Tecnologie Web e Laboratorio: XML

Fabio Buttussi

Dipartimento di Matematica, Informatica e Fisica

Università degli Studi di Udine

Crediti

- Queste slide sono un aggiornamento del materiale del dott. Andrea Urgolo.

- Non è consentita la diffusione, ma solo l’utilizzo per lo studio personale.

Editor

- Per modificare un file XML è sufficiente un editor di testo come il blocco note, ma editor per linguaggi di programmazione e scripting offrono utili funzioni come il syntax highlighting, l'indentazione automatica e/o l'auto-completamento.
- Tre editor open source sono:
  - Notepad++ (https://notepad-plus-plus.org/)
  - Brackets (http://brackets.io/)
  - Visual Studio Code (https://code.visualstudio.com/)
- Un ambiente di sviluppo integrato (IDE) gratuito è:
  - Visual Studio Community (https://visualstudio.microsoft.com/it/vs/community/)
- Esistono anche editor online.

Materiale

- Documentazione ufficiale:
  - Raccomandazione del W3C: Extensible Markup Language (XML) 1.0 (Fifth Edition) https://www.w3.org/TR/xml
- Siti con materiale didattico:
  - https://w3schools.com/xml
- Libro di testo:
  - XML in a Nutshell, 3rd Edition by Elliotte Rusty Harold, W. Scott Means, Pub.:O'Reilly Media, Inc. ISBN: 9780596007645

Linguaggi diversi per usi diversi

PER DEFINIRE PROCEDURE/ALGORITMI	PER DEFINIRE E STRUTTURARE I DATI	PER VISUALIZZZARE I DATI
Python	XML	CSS
Java	HTML	XSL
PHP		XSLT
Javascript		
C#		
ecc.		

Linguaggi di markup

- L'uso del markup permette di strutturare un file in formato testo (documento) in componenti logiche, dette **elementi**, e di "etichettarle" opportunamente.
- Le etichette (i *nomi* degli elementi) specificano il tipo di dato che una certa componente logica rappresenta.
- Le etichette vengono inserite nel documento stesso come speciali sequenze di caratteri, dette *markup tag* o semplicemente *tag*.

Esempio 1 di documento senza markup

Jack Smith

513-555-3465

jsmith@email.com

John Doe

34 Fountain Square Plaza

Cincinnati, OH 45202

US

513-555-8889 (preferred)

513-555-7098

jdoe@email.com

Esempio 1 di documento con markup

```
<address book>
    <contact>
        <name>Jack Smith</name>
        <tel>513-555-3465</tel>
        <email address='jsmith@email.com'/>
    </contact>
    <contact>
        <name>John Doe</name>
        <address>34 Fountain Square Plaza Cincinnati, OH
                45202 US</address>
        <tel>513-555-8889</tel>
        <tel>513-555-7098</tel>
        <email address='jdoe@email.com'/>
    </contact>
</address book>
```

Esempio 2 di documento senza markup

Internet

1. Il World Wide Web

Il World Wide Web (cui spesso ci si riferisce semplicemente con Web o con l'acronimo WWW) è stata cronologicamente l'ultima funzionalità di Internet ad essere sviluppata.

…

2. La posta elettronica

…

Esempio 2 di documento con markup

```
<libro>
    <capitolo><titolo>Internet</titolo>
    <sezione><titolo> 1. il World Wide Web </titolo>
    <paragrafo>Il World Wide Web (cui spesso ci si riferisce
    semplicemente con Web o con l'acronimo WWW) è stata
    cronologicamente l'ultima funzionalità di Internet ad essere
    sviluppata. </paragrafo>
    </sezione>
    <sezione><titolo> 2. La posta elettronica</titolo>
    ...
    </sezione>
    </capitolo>
</libro>
```

