CENG 230
Introduction to C Programming
Week 2 – Overview of C

Sinan Kalkan

Some slides/content are borrowed from Tansel Dokeroglu, Nihan Kesim Cicekli.
How to study?

• Follow the lectures and the labs
• Read the textbook on a weekly basis
• Get your hands dirty
  • Do the exercises in front of the computer
Appointment

• No office hours. Make an appointment.
• Via email: skalkan@ceng.metu.edu.tr
• Office:
  Room B207,
  Department of Computer Engineering
• WWW:
  • http://kovan.ceng.metu.edu.tr/~sinan/
Program, Programming

Previously on CENG 230!

IMPLEMENTED

```c
int alice = 1;
int bob = 456;
int carol;
main(void)
{
    carol = alice*bob;
    printf("%d", carol);
}
```

PROGRAM
What is an algorithm?

An algorithm is a list that looks like

- STEP 1: Do something
- STEP 2: Do something
- STEP 3: Do something
- ...
- ...
- ...
- STEP N: Stop, you are finished

From “Invitation to Computer Science”
"I think you should be more explicit here in step two."

From “Invitation to Computer Science”
DIGITAL COMPUTATION

Turing Machine

Von Neumann Architecture
The Pascaline: One of the Earliest Mechanical Calculators
Difference engine

http://www.youtube.com/watch?v=0anIyVGeWOI
Programming the ENIAC
A computer

Devices

Gates

Transistors
A transistor

This circuit functions as a switch. In other words, based on the control voltage, the circuit either passes $V_{in}$ to output or not.
Everything in a PC is Binary...
well, almost ...

<table>
<thead>
<tr>
<th>States of a Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
NOT Gate

<table>
<thead>
<tr>
<th>X</th>
<th>( \bar{X} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ x \quad \text{---} \quad \bar{x} \]
AND gate

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X·Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
OR Gate

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X+Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Previously on CEng 220!
1-bit full-adder

\[ \text{A} \quad \text{B} \quad \text{CI} \]
\[ \begin{array}{c|ccc|c|c}
A & B & CI & S & CO \\
\hline
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 \\
\end{array} \]
N-bit Adder
Data Representation

• Based on 1s and 0s
  • So, everything is represented as a set of binary numbers

• We will now see how we can represent:
  • Integers: 3, 1234435, -12945 etc.
  • Floating point numbers: 4.5, 124.3458, -1334.234 etc.
  • Characters: /, &, +, -, A, a, ^, 1, etc.
  • ...

Previously on CENG 230!
Binary Representation of Numeric Information

Decimal numbering system

• Base-10
  • Each position is a power of 10
    \[3052 = 3 \times 10^3 + 0 \times 10^2 + 5 \times 10^1 + 2 \times 10^0\]

Binary numbering system

• Base-2
  • Uses ones and zeros
  • Each position is a power of 2
    \[(1101)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0\]
Decimal-to-binary Conversion

Divide the number until zero:

- $35 / 2 = 17 \times 2 + 1$
- $17 / 2 = 8 \times 2 + 1$
- $8 / 2 = 4 \times 2 + 0$
- $4 / 2 = 2 \times 2 + 0$
- $2 / 2 = 1 \times 2 + 0$

Therefore, 35 has the binary representation: 100011
IEEE 32-bit Floating-Point Number Representation

