Exercise 4

An identifier in a programming language is allowed to contain between 1 and 6 letters and digits, the first character must be a letter. Only upper case character are used. Write an ABNF specification for the syntax of such an identifier.

RFC 822 lexical scanner 1


```
CHAR        =  <any ASCII character>          ; (  0-177,  0.-127.)
ALPHA       =  <any ASCII alphabetic character>
                                              ; (101-132, 65.- 90.)
                                              ; (141-172, 97.-122.)
DIGIT       =  <any ASCII decimal digit>      ; ( 60- 71, 48.- 57.)
CTL         =  <any ASCII control             ; (  0- 37,  0.- 31.)
                character and DEL>            ; (    177,    127.)
CR          =  <ASCII CR, carriage return>    ; (     15,     13.)
LF          =  <ASCII LF, linefeed>           ; (     12,     10.)
SPACE       =  <ASCII SP, space>              ; (     40,     32.)
HTAB        =  <ASCII HT, horizontal-tab>     ; (     11,      9.)
<">         =  <ASCII quote mark>             ; (     42,     34.)
CRLF        =  CR LF
```

The same with the 1994 version of ABNF


```
LWSP-char=  SPACE / HTAB                ; semantics = SPACE
ALPHA     = %x41-5A / %x61-7A ; A-Z / a-z
BIT       = "0" / "1"
CHAR      = %x01-7F    ; any 7-bit US-ASCII character, excluding NUL
CR        = %x0D             ; carriage return
CRLF      = CR LF            ; Internet standard newline
CTL       = %x00-1F / %x7F; controls
DIGIT     = %x30-39          ; 0-9
DQUOTE    = %x22             ; " (Double Quote)
HEXDIG    = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
HTAB      = %x09             ; horizontal tab
LF        = %x0A             ; linefeed
LWSP      = *(WSP / CRLF WSP) ; linear white space (past newline)
OCTET     = %x00-FF          ; 8 bits of data
SP        = %x20
```


%d13 %x0D	he
	carriage return character.
b110 1	is the character with binary value 1101, which is a third way of specifying the carriage return character.
%x30- 39	means all characters with hexadecimal values from 30 to 39, which is the digits 0-9 in the ASCII character set.
%d13.10	is a short form for %d13 %d10, which is carriage return followed by line feed.

*:96 Overheads

Part 2ca: Extensible Markup Language (XML)

More about this course about Internet application protocols can be found at URL:

http://dsv.su.se/jpalme/internet-course/Int-app-prot-kurs.html

Last update: 00-03-24 17.24

HTML Example

```
<h2>False Pretences</h2>
<p><b>By: </b>Margaret Yorke<br>
<b>ISBN: </b>0-312-19975-9<br>
<b>Year: </b>1999</p>
```

XML Example

```
<book><author><surname>Yorke</surname>
<given-name>Margaret</given-name></author>
<title>False Pretences</title>
<isbn>0-312-19975-9</isbn>
<year>1999</year></book>
```

The difference between HTML and XML: In XML you can yourself decide which tags to use. In HTML, you can only use the built-in tags specified in HTML. In the example above, I used the tags `<book>`, `<author>`, `<surname>`, `<given-name>`, `<title>`, `<isbn>` and `<year>`. In another application, I could have chosen other tags.

By `combining` of XML with style sheets, you can still get the documented printed in the same way as if you had been using HTML.

Uses of XML

- For transport of information between data bases.
- For sending of information to be displayed to a user, just like with HTML.
- As a rather readable format in itself (except for encoding of special characters).
- For encoding of network operations, as an alternative to ABNF or ASN.1.

Restrictions of XML

- Binary data must be either encoded as BASE64 or sent outside of the XML document (like in HTML).
- A rather wordy format, but compression can reduce this.

Some acronyms

Standard Generalized Markup Language (SGML)

HTML and XML are both simplifications of SGML.

Application Program Interfaces (APIs)

Document Object Model (DOM) is an API för XML. Supported by newer web browsers.

Simple API for XML (SAX) standard for API to XML partsers. Streambased - the XML content is delivered in increments during its interpretation.