Vantaggi dei linguaggi di markup

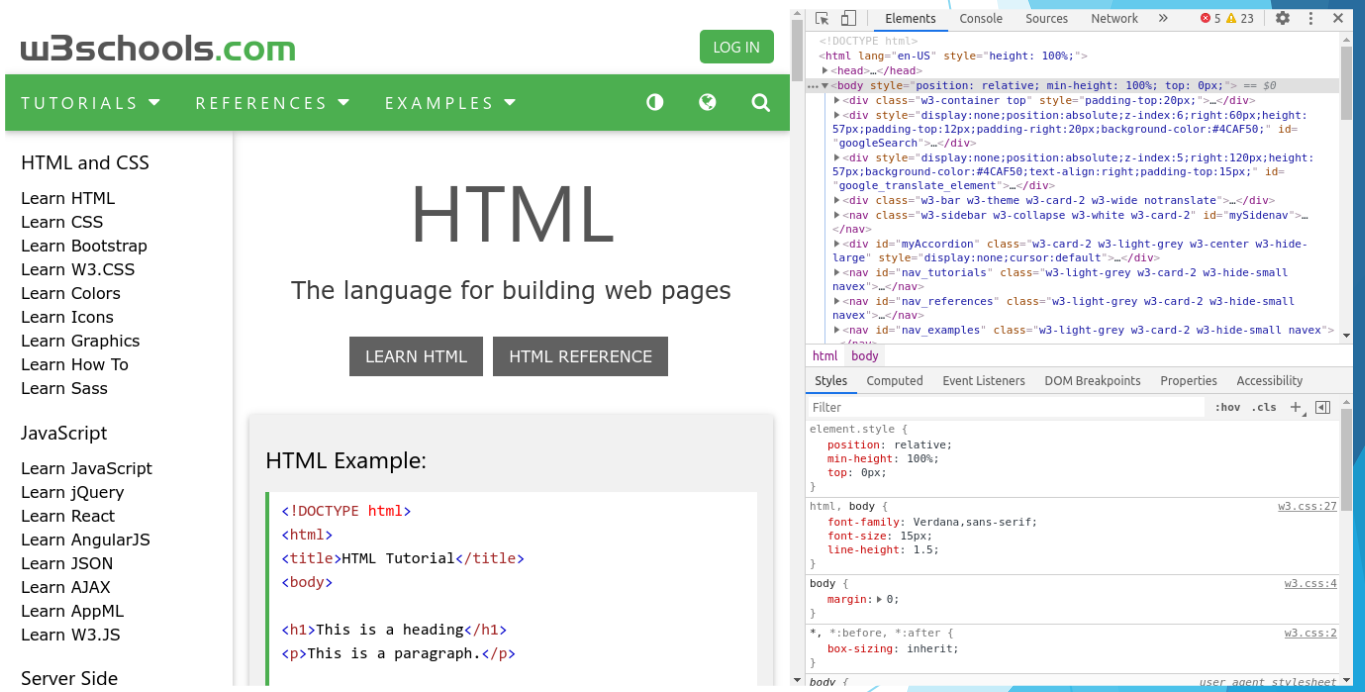
- Un programma può facilmente distinguere tra le componenti logiche di un documento con markup, es.
  - per scopi di visualizzazione: elementi diversi possono essere visualizzati a schermo o stampati usando stili tipografici diversi;
  - elaborazione e interrogazione dei dati rappresentati nel documento.
- Costituiscono un formato semplice per lo scambio di dati (in formato testo).
- Sono comprensibili anche agli umani.

Uso dei linguaggi di markup

- Dagli anni '70 fino agli anni '90: principalmente nel trattamento di testi e nell'editoria elettronica, es. TeX, RTF (Rich Text Format)
- Negli anni '90: pagine Web (HTML)
- Dalla fine degli anni '90: pagine Web, news, MMS, transazioni finanziarie, scambio di dati, disegni tecnici, dati geografici, fogli elettronici, documenti multimediali, oggetti e ambienti tridimensionali,...

Esempi 1/3

- HyperText Markup Language (HTML): linguaggio di markup per la definizione di ipertesti.
- Nasce nel 1993, è una delle basi del WWW.
- Nelle prime versioni markup tag impiegati anche per controllare la visualizzazione del documento.
- HTML5: tag strutturali (presentazione con fogli di stile CSS).

