy-Haired Boss</b> <em>Um... No.</em>
15    <li> <b>Wally</b> <em>Now I'm all confused.
16         Thank you very much.</em>
17 </ul>
```


HTML: Observations

- HTML defines a number of markup **tags**, some of which are required to match (<*t*>...</*t*>).
- Note that HTML tags primarily describe **physical markup** (font size, font weight, indentation, ...)
- Physical markup is of limited use for the comic strip finder (the **tags do not reflect the structure of the comic content**).

Stage 3: XML-Style Logical Markup

- We create a **set of tags that is customized** to represent the content of comics, *e.g.*:

```
<character> Dilbert </character>
<bubble> Speed is the key to success. </bubble>
```

- New types of queries may require new tags: No problem for XML!
  - Resulting set of tags forms a new markup language (**XML dialect**).
- **All** tags need to appear in **properly nested** pairs (*e.g.*, <*t*> . . . <*s*> . . . </*s*>. . . </*t*>).
- Tags can be freely nested to reflect the **logical structure** of the comic content.

✎ Parsing XML?

In comparison to the stage 1 ASCII-level markup parsing, how difficult do you rate the construction of an XML parser?

Markup Languages | Sample Markup Application

Stage 3: XML-Style Logical Markup

- We create a **set of tags that is customized** to represent the content of comics, *e.g.*:

```
<character> Dilbert </character>
<bubble> Speed is the key to success. </bubble>
```

- New types of queries may require new tags: No problem for XML!
  - Resulting set of tags forms a new markup language (**XML dialect**).
- **All** tags need to appear in **properly nested** pairs (*e.g.*, <*t*> . . . <*s*> . . . </*s*>. . . </*t*>).
- Tags can be freely nested to reflect the **logical structure** of the comic content.

✎ Parsing XML?

In comparison to the stage 1 ASCII-level markup parsing, how difficult do you rate the construction of an XML parser?

Torsten Grust (WSI) | Database-Supported XML Processors | Winter 2012/13 | 21

Stage 3: XML-Style Logical Markup

- We create a **set of tags that is customized** to represent the content of comics, *e.g.*:

```
<character> Dilbert </character>
<bubble> Speed is the key to success. </bubble>
```

- New types of queries may require new tags: No problem for XML!
  - Resulting set of tags forms a new markup language (**XML dialect**).
- **All** tags need to appear in **properly nested** pairs (*e.g.*, <*t*> . . . <*s*> . . . </*s*>. . . </*t*>).
- Tags can be freely nested to reflect the **logical structure** of the comic content.

✎ Parsing XML?

In comparison to the stage 1 ASCII-level markup parsing, how difficult do you rate the construction of an XML parser?

In our example

dilbert.xml

```
1  <strip>
2    <panel>
3      <speech>
4        <character>Pointy-Haired Boss</character>
5        <bubble>Speed is the key to success.</bubble>
6      </speech>
7    </panel>
8    <panel>
9      <speech>
10       <character>Dilbert</character>
11       <bubble>Is it okay to do things wrong
12               if we're really, really fast?</bubble>
13     </speech>
14   </panel>
15   <panel>
16     <speech>
17       <character>Pointy-Haired Boss</character>
18       <bubble>Um... No.</bubble>
19     </speech>
20     <speech>
21       <character>Wally</character>
22       <bubble>Now I'm all confused.
23               Thank you very much.</bubble>
24     </speech>
25   </panel>
26 </strip>
```

Stage 4: Full-Featured XML Markup

- Although fairly simplistic, the previous stage clearly constitutes an improvement.
- XML comes with a number of additional constructs which allow us to convey even more useful information, *e.g.*:
  - **Attributes** may be used to qualify tags (avoid the so-called *tag soup*). Instead of
    - `<question>Is it okay ...?</question>`
      `<angry>Now I'm ...</angry>`

    use
    - `<bubble tone="question">Is it okay ...?</bubble>`
      `<bubble tone="angry">Now I'm ...</bubble>`
  - **References** establish links internal to an XML document:
    Establish link target:
    - `<character id="phb">The Pointy-Haired Boss</character>`

    Reference the target:
    - `<bubble speaker="phb">Speed is the key to success.</bubble>`

Markup Languages | Sample Markup Application

Stage 4: Full-Featured XML Markup

- Although fairly simplistic, the previous stage clearly constitutes an improvement.
- XML comes with a number of additional constructs which allow us to convey even more useful information, *e.g.*:
  - **Attributes** may be used to qualify tags (avoid the so-called *tag soup*). Instead of
    - `<question>Is it okay ...?</question>`
      `<angry>Now I'm ...</angry>`

    use
    - `<bubble tone="question">Is it okay ...?</bubble>`
      `<bubble tone="angry">Now I'm ...</bubble>`
  - **References** establish links internal to an XML document:
    Establish link target:
    - `<character id="phb">The Pointy-Haired Boss</character>`

    Reference the target:
    - `<bubble speaker="phb">Speed is the key to success.</bubble>`

Torsten Grust (WSI) | Database-Supported XML Processors | Winter 2012/13 | 23

dilbert.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<strip copyright="United Feature Syndicate" year="2000">
  <prolog>
    <series href="http://www.dilbert.com/">Dilbert</series>
    <author>Scott Adams</author>
    <characters>
      <character id="phb">The Pointy-Haired Boss</character>
      <character id="dilbert">Dilbert, The Engineer</character>
      <character id="wally">Wally</character>
      <character id="alice">Alice, The Technical Writer</character>
    </characters>
  </prolog>
  <panels length="3">
    <panel no="1">
      <scene visible="phb">
        Pointy-Haired Boss pointing to presentation slide.
      </scene>
      <bubbles>
        <bubble speaker="phb">Speed is the key to success.</bubble>
      </bubbles>
    </panel>
    <panel no="2">
      <scene visible="wally dilbert alice">
        Wally, Dilbert, and Alice sitting at conference table.
      </scene>
      <bubbles>
        <bubble speaker="dilbert" to="phb" tone="question">
          Is it ok to do things wrong if we're really, really fast?
        </bubble>
      </bubbles>
    </panel>
    <panel no="3">
      <scene visible="wally dilbert">Wally turning to Dilbert, angrily.
      </scene>
      <bubbles>
        <bubble speaker="phb" to="dilbert">Um... No.</bubble>
        <bubble speaker="wally" to="dilbert" tone="angry">
          Now I'm all confused.  Thank you very much.
        </bubble>
      </bubbles>
    </panel>
  </panels>
</strip>
```