Style sheet languages

The same XML document can be shown in different formats, by using different style sheets.

eXtensible Style Sheet Language (XSL).

eXtensible Style Sheet Language Transformatsions (XSLT).

Cascading Style Sheet, level 1 och 2 (CSS1, CSS2).

Function	HTML	XML
Set of tags	Built-in, predefined set.	Every application can define its own element types and select their tags.
End tag	Not always required.	Always required.
Case sensitive	No, for example, <TITLE> and <title> are identical.	Yes, <TITLE> and <title> are different and <TITLE> must end with </TITLE>, not with </title>.
Coding errors	Often accepted	Not accepted
Support in web browsers	Yes.	Yes in some newer versions.
Text layout and style	HTML tags and style sheets.	Style sheets and XSLT transformation code.

Basics of the XML format

XML facility:	Example:
User-selected tags.	<book>, <songs>, <position> or whatever you need for your data.
Tags can have attributes.	<book author="Margaret Yorke" title="False Pretences">
Tags which have no embedded data can be closed in the opening tag.	<book author="Margaret Yorke" title="False Pretences"/> instead of <book author="Margaret Yorke" title="False Pretences"></book>
Tags can be nested.	<book><author>Margaret Yorke</author>...</book>
Tags must be closed.	Not correct: <book><author>Margaret Yorke</book>
Certain special character must be encoded.	<book title="The "queen"of Sheba"/>

Exercise 41

Here is an example of part of an e-mail heading according to current e-mail standards.

```
From: Nancy Nice <nnice@good.net>
To:   Percy Devil <pdevil@hell.net>
Cc:   Mary Clever <mclever@intelligence.net>,
      Rupert Happy <rhappy@fun.net>
```

How might the same information be encoded using XML?

XML is more strict than accepted HTML practice

HTML browsers accept many kinds of formally illegal HTML encodings. This is not allowed in XML. Examples:

```
Legal: <p>First paragraph.</p><p>Second paragraph</p>
Accepted: <p>First paragraph.<p>Second paragraph</p>

Legal: <b><i>Bold and Italics</i></b>
Accepted: <b><i>Bold and Italics</b></i>

Legal: <FONT COLOR="#FFFF66">
Accepted: <FONT COLOR=#FFFF66>
```

Tags are case-sensitive in XML

```
Illegal: <H1>Heading text</h1>
Legal: <H1>Heading text</H1>
```

White space is sometimes relevant in #PCDATA, but normalized in attributes

```
<CHRISTMAS>
     X
    XXX
   XXXXX
```

```
<CHRISTMAS FATHER="Donald
Duck">
```

is identical to

```
<CHRISTMAS FATHER="Donald Duck">
```

Document Type Definition (DTD)

An XML document may be connected with a document type definition. But this is not mandatory, you can send XML data without a DTD.

The DTD describes the allowed syntax, i.e. the tags and their allowed attributes.

Example of a DTD

```
<!ELEMENT book (author+)>
<!ATTLIST book
  title CDATA #REQUIRED
  year CDATA #IMPLIED >
<!ELEMENT author (#PCDATA)>
```

Example of XML using this DTD

```
<?xml version="1.0" ?>
<!DOCTYPE book SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/book.dtd">
<book title="False Pretences" year="1999" >
<author>Margaret York</author>
</book>
```



```
From: Nancy Nice <nnice@good.net>
To:   Percy Devil <pdevil@hell.net>
Cc:   Mary Clever <mclever@intelligence.net>,
      Rupert Happy <rhappy@fun.net>
```

Exercise 41 solution

```
<?xml version="1.0" ?>
<!DOCTYPE header SYSTEM "header.dtd">
<header>
  <from>
     <person>
        <user-friendly-name>Nancy Nice</user-friendly-name>
        <local-id>nnice</local-id><domain>good.net</domain>
     </person></from>
  <to>
     <person>
        <user-friendly-name>Percy Devil</user-friendly-name>
        <local-id>pdevil</local-id><domain>hell.net</domain>
     </person></to>
  <cc>
     <person>
        <user-friendly-name>Mary Clever</user-friendly-name>
        <local-id>mclever</local-id>
        <domain>intelligence.net</domain>
     </person><person>
        <user-friendly-name>rupert happy</user-friendly-name>
        <local-id>rhappy</local-id><domain>fun.net</domain>
     </person></cc>
</header>
```