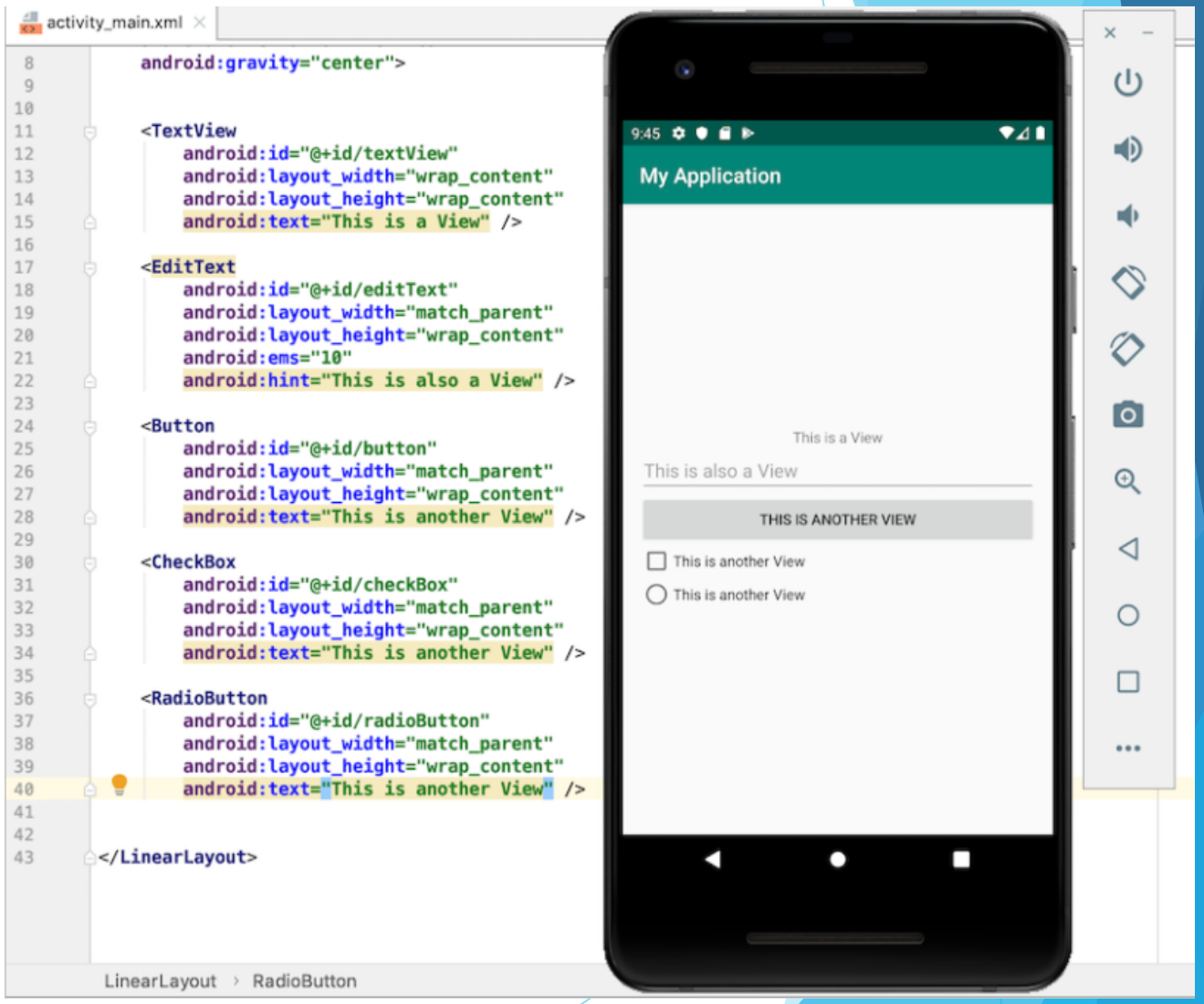


Esempi 2/3

- Scalable Vector Graphics (SVG) linguaggio di markup per la definizione di grafica bidimensionale (www.w3.org/Graphics/SVG/)
- Synchronized Multimedia Integration Language (SMIL): linguaggio di markup per la definizione di presentazioni multimediali (www.w3.org/AudioVideo/)
- Extensible 3D (X3D): linguaggio di markup per la definizione di oggetti e ambienti tridimensionali (www.web3d.org/)

Esempi 3/3

- Definizione GUI:
  - App Android
  - App Universal Windows Platform (XAML)
  - Giochi Unity (nuova versione UI, UXML)



Cos’è XML

- XML è l'acronimo di eXtensible Markup Language
  - **linguaggio formale** definito tramite un insieme di regole (una grammatica) che permettono di specificare come comporre un documento
  - informazioni sul significato del contenuto presente nel documento vengono fornite tramite annotazioni realizzate con codice di **markup** sintatticamente distinguibile
  - l'insieme dei tag di marcatura può essere **esteso** per soddisfare diverse esigenze e adattarsi a diversi domini applicativi
- XML è un **meta-linguaggio** di markup, ovvero un linguaggio per la costruzione di linguaggi di markup più specifici (ad es. XHTML)

Cosa non è XML

- XML non è un:
  - linguaggio di presentazione come SMIL, SVG, HTML (prime versioni): il codice di markup utilizzato nei documenti XML definisce il significato del contenuto, non dice nulla sullo stile con cui va presentato
  - linguaggio di programmazione come C/C++, C#, Java, Python: un documento XML non effettua nessun tipo di computazione
  - protocollo di trasmissione come HTTP: XML non dà indicazioni su come trasferire informazioni attraverso la rete
  - database management system come MySQL, PostgreSQL, Oracle, MS SQL Server: XML non fornisce funzionalità per l'archiviazione e il recupero di dati