- Example: 12.375
  - The digits before the dot:
    - \((12)_{10} \rightarrow (1100)_2\)
  - The digits after the dot:
    - 1\textsuperscript{st} Way: 0.375 \rightarrow 0\times\frac{1}{2} + 1\times\frac{1}{4} + 1\times\frac{1}{8} \rightarrow 011
    - 2\textsuperscript{nd} Way: Multiply by 2 and get the integer part until 0:
      - 0.375 \times 2 = 0.750 = 0 + 0.750
      - 0.750 \times 2 = 1.50 = 1 + 0.50
      - 0.50 \times 2 = 1.0 = 1 + 0.0
  - \((12.375)_{10} = (1100.011)_2\)
  - NORMALIZE: \((1100.011)_2 = (1.100011)_2 \times 2^3\)
  - Exponent: 3, adding 127 to it, we get 1000 0010
  - Fraction: 100011
  - Then our number is: 0 \textcolor{red}{10000010} \textcolor{red}{100011}00000000000000000
Binary Representation of Textual Information (cont’d)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>00110000</td>
<td>0</td>
<td>0x30</td>
<td>0x0030</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>00110001</td>
<td>1</td>
<td>0x31</td>
<td>0x0031</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>00110010</td>
<td>2</td>
<td>0x32</td>
<td>0x0032</td>
<td>2</td>
</tr>
<tr>
<td>51</td>
<td>00110011</td>
<td>3</td>
<td>0x33</td>
<td>0x0033</td>
<td>3</td>
</tr>
<tr>
<td>52</td>
<td>00110100</td>
<td>4</td>
<td>0x34</td>
<td>0x0034</td>
<td>4</td>
</tr>
<tr>
<td>53</td>
<td>00110101</td>
<td>5</td>
<td>0x35</td>
<td>0x0035</td>
<td>5</td>
</tr>
<tr>
<td>54</td>
<td>00110110</td>
<td>6</td>
<td>0x36</td>
<td>0x0036</td>
<td>6</td>
</tr>
<tr>
<td>55</td>
<td>00110111</td>
<td>7</td>
<td>0x37</td>
<td>0x0037</td>
<td>7</td>
</tr>
<tr>
<td>56</td>
<td>00111000</td>
<td>8</td>
<td>0x38</td>
<td>0x0038</td>
<td>8</td>
</tr>
<tr>
<td>57</td>
<td>00111001</td>
<td>9</td>
<td>0x39</td>
<td>0x0039</td>
<td>9</td>
</tr>
<tr>
<td>58</td>
<td>00111010</td>
<td>:</td>
<td>0x3A</td>
<td>0x003A</td>
<td>:</td>
</tr>
<tr>
<td>59</td>
<td>00111011</td>
<td>;</td>
<td>0x3B</td>
<td>0x003B</td>
<td>;</td>
</tr>
<tr>
<td>60</td>
<td>00111100</td>
<td>&lt;</td>
<td>0x3C</td>
<td>0x003C</td>
<td>&lt;</td>
</tr>
<tr>
<td>61</td>
<td>00111101</td>
<td>=</td>
<td>0x3D</td>
<td>0x003D</td>
<td>=</td>
</tr>
<tr>
<td>62</td>
<td>00111110</td>
<td>&gt;</td>
<td>0x3E</td>
<td>0x003E</td>
<td>&gt;</td>
</tr>
<tr>
<td>63</td>
<td>00111111</td>
<td>?</td>
<td>0x3F</td>
<td>0x003F</td>
<td>?</td>
</tr>
<tr>
<td>64</td>
<td>01000000</td>
<td>@</td>
<td>0x40</td>
<td>0x0040</td>
<td>@</td>
</tr>
<tr>
<td>65</td>
<td>01000001</td>
<td>A</td>
<td>0x41</td>
<td>0x0041</td>
<td>A</td>
</tr>
<tr>
<td>66</td>
<td>01000010</td>
<td>B</td>
<td>0x42</td>
<td>0x0042</td>
<td>B</td>
</tr>
</tbody>
</table>

ASCII 7 bits long
Unicode 16 bits long

Partial listings only!
Computer Organization

Logical organization of computer

- Input
- CPU (ALU)
- Memory
- Output
- Secondary Storage
Memory and Cache (continued)

• RAM (Random Access Memory)
  Often called memory, primary memory
  
  • Memory made of addressable “cells”
  
  • Cell size is 8 bits
    • Nowadays, it is 32 or 64 bits.
  