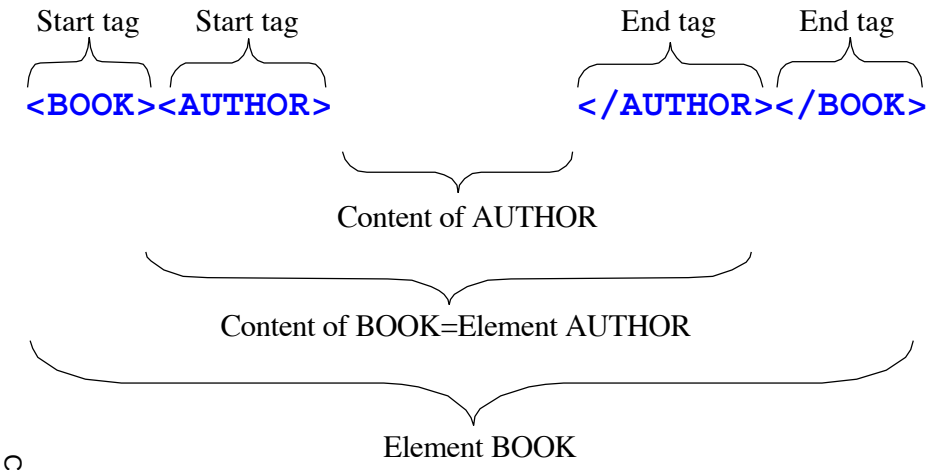

Relation between DTD and XML

Enviroment:	"ABNF"	"ASN.1"	"XML"
Language for specifying the encodings for a particular application.	ABNF	ASN.1	DTD (but not mandatory, and not as strong typing as in ASN.1)
Language used to actually encode data.	Text, often as a list of lines beginning with a name, a colon, followed by a value.	BER (or some other ASN.1 encoding rule)	XML

Special Character Encoding in XML

Reserved character	Predefined entity to use instead
<	<
&	&
>	>
'	'
"	"

ELEMENT and TAG



How to specify the DTD in the XML using it

```
<?xml version="1.0"?>
```

Specifies that this is XML-encoded data

```
<!DOCTYPE person
SYSTEM "person.dtd">
```

Specifies where to find the DTD. "Person.dtd" can be a complete URL, which gives a globally unique reference to this DTD.

```
<PERSON>
  <NAME>John Smith</NAME>
  <BIRTHYEAR>1941</BIRTHYEAR>
  <WAGE>57000</WAGE>
</PERSON>
```

Here comes the XML encoded according to this DTD.

DTD ELEMENT with subelements

`(a,b)` means the element `a` followed by the element `b`.

Example of a DTD

```
<!ELEMENT author (givenname,surname)>
<!ELEMENT givenname (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE author SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/author.dtd">
<author>
<givenname>Margaret</givenname>
<surname>York</surname>
</author>
```

Well-formed = correct XML, but need not have any DTD

Valid = correct XML and in accordance with a specified DTD

DTD ELEMENT with free text content

Example of a DTD

```
<!ELEMENT author (#PCDATA)>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE author SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/author.dtd">
<author>Margaret York</author>
```

Example 2 of XML using this DTD

```
<author>Text containing &lt; special markup &gt;</author>
```

Example 3 of XML using this DTD

```
<author>
<![CDATA[
Text containing < special markup > like & and " and '
]]>
</author>
```

DTD ELEMENT with subelements

(a?) means that the element a is repeated 0 or 1 times.