Storia

- XML è un discendente dello Standard Generalized Markup Language (SGML) inventato da Goldfarb, Mosher e Lorie dell'IBM negli anni '70 e diventato standard (ISO 8879) nel 1986.
- SGML è un linguaggio di markup semantico e strutturale per documenti testuali complesso ed espressivo. Impiegato dalle forze armate e dal governo US in domini che richiedevano la gestione di lunghi documenti tecnici.
- In 1996, Bosak, Bray, McQueen, Clark e altri iniziarono a lavorare su una versione "lite" di SGML che manteneva la maggior parte del potere espressivo, tagliando caratteristiche ridondanti, complicate, confuse o non utili.
- Nel febbraio 1998, le specifiche di XML 1.0 divennero una raccomandazione ufficiale del W3C.
- Presto ci si accorse che XML non era limitato al contesto Web, ma era impiegabile in vari contesti, dalla definizione della struttura di documenti, allo scambio delle informazioni tra sistemi, dalla rappresentazione di immagini alla definizione di formati di dati.

Elementi

- Sono "contenitori" che incapsulano altri elementi o dati intesi come stringhe di caratteri (character data)
  - es.: <address ><street>...</street></address>
  - es.: <tel>513-555-7098</tel>
- Un elemento ha un nome e un contenuto
  - nel secondo es.: nome: tel, contenuto: 513-555-7098
- Il contenuto è delimitato da un **tag di inizio** e da un **tag di fine**
- Il tag di inizio contiene il nome circondato da parentesi angolari (< >)
- il tag di fine contiene a sua volta il nome tra parentesi angolari, ma preceduto da uno "slash" (/)

Nomi di elementi validi

- Devono iniziare con una lettera o con "_" (underscore)
- Gli altri caratteri del nome possono essere lettere, cifre, "_", "." e "-" (gli spazi non sono ammessi)
- Non possono iniziare con la stringa "xml"

- Sono *case sensitive*, cioè scrivere in maiuscolo o minuscolo fa differenza
  - es.: <address> e <ADDRESS> sono due elementi diversi

Attributi (1/2)

- Informazioni aggiuntive che vengono associate agli elementi
- Hanno un **nome** e un **valore** (per il nome valgono le regole viste nel caso degli elementi)
- Vengono inseriti all'interno del tag di inizio, dopo il nome dell'elemento
  - es.: <tel preferred="true">513-555-8889</tel>
  - nome dell'attributo: preferred
  - valore dell'attributo: true

Attributi (2/2)

- Per ogni elemento si possono inserire zero, uno o più attributi (separati da spazi)
  - es.: <tel preferred="true" location="home">513-555-8889<tel>
- Il valore può essere indicato tra due doppi apici ("...")
  - es.: <tel preferred="true">
- oppure tra due apici singoli ('...')
  - es.: <tel preferred='true'>

Esercizio

- Scrivere un documento XML che rappresenti una playlist musicale
- Possibile soluzione: Esempi/XML/Playlist.xml

Entità

- Un'entità è un riferimento ad una sequenza di caratteri
- Ha un **nome** e un **contenuto** (una sequenza di caratteri)
- Per inserire un'entità si utilizza il codice **&nome-entità;** (entity reference), cioè il nome dell'entità tra '&' e ';'
- Si può immettere solo nel contenuto degli elementi e nel valore degli attributi ed è equivalente ad inserire nella medesima posizione il contenuto riferito dall'entità
- Es.: se assumiamo di aver definito l'entità us con contentuto United States, allora sono equivalenti
  - `<state>&us;</state>`
  - `<state>United States</state>`

Entità predefinite (1/2)

- Caratteri impiegati nel codice di markup (in questo caso è obbligatorio usare l'entità)

Riferimento all'entità	Contenuto dell'entità
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	'

Entità predefinite (2/2)

- Ogni carattere può essere sostituito dal suo codice UNICODE (sia in formato decimale che esadecimale)
- I riferimenti che iniziano con &#x usano il formato esadecimale, i riferimenti che iniziano con &# usano il formato decimale
- Es.:
  - `<scuola>Universit&#x00E0;</scuola>`
- è equivalente a
  - `<scuola>Università</scuola>`
- Lista codici UNICODE in formato esadecimale: http://www.unicode.org/charts

Dichiarazione

- La dichiarazione XML identifica il documento come un documento XML
- La forma più semplice di dichiarazione XML è **<?xml version="1.0"?>**
- Non è obbligatoria, ma se presente deve iniziare nel **primo carattere** della **prima riga** del documento
- È consigliato inserirla sempre
- Attributi opzionali
  - **encoding**: specifica la codifica dei caratteri adottata (default: UTF 8)
  - **standalone** (yes | no): se yes, il documento è auto-contenuto, il che significa che tutti i suoi valori sono presenti in esso (dichiarazioni entità e valori di default); se no (default), possono essere specificati valori del documento in un DTD esterno.
- Es.: <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

Commenti

- È possibile inserire dei commenti in un documento XML (ovvero, del testo che NON fa parte del documento, ma è utile a chi lo ha scritto o lo deve modificare)

- La sintassi dei commenti è:
- `<!-- Questo è un commento -->`

- Attenzione: i commenti non possono essere inseriti all'interno del markup!