  • All memory cells accessed in equal time
  
  • Memory address
    • Unsigned binary number $N$ long
    • Address space is then $2^N$ cells
Previously on CEng 230!

- **Machine language**
- **Assembly language**
- **Programming languages such as C++, Java**
- **Pseudocode**
- **English, Spanish, Japanese, ...**

- **Low-level languages** (closely related to the hardware)
- **High-level languages** (more removed from details of the hardware)
- **Natural languages** (not related to the hardware)
main:
pushq %rbp
movq %rsp, %rbp
movl alice(%rip), %edx
movl bob(%rip), %eax
imull %edx, %eax
movl %eax, carol(%rip)
movl $0, %eax
leave
ret
alice:
.long 123

bob:
.long 456

int alice = 123;
int bob = 456;
int carol;
main(void)
{
    carol = alice*bob;
}
How are languages implemented
C language development environment
Bugs, Errors

• Syntax Errors
  Area = 3.1415 * R * R
  Area = 3.1415 x R x R

• Run-time Errors

```python
>>> def SqrtDelta(a,b,c):
    return sqrt(b*b - 4*a*c)

>>> print SqrtDelta(1,3,1)
2.2360679774997898
>>> print SqrtDelta(1,1,1)
ValueError: math domain error
```
Bugs, Errors

• Logical Errors

\[ \text{root}_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \]

>>> root1 = (- b + sqrt(b*b - 4*a*c)) / 2*a

• Design Errors

\[ x^3 + ax^2 + bx + c = 0 \]

\[ \text{root}_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \]
History of C

- Developed by Denis M. Ritchie at AT&T Bell Labs in 1972 as a systems programming language
- Used to develop UNIX
- Used to write modern operating systems
- Hardware independent (portable)

• Standardization
  - Many slight variations of C existed, and were incompatible
  - Committee formed to create a "unambiguous, machine independent" definition
  - Standard created in 1989, updated in 1999
Fig. 2.1: `fig02_01.c`  
A first program in C */

```c
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
    printf( "Welcome to C!\n" );
    return 0; /* indicate that program ended successfully */
} /* end function main */
```

Welcome to C!

**Fig. 2.1** A first program in C.
Notes

• The lectures notes:
  
  http://www.kovan.ceng.metu.edu.tr/~sinan/ceng230/
  
  • You can also just google my name → Follow “Courses” → “Ceng 230”

• Location:
  
  • Starting from “5 March, 2015”, lectures will be held in BMB-1 (in Computer Engineering Dept.)

• Midterm:
  
  • 28 April, 2015 at 17:40
Today

• Continue with the overview of C
• Introduction of the basic concepts

• Let me collect the assignment
Variables and identifiers
C has the following basic built-in datatypes.

- `int`
- `float`
- `double`
- `char`

**TABLE 2.4 Type double Constants (real numbers)**

<table>
<thead>
<tr>
<th>Valid double Constants</th>
<th>Invalid double Constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.14159</td>
<td>150 (no decimal point)</td>
</tr>
<tr>
<td>0.005</td>
<td>.12345e (missing exponent)</td>
</tr>
<tr>
<td>12345.0</td>
<td>15e−0.3 (0.3 is invalid exponent)</td>
</tr>
<tr>
<td>15.0e−04 (value is 0.0015)</td>
<td></td>
</tr>
<tr>
<td>2.345e2 (value is 234.5)</td>
<td>12.5e.3 (.3 is invalid exponent)</td>
</tr>
<tr>
<td>1.15e−3 (value is 0.00115)</td>
<td>34,500.99 (comma is not allowed)</td>
</tr>
<tr>
<td>12e+5  (value is 1200000.0)</td>
<td></td>
</tr>
</tbody>
</table>
**Valid Identifiers**

