EECS 10: Computational Methods in Electrical and Computer Engineering
Lecture 5

Brian Demsky
bdemsky@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine
Lecture 5: Overview

- Arithmetic Operations in C
- Basic Types
- Type conversion
  - explicit
  - implicit
- Types in Expressions
- Arithmetic computation
  - Example `Arithmetic.c`
- Shift Operators
- Conditional Operator
- Formatting Output
Arithmetic Operations in C

- Arithmetic Operators
  - parentheses
  - unary plus, minus
  - multiplication, division, modulo
  - addition, subtraction
  - shift left, shift right

- Evaluation order of expressions
  - usually left to right
  - by operator precedence
    - ordered as in table above (higher operators are evaluated first)

- Arithmetic operators are available
  - for integer types: all
  - for floating point types: all except %, <<, >>
Basic Types in C

- Integer types
  - `char` Character, e.g. ‘a’, ‘b’, ‘1’, ‘*’
    - typical range `[-128, 127]`
  - `short int` Short integer, e.g. –7, 0, 42
    - typical range `[-32768, 32767]`
  - `int` Integer, e.g. –7, 0, 42
    - typical range `[-2147483648, 2147483647]`
  - `long int` Long integer, e.g. –991, 91, 1231
    - typical range `[-2147483648, 2147483647]`
  - `long long int` Very long integer, e.g. 1234511
    - typical range
      `[-9223372036854775808, 9223372036854775807]`
- Integer types can be
  - `signed` negative and positive values (and 0)
  - `unsigned` positive values only (and 0)
Basic Types in C

- Floating point types
  - `float` Floating point with single precision
    - Example `3.5f`, `-0.234f`, `10E8f`
  - `double` Floating point with double precision
    - Example `3.5`, `-0.23456789012`, `10E88`
  - `long double` Floating point with high precision
    - Example `12345678.123456E123l`

- Floating point values are in many cases *approximations* only!
  - Storage size of floating point values is fixed
  - Many values can only be represented as approximations
  - Example: `1.0/3.0 = .333...`
Type Conversion

- Explicit Type Conversion
  - types can be explicitly converted to other types, by use of the type cast operator:
    \((\text{type}) \ \text{expression}\)
  - the target type is named explicitly in parentheses before the source expression
  - Examples:
    - \text{Float} = (\text{float}) \ \text{LongInt}
      - converts the \text{long int} type into a \text{float} type
    - \text{Integer} = (\text{int}) \ \text{Double}
      - converts the \text{double} type into an \text{int} type
      - any fractional part is truncated!
    - \text{Char} = (\text{char}) \ \text{LongLongInt}
      - converts the \text{long long int} type into a \text{char} type
      - any out-of-range values are silently cut off!
Type Conversion

• Implicit Type Conversion
  • Type promotion
    • integral promotion
      – unsigned or signed char is promoted to unsigned or signed int before any operation
      – unsigned or signed short is promoted to unsigned or signed int before any operation
    • floating-point promotion
      – float is promoted to double before any operation
  • binary arithmetic operators are defined only for same types
    – Examples:
      – the smaller type is converted to the larger type
        » ShortInt * LongInt results in a long int type
        » LongDouble * Float results in a long double type
  • Type coercion
    • most types are automatically converted to expected types
    • Example: Double = Float, or Char = LongInt
Types in Expressions

- Expressions are composed of constants, variables and operators, each of which has an associated type.

Example:

```c
short int s;
int i;
long int l;
float f;
double d;

l = 2 * s + i * f - 0.5 * d;
```
Example Program

- Program example:
  - Task: Write a C program that exercises arithmetic computation by use of different types and operators!
  - The program should compute the following equations:
    - Quotient of sums:
      \[ q = \frac{a + b}{c + d} \]
    - Remainder:
      \[ r = \text{rem}(2^n / 7) \]
    - Assume that \( a, b, c, d, \) and \( n \) are whole numbers.
Program example: **Arithmetic.c** (part 1/3)

```c
/* Arithmetic.c: arithmetic expressions */
/* */
/* author: Rainer Doemer */
/* */
/* modifications: */
/* 10/06/04 RD initial version */

#include <stdio.h>

/* main function */
int main(void)
{
    /* variable definitions */
    int    a, b, c, d, n;
    double p, q, r;

    ...
```
*/ input section */
printf("Please enter the value for integer a: ");
scanf("%d", &a);
printf("Please enter the value for integer b: ");
scanf("%d", &b);
printf("Please enter the value for integer c: ");
scanf("%d", &c);
printf("Please enter the value for integer d: ");
scanf("%d", &d);
printf("Please enter the value for integer n: ");
scanf("%d", &n);
```

...  

/* computation section */
q = ((double)(a + b)) / ((double)(c + d));
r = (1<<n) % 7;

/* output section */
printf("The value for the quotient q is %f.\n", q);
printf("The value for the remainder r is %f.\n", r);

/* exit */
return 0;
} /* end of main */

/* EOF */
```
Example Program