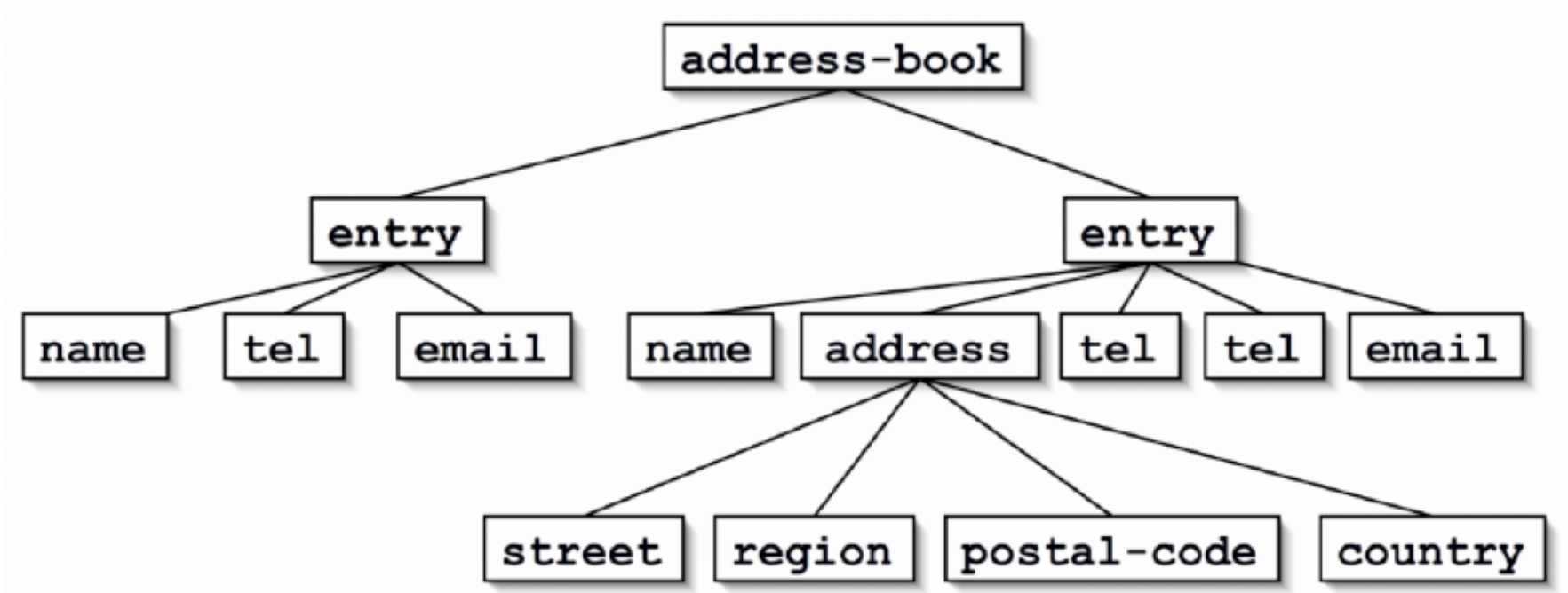
Documento XML ben formato

- Un documento XML si dice ben formato se:
  - ogni tag di inizio ha un corrispettivo tag di fine
  - i valori degli attributi sono delimitati da apici o doppi apici
  - un elemento non può avere due attributi con lo stesso nome
  - gli elementi sono correttamente annidati
  - c'è esattamente un "elemento radice"
  - vale quanto detto in precedenza: validità nomi, escaping caratteri di markup, commenti non all'interno del markup,…

Struttura logica

- Un documento XML ben formato è rappresentabile come un albero di elementi in cui:
  - la radice dell'albero è il primo elemento del documento
  - se l'elemento x è contenuto nell'elemento y, allora x è figlio di y e y è il genitore di x
  - le foglie sono quindi gli elementi che non contengono altri elementi

Esempio (albero)



Esempio (XML)

```
<address-book>
    <entry>
        <name>Jack Smith</name>
        <tel>513-555-3465</tel>
        <email address='jsmith@email.com'></email>
    </entry>
    <entry>
        <name>John Doe</name>
        <address>
            <street>34 Fountain Square Plaza</street>
            <region>Cincinnati, OH</region>
            <postal-code>45202</postal-code>
            <country>US</country>
        </address>
        <tel>513-555-8889</tel>
        <tel>513-555-7098</tel>
        <email address='jdoe@email.com'></email>
    </entry>
</address-book>
```