letter_1, letter_2, inches, cent, CENT_PER_INCH, Hello, variable

**TABLE 2.2 Invalid Identifiers**

<table>
<thead>
<tr>
<th>Invalid Identifier</th>
<th>Reason Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>lLetter</td>
<td>begins with a letter</td>
</tr>
<tr>
<td>double</td>
<td>reserved word</td>
</tr>
<tr>
<td>int</td>
<td>reserved word</td>
</tr>
<tr>
<td>TWO*FOUR</td>
<td>character * not allowed</td>
</tr>
<tr>
<td>joe's</td>
<td>character ' not allowed</td>
</tr>
</tbody>
</table>

int1 and Int1 are not the same identifiers/variables
### Keywords

<table>
<thead>
<tr>
<th>auto</th>
<th>double</th>
<th>int</th>
<th>struct</th>
</tr>
</thead>
<tbody>
<tr>
<td>break</td>
<td>else</td>
<td>long</td>
<td>switch</td>
</tr>
<tr>
<td>case</td>
<td>enum</td>
<td>register</td>
<td>typedef</td>
</tr>
<tr>
<td>char</td>
<td>extern</td>
<td>return</td>
<td>union</td>
</tr>
<tr>
<td>const</td>
<td>float</td>
<td>short</td>
<td>unsigned</td>
</tr>
<tr>
<td>continue</td>
<td>for</td>
<td>signed</td>
<td>void</td>
</tr>
<tr>
<td>default</td>
<td>goto</td>
<td>sizeof</td>
<td>volatile</td>
</tr>
<tr>
<td>do</td>
<td>if</td>
<td>static</td>
<td>while</td>
</tr>
</tbody>
</table>

*Keywords added in C99*

_Bool _Complex _Imaginary inline restrict

**Fig. 2.15** | C’s keywords.
Basic Input/Output in C
Output

- `printf` (format string, var1, var2, ...)
  - Format string contains:
    - d,i: integers
    - f: float, double
    - e: float, double in exponential notation
    - c: character
    - s: string
Input

- `scanf(format string, &var1, &var2, ... )`
  - `var1, var2, ..:` variables!
  - Format string contains:
    - `d,i:` integers
    - `f:` float, double
    - `e:` float, double in exponential notation
    - `c:` character
    - `s:` string
/* Fig. 2.3: fig02_03.c */
/* Printing on one line with two printf statements */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
    printf( "Welcome " );
    printf( "to C!\n" );
    return 0; /* indicate that program ended successfully */
} /* end function main */

Fig. 2.3 | Printing on one line with two printf statements | (Part 1 of 2.)

Welcome to C!
/* Fig. 2.4: fig02_04.c
Printing multiple lines with a single printf */

#include <stdio.h>

/* function main begins program execution */
int main( void )
{
    printf( "Welcome\n\nto\nC!\n" );
    return 0; /* indicate that program ended successfully */
} /* end function main */

Fig. 2.4 | Printing multiple lines with a single printf.
<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\n</code></td>
<td>Newline. Position the cursor at the beginning of the next line.</td>
</tr>
<tr>
<td><code>\t</code></td>
<td>Horizontal tab. Move the cursor to the next tab stop.</td>
</tr>
<tr>
<td><code>\a</code></td>
<td>Alert. Sound the system bell.</td>
</tr>
<tr>
<td><code>\\</code></td>
<td>Backslash. Insert a backslash character in a string.</td>
</tr>
<tr>
<td><code>\&quot;</code></td>
<td>Double quote. Insert a double-quote character in a string.</td>
</tr>
</tbody>
</table>

**Fig. 2.2** | Some common escape sequences.
```c
int first, second;
scanf("%d%d", &first, &second);

double miles; /* distance in miles */
scanf("%lf", &miles);
```
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
    int integer1; /* first number to be input by user */
    int integer2; /* second number to be input by user */
    int sum; /* variable in which sum will be stored */

    printf( "Enter first integer\n" ); /* prompt */
    scanf( "%d", &integer1 ); /* read an integer */

    printf( "Enter second integer\n" ); /* prompt */
    scanf( "%d", &integer2 ); /* read an integer */

    sum = integer1 + integer2; /* assign total to sum */

    printf( "Sum is %d\n", sum ); /* print sum */

    return 0; /* indicate that program ended successfully */
} /* end function main */
Variable names such as `integer1`, `integer2` and `sum` actually correspond to locations in the computer’s memory. Every variable has a name, a `type` and a `value`.