Example of a DTD

```
<!ELEMENT basic-family (father?,mother?,child*)>
<!ELEMENT father (#PCDATA)>
<!ELEMENT mother (#PCDATA)>
<!ELEMENT child (#PCDATA)>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE basic-family SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/basic-
family.dtd">
<basic-family>
<father>John</father>
<child>Eve</child>
<child>Peter</child>
</basic-family>
```


Exercise 42

Write a DTD for an XML-variant of the e-mail header in Exercise 41.

```
From: Nancy Nice <nnice@good.net>
To: Percy Devil <pdevil@hell.net>
Cc: Mary Clever
    <mclever@intelligence.net>,
    Rupert Happy <rhappy@fun.net>
```


DTD ELEMENT with subelements

(a*) means that a is repeated 0, 1 or more times.

Example of a DTD

```
<!ELEMENT family (father,mother,child*)>
<!ELEMENT father (#PCDATA)>
<!ELEMENT mother (#PCDATA)>
<!ELEMENT child (#PCDATA)>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE family SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/family.dtd">
<family>
<father>John</father>
<mother>Margaret</mother>
<child>Eve</child>
<child>Peter</child>
</family>
```


DTD ELEMENT with subelements

(a+) means that a is repeated 1 or more times.

Example of a DTD

```
<!ELEMENT child-family (father,mother,child+)>
<!ELEMENT father (#PCDATA)>
<!ELEMENT mother (#PCDATA)>
<!ELEMENT child (#PCDATA)>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE child-family SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/child-
family.dtd">
<child-family>
<father>John</father>
<mother>Margaret</mother>
<child>Eve</child>
<child>Peter</child>
</child-family>
```

Exercise 43

Specify DTD and an XML example for a protocol to send either a name (single string), a social-security number (another single string) or both.

Exercise 43 solution page A

DTD specification:	XML examples:
<!ELEMENT id (name | social-security-no | both)> <!ELEMENT both (name, social-security-no)> <!ELEMENT name (#PCDATA)> <!ELEMENT social-security-no (#PCDATA)>	**<?xml version="1.0" ?> <!DOCTYPE id SYSTEM "id.dtd"> <id><social-security-no>410201-1410</social-security-no></id>**
	<?xml version="1.0" ?> <!DOCTYPE id SYSTEM "id.dtd"> <id><both><name>Eliza Doolittle</name> <social-security-no>410201-1410</social-security-no></both></id>
	<?xml version="1.0" ?> <!DOCTYPE id SYSTEM "id.dtd"> <id><name>Eliza Doolittle</name> </id>

Exercise 43

Specify DTD and an XML example for a protocol to send either a name (single string), a social-security number (another single string) or both.

Exercise 42 solution

```
<!ELEMENT header (from, to?, cc?)>
<!ELEMENT from (person)>
<!ELEMENT to (person+)>
<!ELEMENT cc (person+)>
<!ELEMENT person (user-friendly-name,local-id,domain)>
<!ELEMENT user-friendly-name (#PCDATA)>
<!ELEMENT local-id (#PCDATA)>
<!ELEMENT domain (#PCDATA)>
```

Exercise 42

Write a DTD for an XML-variant of the e-mail header in Exercise 41.

```
From: Nancy Nice <nnice@good.net>
To: Percy Devil <pdevil@hell.net>
Cc: Mary Clever
    <mclever@intelligence.net>,
    Rupert Happy <rhappy@fun.net>
```


DTD ELEMENT with subelements

"|" means either-or "," means succession.

EMPTY (without parenthesis) means no contained data.

Example of a DTD

```
<!ELEMENT operations (((get | put),uri)*)>
<!ELEMENT get EMPTY>
<!ELEMENT put EMPTY>
<!ELEMENT uri (#PCDATA)>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE operations SYSTEM
"http://dsv.su.se/jpalme/internet-course/xml/operations.dtd">
<operations>
<get/><uri>http://cmc.dsv.su.se/file1</uri>
<get/><uri>http://cmc.dsv.su.se/file2</uri>
<put/><uri>http://cmc.dsv.su.se/file3</uri>
</operations>
```

Note: `<get/>` *is a short form for* `<get></get>`

Any Specification

The ANY specification (example:

<!ELEMENT miscellaneous ANY>)

allows any kind of un-specified XML content. This specification should in most cases be avoided, since it makes it difficult for software to check or interpret the content.