Annidamento corretto

- I figli devono essere completamente contenuti nei genitori
- Non è possibile che il tag di fine di un figlio appaia dopo il tag di fine di un genitore

- es. errato:
- `<name><fname>Jack</fname><lname>Smith</name></lname>`

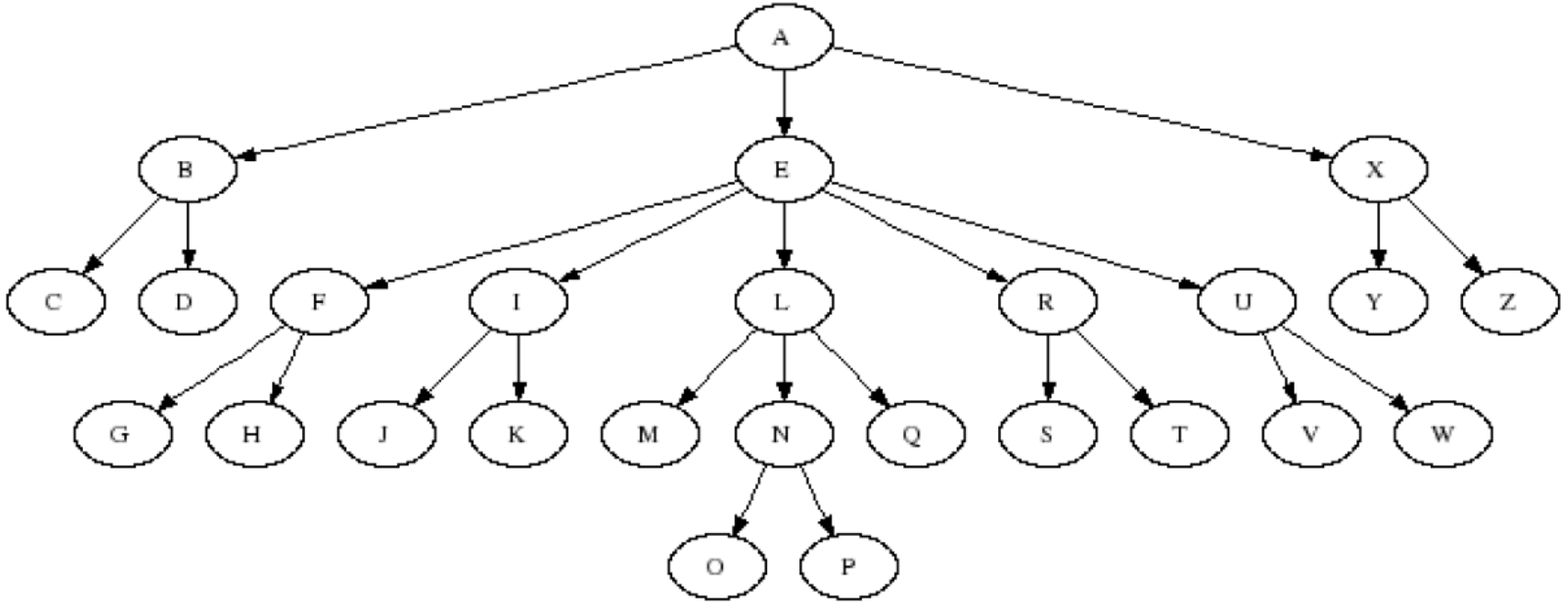
Unicità della radice

- Solo un elemento del documento XML può essere la radice dell'albero, e tutti gli altri elementi devono essere suoi discendenti

- es. errato:
- `<?xml version="1.0"?>`
- `<entry><name>John Doe</name></entry>`
- `<entry><name>Jack Smith</name></entry>`

Esercizio

- Scrivere l'XML equivalente all'albero in figura



- Soluzione: Esempi/XML/Albero.xml

Document Type Definition (DTD)

- Il Document Type Definition (DTD) dichiara come un certo "tipo" di documento XML deve essere scritto
- Es. che elementi e attributi si possono usare, che contenuto possono avere i diversi elementi,…

- Se un documento XML soddisfa le regole di un certo DTD si dice **valido** rispetto a quel DTD

Composizione di un DTD

- Un DTD è un documento in formato testuale composto da una lista di dichiarazioni
- Ogni dichiarazione è relativa a un "oggetto" che può essere contenuto nel tipo di documento XML considerato
- Ogni "oggetto" (elementi, attributi,…) usabile in quel tipo di documento va definito
- Il DTD segue una sintassi formale ben definita