In the addition program of Fig. 2.5, when the statement (line 13)

```c
scanf( "%d", &integer1 ); /* read an integer */
```

is executed, the value typed by the user is placed into a memory location to which the name `integer1` has been assigned. Suppose the user enters the number 45 as the value for `integer1`. The computer will place 45 into location `integer1` as shown in Fig. 2.6.

![Fig. 2.6](image)

**Fig. 2.6** | Memory location showing the name and value of a variable.

Whenever a value is placed in a memory location, the value replaces the previous value in that location; thus, placing a new value into a memory location is said to be `destructive`. 
Specifying the format of an integer value displayed by a C program is fairly easy. You simply add a number between the % and the d of the %d placeholder in the printf format string. This number specifies the field width—the number of columns to use for the display of the value. The statement

printf("Results: %3d meters = %4d ft. %2d in.\n",
       meters, feet, inches);

indicates that 3 columns will be used to display the value of meters, 4 columns will be used for feet, and 2 columns will be used for inches (a number between 0 and 11). If meters is 21, feet is 68, and inches is 11, the program output will be

Results: 21 meters = 68 ft. 11 in.

<table>
<thead>
<tr>
<th>Value</th>
<th>Format</th>
<th>Displayed Output</th>
<th>Value</th>
<th>Format</th>
<th>Displayed Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>234</td>
<td>%4d</td>
<td>234</td>
<td>-234</td>
<td>%4d</td>
<td>-234</td>
</tr>
<tr>
<td>234</td>
<td>%5d</td>
<td>234</td>
<td>-234</td>
<td>%5d</td>
<td>-234</td>
</tr>
<tr>
<td>234</td>
<td>%6d</td>
<td>234</td>
<td>-234</td>
<td>%6d</td>
<td>-234</td>
</tr>
<tr>
<td>234</td>
<td>%1d</td>
<td>234</td>
<td>-234</td>
<td>%2d</td>
<td>-234</td>
</tr>
</tbody>
</table>
### TABLE 2.16 Formatting Type double Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Format</th>
<th>Displayed Output</th>
<th>Value</th>
<th>Format</th>
<th>Displayed Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.14159</td>
<td>%5.2f</td>
<td>3.14</td>
<td>3.14159</td>
<td>%4.2f</td>
<td>3.14</td>
</tr>
<tr>
<td>3.14159</td>
<td>%3.2f</td>
<td>3.14</td>
<td>3.14159</td>
<td>%5.1f</td>
<td>3.1</td>
</tr>
<tr>
<td>3.14159</td>
<td>%5.3f</td>
<td>3.142</td>
<td>3.14159</td>
<td>%8.5f</td>
<td>3.14159</td>
</tr>
<tr>
<td>.1234</td>
<td>%4.2f</td>
<td>0.12</td>
<td>-.006</td>
<td>%4.2f</td>
<td>-0.01</td>
</tr>
<tr>
<td>-.006</td>
<td>%8.3f</td>
<td>-0.006</td>
<td>-.006</td>
<td>%8.5f</td>
<td>-0.00600</td>
</tr>
<tr>
<td>-.006</td>
<td>%3.3f</td>
<td>-0.006</td>
<td>-3.14159</td>
<td>%4.4f</td>
<td>-3.1416</td>
</tr>
</tbody>
</table>
Operators and Expressions
```c
printf( "Welcome to \%d", (3/2) );
```

