Chapter 2. Lexical Elements & Operators

Byoung-Tak Zhang
TA: Hanock Kwak

Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National University

http://bi.snu.ac.kr
The C System

- C Language
- Preprocessor
- Compiler
- (Standard) Library
- Linker
- Debugger
- Editor
C Compiler

- **Syntax of the language**
  - Rules for putting together words and punctuation to make correct, or legal, programs

- **Compiler**
  - A program that checks on the legality of C code
  - If errors exist, compiler prints error messages and stops
  - If there is no errors, compiler translates the code into object code
C Program

- C program
  - A sequence of characters that will be converted by a C compiler to object code
  - Compilers first collects the characters of the C program into tokens.
- 6 kinds of tokens
  - Keywords, Identifiers, Constants,
  - String constants, Operators, Punctuators
Characters used in a C Prog.

- Lowercase letters
  - a b c ...
- Uppercase letters
  - A B C ...
- Digits
  - 1 2 3 4 5 6 7 8 9
- Other special characters
  - + - * / = ( ) { } [ ] < > ‘ ” ! # % & _ | ^ ~ \ , ; : ?
- Whitespace characters
  - blank, newline, linefeed, tab, etc.
Comments

- **Comment**
  - A sequence of characters used for memo or documentation which are ignored by the compiler

- **Single line**
  - Starts with `//`

- **Closed Format**
  - Start with `/*` and ends with `*/`
  - Multiline of comments are possible.

```c
// blah blah I'm free from the codes
printf("hello"); // print hello !!

/* this is comment !!
blah blah blah
blah blah blah */
printf("hello"); /* print hello !! */
```
Comments

- Documentation
  - Codes are frequently shared with other people or communities.
  - Comments must provide clarity to the C source code.
  - Always write comments for clarity even if the codes are not shared. It will help debugging.

```c
// calculate the area for given radius and prints it
area = 3.14 * r * r;
printf("area: %.2f", area);
```
Keywords

- **Reserved words**
  - Keywords have a strict meaning as individual tokens in C.
  - Keywords cannot be redefined or used in other contexts.

<table>
<thead>
<tr>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
</tr>
<tr>
<td>break</td>
</tr>
<tr>
<td>case</td>
</tr>
<tr>
<td>char</td>
</tr>
<tr>
<td>const</td>
</tr>
<tr>
<td>continue</td>
</tr>
<tr>
<td>default</td>
</tr>
</tbody>
</table>
Identifiers

- A token composed of a sequence of letters, digits, and the special character _ (underscore)
- A letter or underscore must be the first character of an identifier
- Lowercase and uppercase are distinct.

< legal >
- knot
- _id101
- iamanidentifier2
- so_am_i

< illegal >
- not#2
- 101_south
- -plus
Identifiers

- Give unique names to objects in a program.
  - Objects: functions, variables, etc
- Keywords can be thought of as identifiers that are reserved to have special meaning. (Note: Keywords are not identifiers.)
- The identifier **main** is special.
  - It is identified as a main function (entry point) of a program by the compiler.
- Choose names that are meaningful!
  - **tax_rate**
Identifiers

```c
/*the distance of a marathon in kilometers*/
#include <stdio.h>
int main(void)
{
    int miles, yards;
    float kilometers;

    miles = 26;
    yards = 385;
    kilometers = 1.609 * (miles + yards / 1760.0);
    printf("A marathon is \%f kilometers.\n\n", kilometers);
    return 0;
}
```
Constants

- Integer constants
  - 0, -7, 17
- Floating constants
  - 1.0, 3.14159
- Character constants
  - ‘a’, ‘b’, ‘c’
- Special character constants
  - ‘\n’ (newline)
  - ‘\t’ (tab)
  - ‘\\’ (backslash \\)
- Backslash is the escape character. It has an alternative interpretation.
Constants

- **Integer constants**
  - Decimal integers 17
  - Octal integers 017
  - Hexadecimal integers 0x17
String Constants

- A sequence of characters enclosed in a pair of double-quote marks
  - “abc”
  - collected as a single token
  - ‘a’ and “a” are NOT the same.
  - /* “this is not a string” */
  - “ /* this is not a comment */ ”
Punctuators

- Punctuators
  - parentheses, braces, commas, semicolons, etc

- Operators and punctuators, along with white space, serve to separate language elements

```c
int main(void)
{
    int a, b = 2, c = 3;

    a = 17 * (b + c);
    ......
Operators

- Arithmetic Operators
  - +, -, *, /, %
  - % is a modulus operator
  - a % b is a remainder when a is divided by b

- Operators can be used to separate identifiers
  - a + b

- Some symbols have meanings that depend on context
  - printf(“%d”, a);  // % is used for formatted string
  - a = b%7;  // % is used as a modulus operator
Relational Operators

- relational operators
  - ==, !=, >, <, >=, <=
  - Result value of the relational operator is one (true) or zero (false).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater or equal than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal than</td>
</tr>
</tbody>
</table>

```java
if (a == b) {
    print("a, b are equal!!");
}
```
Logical Operators

- logical operators
  - &&, ||, !
  - Result value of the logical operator is one(true) or zero(false).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Negation</td>
</tr>
</tbody>
</table>

- $(1==1) \land (3 > 2)$: true
- $(1!=1) \land (3 > 2)$: false
- $!(0 > 1) \lor (2 < 1)$: true
- $0 \lor 0$: false
Increment and Decrement Operators

- **++**, **--**
  - Increase(decrease) an operand by one
  - Operands must be variables.

++a;  // same as a = a + 1
--b;  // same as b = b - 1

- Prefix and postfix

z = ++a;    a = a + 1;    z = a++;    z = a;
    a = a + 1;
Binary & Unary Operators

- **Binary**
  - Binary operators have two operands.
  - $a + b$, $100 / 65$

- **Unary**
  - Unary operators have a single operand.
  - $-2$, $++a$
Precedence and Associativity of Operators

- **Precedence**: order of an operator
  - $1 + 2 \times 3 \Leftrightarrow 1 + (2 \times 3)$

- **Associativity**: direction of an operator when precedence are same
  - $1 + 2 - 3 + 4 - 5 \Leftrightarrow (((1+2) - 3) + 4) - 5$
  - note that binary $+$ and binary $-$ have same precedence

- Parentheses can be used to clarify or change the order in which operators are performed.
  - $(1 + 2) \times 3$
Assignment Operator

\[ a = b + c; /* assignment statement */ \]

- is treated as an operator
  - Its precedence is lower than all others
  - “right to left” associativity
- variable = right_side
  - right_side is itself expression
  - The value of right_side is assigned to variable.
  - Result value of the assignment operator is the value of right_side.

\[
\begin{align*}
  a &= b = c = 3 &\rightarrow& a &= b = (c = 3) &\rightarrow& a &= b = 3 &\rightarrow& a &= 3 &\rightarrow& 3 \\
  z &= 1 + (x = 3 - 2) &\rightarrow& z &= 1 + (x = 1) &\rightarrow& z &= 1 + 1 &\rightarrow& z &= 2 &\rightarrow& 2
\end{align*}
\]
Assignment Operator

- Some complex assignment operators
  - `+=`, `-=``, `*=``, `/=``, `%=`
  - `a += b` is same as `a = a + b`

- Assignment could be used in the variable declaration statements.
  - `int a = 3;`
  - `float price = 3000, discount = 0.5;`