Dichiarazione di elementi (1/5)

- <!ELEMENT nome_el contenuto_el>

- nome_el è il nome dell'elemento
- contenuto_el specifica il contenuto dell'elemento, es.
  - vuoto
  - testo
  - altri elementi

Dichiarazione di elementi (2/5)

- <!ELEMENT nome_el EMPTY>
- Dichiara un elemento vuoto

- Es. <!ELEMENT email EMPTY>
- nel documento XML, è corretto:
  - <email/> o <email></email>
- non è corretto:
  - <email>john@email.com</email>
- oppure
  - <email><home>john@email.com</home></email>

Dichiarazione di elementi (3/5)

- <!ELEMENT nome_el (#PCDATA)>
- Dichiara un elemento che può contenere solo testo (di solito si usa per elementi foglia)

- Es. <!ELEMENT name (#PCDATA)>
- nel documento XML, è corretto:
  - <name>John Doe</name>
- non è corretto:
  - <name><fname>John</fname></name>

Dichiarazione di elementi (4/5)

- <!ELEMENT nome_el ANY>
- Dichiara un elemento che può contenere qualunque altro elemento dichiarato nel DTD (si usa raramente)

- Es. <!ELEMENT entry ANY>
- nel documento XML, è corretto mettere nel contenuto di entry qualunque altro elemento dichiarato nel DTD
- è scorretta qualsiasi altra cosa

Dichiarazione di elementi (5/5)

- <!ELEMENT nome_el (expr)>
- Dichiara un elemento che deve contenere altri elementi, in un certo ordine e numero definiti dall'espressione regolare *expr*

- Es. <!ELEMENT address-book (entry)>
- nel documento XML è necessario inserire l'elemento entry dentro ogni elemento address-book
- Es. corretto <address-book><entry>...</entry></address-book>
- Es. non corretto: <address-book><a></address-book>

Indicatori di occorrenza

- Come si definisce la **numerosità** degli elementi contenuti nell'elemento dichiarato?
- In expr si utilizzano i simboli **+**, ***** e **?**, i quali sono **indicatori di occorrenza** degli elementi
- se il nome di un elemento è senza alcun simbolo di occorrenza, deve apparire una e una sola volta
  - es.: <!ELEMENT address-book (entry)>
- un elemento seguito da + deve apparire una o più volte
  - es.: <!ELEMENT address-book (entry+)>
- un elemento seguito da * deve apparire zero o più volte
  - es.: <!ELEMENT address-book (entry*)>
- un elemento seguito da ? deve apparire zero o una volta
  - es.: <!ELEMENT address-book (entry?)>

Esempi

- <!ELEMENT address-book (entry?)>
- Ogni elemento address book del documento deve contenere zero oppure un elemento entry

- <!ELEMENT address-book (entry+)>
- Ogni elemento address book del documento deve contenere uno o più elementi entry
- es.:

<address-book>

<entry>…</entry><entry>…</entry>…

<address-book>

Indicatori di ordine (1/2)

- Come si definisce l'ordine degli elementi contenuti nell'elemento dichiarato?
- In expr si utilizzano i simboli e , e |
- , permette di definire una sequenza di elementi
- es.: <!ELEMENT address (street, city, state)>
- corretto:
  - `<address><street>…</street><city>…</city><state>…</state></address>`
- non corretto:
  - `<address><state>…</state><city>…</city><street>…</street></address>`
- non corretto:
  - `<address><street>…</street><city>…</city></address>`

Indicatori di ordine (2/2)

- | permette di definire elementi da inserire in alternativa
- es.: <!ELEMENT address (street | city | state)>
- corretto:
  - `<address><street>…</street></address>`
- corretto:
  - `<address><city>…</city></address>`
- corretto:
  - `<address><state>…</state></address>`
- non corretto:
  - `<address><state>…</state><city>…</city><street>…</street></address>`

Raggruppamento di elementi

- È possibile usare le parentesi tonde per raggruppare elementi in expr
- Es.: <!ELEMENT name (lname , (fname | title))>

- È possibile dichiarare un contenuto misto
- Es. <!ELEMENT name (#PCDATA | (fname , lastname))*>

Esercizio

- Scrivere il DTD per l'XML di address-book
- Soluzione: Esempi/XML/Address-Book.dtd

Dichiarazione di attributi

- <!ATTLIST nome_el nome_attr tipo_attr uso_attr>

- nome_el è il nome dell'elemento in cui l'attributo si può usare
- nome_attr è il nome dell'attributo
- tipo_attr fornisce informazioni sul valore dell'attributo
- uso_attr fornisce informazioni sull'utilizzo dell'attributo