Exercise 43 solution page B

Note: The following will not work:

```
<!ELEMENT id ( name |
social-security-no |
(name, social-security-
no))>
<!ELEMENT name (#PCDATA)>
<!ELEMENT social-
security-no (#PCDATA)>
```

This will not work, because the receiving program will not be able to know, when it starts to scan <name> whether this is the first or the third branch of the choice.

DTD ELEMENT with XML attributes

Example of a DTD

```
<!ELEMENT book EMPTY>
<!ATTLIST book
  title CDATA #REQUIRED
  author CDATA 'anonymous'
  weight CDATA #IMPLIED
  format (paper-back | hard-back) 'paper-back'
>
```

Example 1 of XML using this DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE book SYSTEM
"http://www.dsv.su.se/~jpalme/internet-course/xml/book.dtd">
<book
  title="False Pretences"
  author="Margaret Yorke"
  format="hard-back"
/>
```


Operators in lists of subelements:

Code:	Explanation:
a, b	Mandatory a followed by mandatory b.
a | b	Either a or b.
a*	0, 1 or more occurences of a.
a+	1 or more occurences of a.
a?	0 or one occurences of a.

Type:	Example:	Description:
IDREFS	`<!ATTLIST author authorid ID #REQUIRED>` `<!ATTLIST book authorids IDREFS #REQUIRED>`	Similar to `IDREF`, but allows a list of more than one value. Needed in this example, if a book can have more than one author.

ENTITY	DTD text:
	`<!ELEMENT LOGO EMPTY>`
	`<!ATTLIST LOGO GIF-FILE ENTITY #REQUIRED>`
	`<!ENTITY DSV-LOGO SYSTEM "dsv-logo.gif">`
	XML text:
	`<LOGO GIF-FILE="DSV-LOGO"/>`
		This is one way to include binary data in an XML file, by referring to the URI of the binary data. Just like with `<IMG>` tags in HTML, the actual binary file is not included, just referenced.

Type:	Example:	Description:

ENTITIES	DTD text:
	`<!ELEMENT LOGO EMPTY>`
	`<!ATTLIST LOGO GIF-FILE ENTITIES #REQUIRED>`
	`<!ENTITY DSV-LOGO SYSTEM "dsv-logo.gif">`
	`<!ENTITY KTH-LOGO SYSTEM "kth-logo.gif">`
	XML text:
	`<LOGO GIF-FILE="DSV-LOGO KTH-LOGO"/>`
		A list of more than one entity.

NMTOKEN	`<!ATTLIST variable-name #NMTOKEN>`	A name, formatted like a variable name in a computer program. Useful when you use XML to generate source program code.
NMTOKENS	`<!ATTLIST variables #NMTOKENS>`	A list of names, similar as for NMTOKEN above.
NOTATION	`<!ATTLIST SPEECH PLAYER NOTATION ( MP3 | QUICKTIME ) #REQUIRED>`	The name of a non-XML encoding.

Default values for XML attributes

DTD term:	Example:	Description:
A single value within quotes at the end of the attribute.	`<!ATTLIST book binding (hardback | paperback) "hardback">`	This default value should be assumed if the attribute is not specified in the XML text.
#REQUIRED	`<!ATTLIST book binding (hardback | paperback) #REQUIRED>`	No default value is allowed, the attribute must always be specified in the XML text.
#IMPLIED	`<!ATTLIST book binding (hardback | paperback) #IMPLIED>`	No default value, but the attribute is not required. If the attribute is not given, this might mean that it is unknown or not valid.
#FIXED	`<!ATTLIST book binding (hardback | paperback) #FIXED "hardback">`	The XML can either contain this attribute or not, but if it is there, it must always have this particular value.

Types of XML attributes

Type:	Example:	Description:
CDATA	`<!ATTLIST book title #REQUIRED>`	Any character string.
A list of enumerated values	`<!ATTLIST book binding (hardback | paperback) "hardback">`	Restricted to the listed values only.
ID	`<!ATTLIST book entryno ID #REQUIRED>`	Gives a name to this particular element. No other element in the XML text can have the same name. Unique names on elements are useful in some cases for programs which manipulate the XML text.
IDREF	`<!ATTLIST author authorid ID #REQUIRED>` `<!ATTLIST book authorid IDREF #REQUIRED>`	Reference to the unique name, which was given to another element in the XML text. In the example, every element of type author has an `ID` authorid, and every element of type book has an `IDREF` referring to the `ID` of the element for the author of that book.

Exercise 44 solution

DTD specification:

```
<!ELEMENT movie (title,
  person+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT person EMPTY>
<!ATTLIST person
  name CDATA #REQUIRED
  role (actor | photographer |
  director | author |
  administrator)
  #IMPLIED
>
```

XML data:

```
<?xml version="1.0" ?>
<!DOCTYPE movie SYSTEM
  "movie.dtd">
<movie>
<title>
The Postman Always Rings
Twice</title>
<person name="Lana Turner"
   role="actor"/>
<person name="John Garfield"
   role="actor"/>
<person name="Tay Garnet"
   role="director"/>
<person name="James M. Cain"
   role="author"/>
</movie>
```

Exercise 44

Specify DTD and an XML example for a protocol to send a record describing a movie. The record contains a title and a list of people. Each person is identified by the attributes name, and optionally, the attribute role as either actor, photographer, director, author or administrator. As an XML example, use the movie "The Postman Always Rings Twice", directed by Tay Garnet based on a book by James M. Cain with leading actors Lana Turner and John Garfield.

ENTITIES

Built-in character entities

Example: `&quot; &`

Internal entities

You can add your own additional entity declarations to represent characters or sequences of characters. For example:

```
<!ENTITY KTH "Kungliga Tekniska Högskolan">
<DESCRIPTION>&KTH; is a technical university.</DESCRIPTION>
```

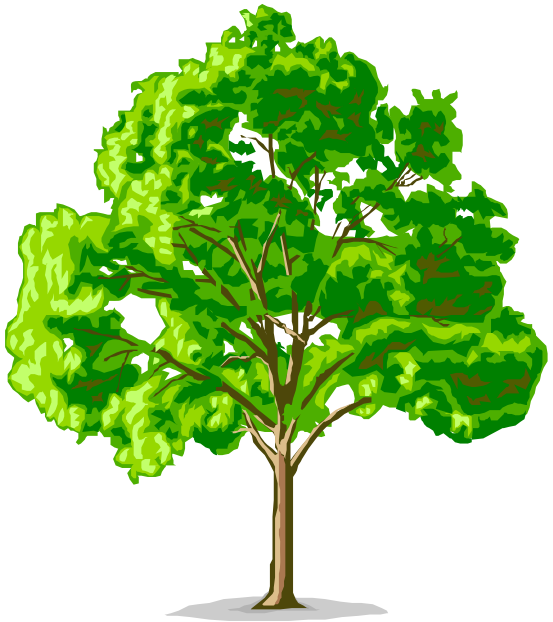
is identical to

```
<DESCRIPTION>Kungliga Tekniska Högskolan is a technical
university.</DESCRIPTION>
```

External entities

```
<!ENTITY polisvåld SYSTEM
"http://www.palme.nu/free/pv.html">

<!ENTITY comic SYSTEM
"http://www.palme.nu/comics/a-11.gif" NDATA GIF87A>
```

Elements versus attributes

```
<book><author><surname>
Yorke</surname><given-
name>Margaret</given-
name></author></book>
```

versus

```
<book author="Margaret
Yorke"/>
```

Elements are like a tree with branches, each branch can split into new branches.

Attributes are like leaves or fruits, they are the end point, cannot be split further. They also give some rudimentary type control.

Exercise 44

Specify DTD and an XML example for a protocol to send a record describing a movie. The record contains a title and a list of people. Each person is identified by the attributes name, and optionally, the attribute role as either actor, photographer, director, author or administrator. As an XML example, use the movie "The Postman Always Rings Twice", directed by Tay Garnet based on a book by James M. Cain with leading actors Lana Turner and John Garfield.