Output is : 1
<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Operation(s)</th>
<th>Order of evaluation (precedence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Parentheses</td>
<td>Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they’re evaluated left to right.</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>Evaluated second. If there are several, they’re evaluated left to right.</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>Evaluated last. If there are several, they’re evaluated left to right.</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>Evaluated last. If there are several, they’re evaluated left to right.</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>Evaluated last. If there are several, they’re evaluated left to right.</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>Evaluated last. If there are several, they’re evaluated left to right.</td>
</tr>
<tr>
<td>Arithmetic Operator</td>
<td>Meaning</td>
<td>Examples</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
<td>5 + 2 is 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.0 + 2.0 is 7.0</td>
</tr>
<tr>
<td>−</td>
<td>subtraction</td>
<td>5 - 2 is 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.0 - 2.0 is 3.0</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>5 * 2 is 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.0 * 2.0 is 10.0</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>5.0 / 2.0 is 2.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 / 2 is 2</td>
</tr>
<tr>
<td>%</td>
<td>remainder</td>
<td>5 % 2 is 1</td>
</tr>
</tbody>
</table>
**TABLE 2.10**  Results of Integer Division

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>15</td>
<td>=</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>=</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>=</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>=</td>
<td>5</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>=</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>-3</td>
<td>=</td>
<td>-5</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>=</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>is</td>
<td>undefined</td>
</tr>
</tbody>
</table>

**TABLE 2.11**  Results of % Operation

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>%</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>%</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>%</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>%</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>%</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>%</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>%</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>%</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>%</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>%</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>-7</td>
<td>%</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>%</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>is</td>
<td>undefined</td>
</tr>
<tr>
<td>Mathematical Formula</td>
<td>C Expression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. $b^2 - 4ac$</td>
<td>$b * b - 4 * a * c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. $a + b - c$</td>
<td>$a + b - c$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. $\frac{a + b}{c + d}$</td>
<td>$(a + b) / (c + d)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. $\frac{1}{1 + x^2}$</td>
<td>$1 / (1 + x * x)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. $a \times -(b + c)$</td>
<td>$a * -(b + c)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rules for Evaluating Expressions

a. Parentheses rule: All expressions in parentheses must be evaluated separately. Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.

b. Operator precedence rule: Operators in the same expression are evaluated in the following order:

- unary +, − first
- *, /, % next
- binary +, − last

c. Associativity rule: Unary operators in the same subexpression and at the same precedence level (such as + and −) are evaluated right to left (right associativity). Binary operators in the same subexpression and at the same precedence level (such as + and −) are evaluated left to right (left associativity).
Fig. 2.11 | Order in which a second-degree polynomial is evaluated.
**Evaluation of a Second-Degree Polynomial**

To develop a better understanding of the rules of operator precedence, let’s see how C evaluates a second-degree polynomial.

\[
y = a \times x \times x + b \times x + c;
\]

6 1 2 4 3 5

---

**Step 1.** \( y = 2 \times 5 \times 5 + 3 \times 5 + 7; \) *(Leftmost multiplication)*

2 \times 5 = \boxed{10}

**Step 2.** \( y = 10 \times 5 + 3 \times 5 + 7; \) *(Leftmost multiplication)*

10 \times 5 = \boxed{50}

**Step 3.** \( y = 50 + 3 \times 5 + 7; \) *(Multiplication before addition)*

3 \times 5 = \boxed{15}

**Step 4.** \( y = 50 + 15 + 7; \) *(Leftmost addition)*

50 + 15 = \boxed{65}

**Step 5.** \( y = 65 + 7; \) *(Last addition)*

65 + 7 = \boxed{72}

**Step 6.** \( y = 72 \) *(Last operation—place 72 in y)*

---

**Fig. 2.11** | Order in which a second-degree polynomial is evaluated.
Homework

• Write a C code that calculates the roots of the following equation:

\[ ax^2 + bx + c = 0 \]

• Your program should read \( a, b \) and \( c \) from standard input.

• Bring the print-out of the C code to the next lecture.

• Hint: You will need to use the sqrt() function defined in the math.h library. Google it for more detail.