Uso dell’attributo

- Valori possibili per uso_attr:
  - **#REQUIRED** il valore dell'attributo deve essere inserito nel documento
  - **#IMPLIED** l’attributo è opzionale; non va fornito un valore di default
  - **#FIXED** 'val': il valore dell'attributo è sempre val
  - **'val'** l'attributo, se omesso, ha il valore di default val

Tipo dell’attributo (1/2)

- Valori possibili per tipo_attr:
  - **CDATA**: il valore dell'attributo è una sequenza di caratteri arbitraria
  - **ID**: un identificatore univoco per l'attributo; il valore dell'attributo è un nome (e deve seguire le regole viste sui nomi XML) che deve essere unico nel documento (lo stesso valore non può essere assegnato a più elementi)
  - **IDREF**: contiene il riferimento all’attributo ID di un elemento nel documento
  - **IDREFS**: contiene riferimenti a più attributi ID di elementi nel documento

Tipo dell’attributo (2/2)

- **NMTOKEN**: name token, il valore dell'attributo è una parola senza spazi che può contenere lettere, numeri e i simboli '-', '_', '.' e ':'
- **NMTOKENS**: il valore dell’attributo è un elenco di nmtoken separati da spazi
- **(n1 | n2 | … | nk)**: il valore dell'attributo deve essere uno fra i nmtoken n1, n2, …, nk
- **ENTITY**, **ENTITIES**, **NOTATION**: riferimenti a entità, elenco di riferimenti a entità separati da spazi e informazioni su formati interpretabili da applicazioni esterne (poco usati), es.:

```
<!NOTATION gif SYSTEM "image/gif">
<!NOTATION jpeg SYSTEM "image/jpeg">
<!NOTATION png SYSTEM "image/png">
<!ATTLIST image type NOTATION (gif | jpeg | png) #REQUIRED>
```

Esempi

- `<!ATTLIST tel preferred (true | false) "false">`
- Il valore di preferred (attributo dell'elemento tel) può essere solo true oppure false e se omesso vale false

- `<!ATTLIST email href CDATA #REQUIRED>`
- Il valore di href (attributo dell'elemento email) può essere qualsiasi sequenza di caratteri e deve essere specificato in ogni elemento email

Attributi multipli

- È possibile definire più attributi per lo stesso elemento usando un'unica dichiarazione di tipo ATTLIST

```
<!ATTLIST nome_el nome_attr1 tipo_attr1
uso_attr1 nome_attr2 tipo_attr2 uso_attr1… >
```

- Es.
```
<!ATTLIST tel prefix CDATA #REQUIRED
          preferred (true | false) "false">
```

Esercizio

- Scrivere il DTD per il file Esempi/XML/Playlist.xml
- Soluzione: Esempi/XML/Playlist.dtd

Collegare un XML a un DTD

- Per collegare un documento XML a un DTD, inserire nel documento XML (sulla riga successiva all'eventuale dichiarazione XML) una document type declaration

```
<!DOCTYPE el_radice DTD>
```

- dove DTD può essere
  - interno: incluso nel documento stesso tra [e]
  - esterno: un riferimento alla risorsa in cui il DTD è contenuto: SYSTEM dtd_URL
  - esterno pubblico: un riferimento PUBLIC dtd_URN (ad es. "-//Uniud//Address Book/EN"), seguito da un riferimento alla risorsa (dtd_URL)
  - sia interno, che esterno: parte del DTD tra [] e parte in un file esterno (indicato tramite SYSTEM o PUBLIC)
- Es.: `<!DOCTYPE el_radice SYSTEM "URL" [DTD]>`

Esempi

- ```
  <!DOCTYPE address-book [
   <!ELEMENT address-book (entry+)>
   <!ELEMENT entry (name, address?, tel+, email*)>
   <!ELEMENT name (#PCDATA)>
   ...
   ]>
  ```
- ```
  <!DOCTYPE address-book SYSTEM "address-book.dtd">
  ```
- ```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
  ```
  ```

Documento XML valido

- Include una document type declaration che identifica il DTD da soddisfare
- Rispetta quanto viene dichiarato nel DTD
- Principio di validità: tutto ciò che non è permesso è vietato
  - il DTD elenca tutti gli elementi, attributi ed entità che possono essere usati nel documento e loro contesto d'impiego
  - non si possono usare elementi, attributi ed entità non dichiarati nel DTD
- Validatori
  - https://www.xmlvalidation.com, https://www.truugo.com/xml_validator
  - xmllint http://xmlsoft.org/downloads.html, parser XML chiamabile da riga di comando es.: `xmllint --valid --noout documento.xml`
  - Validatore integrato in Visual Studio Community