Introduction to Compiler Construction

Scanning: Preliminaries

Outline

Scanner

Scanner

The first step in compiling a program is to break it into a sequence of tokens

Scanner

The first step in compiling a program is to break it into a sequence of tokens

The program that does this is called a scanner (aka lexical analyzer)

Scanner

The first step in compiling a program is to break it into a sequence of tokens

The program that does this is called a scanner (aka lexical analyzer)

A scanner may be hand-crafted or generated

Scanner

Scanner

Example: the following *j--* program

HelloWorld.java

```
1  // Writes the message "Hello, World" to standard output.
2
3  import java.lang.System;
4
5  public class HelloWorld {
6      // Entry point.
7      public static void main(String[] args) {
8          System.out.println("Hello, World");
9      }
10 }
```

is broken into the sequence `import`, `java`, `.`, `lang`, `.`, `System`,`;`, `public`, `class`, `HelloWorld`, `{`, `...`, `;`, `}`, `}`

Scanner

Example: the following *j--* program

</> HelloWorld.java

```
1  // Writes the message "Hello, World" to standard output.
2
3  import java.lang.System;
4
5  public class HelloWorld {
6      // Entry point.
7      public static void main(String[] args) {
8          System.out.println("Hello, World");
9      }
10 }
```

is broken into the sequence `import`, `java`, `.`, `lang`, `.`, `System`,`;`, `public`, `class`, `HelloWorld`, `{`, `...`, `;`, `}`, `}`

Comments and whitespace characters are ignored by the scanner

Token Representation

Token Representation

Token representation consists of the token's kind, its image, and the line in which it occurs in the source program

Token Representation

Token representation consists of the token's kind, its image, and the line in which it occurs in the source program

In *j--*, each token is represented as a `TokenInfo` object

Token Representation

Token representation consists of the token's kind, its image, and the line in which it occurs in the source program

In *j--*, each token is represented as a `TokenInfo` object

Tokens are grouped into categories such as separators, operators, identifiers, literals, and reserved words (aka keywords)

Token Representation

Token Representation

Example (sequence of tokens in `HelloWorld.java`)

Kind	Image	Line	Category
IMPORT	"import"	3	keyword
<IDENTIFIER>	"java"	3	identifier
DOT	"."	3	separator
<IDENTIFIER>	"lang"	3	identifier
DOT	"."	3	separator
<IDENTIFIER>	"system"	3	identifier
SEMI	";"	3	separator
PUBLIC	"public"	5	keyword
CLASS	"class"	5	keyword
<IDENTIFIER>	"HelloWorld"	5	identifier
LCURLY	"{"	5	separator
PUBLIC	"public"	7	keyword
STATIC	"static"	7	keyword
VOID	"void"	7	keyword
<IDENTIFIER>	"main"	7	identifier
<LPAREN>	"("	7	separator
<IDENTIFIER>	"String"	7	identifier

Kind	Image	Line	Category
LBRACK	"["	7	separator
<RBRACK>	"]"	7	separator
<IDENTIFIER>	"args"	7	identifier
<RPAREN>	")"	7	separator
LCURLY	"{"	7	separator
<IDENTIFIER>	"System"	8	identifier
DOT	"."	8	separator
<IDENTIFIER>	"out"	8	identifier
DOT	"."	8	separator
<IDENTIFIER>	"println"	8	identifier
<LPAREN>	"("	8	separator
<STRING_LITERAL>	"Hello, World"	8	literal
<RPAREN>	")"	8	separator
<SEMI>	";"	8	separator
RCURLY	"}"	9	separator
RCURLY	"}"	10	separator
<EOF>	"<end of file>"	11	-