- Example session: **Arithmetic.c**

```bash
% vi Arithmetic.c
% gcc Arithmetic.c -Wall -ansi -o Arithmetic
% ls -l
```
```
total 20
-rwx------ 1 doemer faculty  7344 Oct  6 08:42 Arithmetic*
-rw------- 1 doemer faculty   1154 Oct  6 08:37 Arithmetic.c
% Arithmetic
Please enter the value for integer a: 5
Please enter the value for integer b: 6
Please enter the value for integer c: 7
Please enter the value for integer d: 8
Please enter the value for integer n: 9
The value for the quotient q is 0.733333.
The value for the remainder r is 1.000000.
%
```
Shift Operators

- **Left-shift operator:** \( x << n \)
  - shifts \( x \) in binary representation \( n \) times to the left
  - multiplies \( x \) \( n \) times by 2
  - Examples
    - \( 2x = x << 1 \)
    - \( 4x = x << 2 \)
    - \( x \times 2^n = x << n \)
    - \( 2^n = 1 << n \)

- **Right-shift operator:** \( x >> n \)
  - shifts \( x \) in binary representation \( n \) times to the right
  - divides \( x \) \( n \) times by 2
  - Examples
    - \( x / 2 = x >> 1 \)
    - \( x / 4 = x >> 2 \)
    - \( x / 2^n = x >> n \)
Conditional Operator

• Conditional evaluation of values in expressions
• Question-mark operator:
  \( \text{test} \ ? \ \text{true-value} : \ \text{false-value} \)
  • evaluates the \text{test}
  • if \text{test} is true, then the result is \text{true-value}
  • otherwise, the result is \text{false-value}
• Examples:
  • \((4 < 5) \ ? \ (42) : (4+8)\) evaluates to \text{42}
  • \((2==1+2) \ ? \ (x) : (y)\) evaluates to \text{y}
  • \((x < 0) \ ? \ (-x) : (x)\) evaluates to \text{abs}(x)
## Conversion Specifiers for Basic Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Print specifier</th>
<th>Scan specifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>long double</code></td>
<td><code>%Lf</code></td>
<td><code>%Lf</code></td>
</tr>
<tr>
<td><code>double</code></td>
<td><code>%f</code></td>
<td><code>%lf</code></td>
</tr>
<tr>
<td><code>float</code></td>
<td><code>%f</code></td>
<td><code>%f</code></td>
</tr>
<tr>
<td><code>unsigned long long</code></td>
<td><code>%llu</code></td>
<td><code>%llu</code></td>
</tr>
<tr>
<td><code>long long</code></td>
<td><code>%lld</code></td>
<td><code>%lld</code></td>
</tr>
<tr>
<td><code>unsigned long</code></td>
<td><code>%lu</code></td>
<td><code>%lu</code></td>
</tr>
<tr>
<td><code>long</code></td>
<td><code>%ld</code></td>
<td><code>%ld</code></td>
</tr>
<tr>
<td><code>unsigned int</code></td>
<td><code>%u</code></td>
<td><code>%u</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>%d</code></td>
<td><code>%d</code></td>
</tr>
<tr>
<td><code>short</code></td>
<td><code>%hd</code></td>
<td><code>%hd</code></td>
</tr>
<tr>
<td><code>char</code></td>
<td><code>%c</code></td>
<td><code>%c</code></td>
</tr>
</tbody>
</table>
Formatted Output

- Formatted output using `printf()`
  - standard format specifiers for integral values
    - `unsigned long long %llu`
    - `long long %lld`
    - `unsigned long %lu`
    - `long %ld`
    - `unsigned int %u`
    - `int %d`
    - `short %hd`
  - standard format specifiers for floating point values
    - `long double %Lf`
    - `double %f`
    - `float %f`
Formatted Output

- Detailed formatting sequence for integral values
  - % flags width length conversion
- flags
  - (none) standard formatting (right-justified)
  - - left-justified output
  - + leading plus-sign for positive values
  - 0 leading zeros
- field width
  - (none) minimum number of characters needed
  - integer width of field to be filled with output
- length modifier
  - (none) int type
  - h short int type
  - l long int type
  - ll long long int type
- conversion specifier
  - d signed decimal value
  - u unsigned decimal value
  - o (unsigned) octal value
  - x (unsigned) hexadecimal value using characters 0–9, a–f
  - X (unsigned) hexadecimal value using characters 0–9, A–F
Formatted Output

- Detailed formatting sequence for floating-point values
  - `% flags width precision length conversion`

- **flags**
  - (none) standard formatting (right-justified)
  - – left-justified output
  - + leading plus-sign for positive values
  - 0 leading zeros

- **field width**
  - (none) minimum number of characters needed
  - integer width of field to be filled with output

- **precision**
  - (none) default precision (e.g. 6)
  - .int number of digits after decimal point (for f, e, or E), maximum number of significant digits (for g, or G)

- **length modifier**
  - (none) float or double type
  - L long double type

- **conversion specifier**
  - f standard floating-point notation (fixed-point)
  - e or E exponential notation using (e or E)
  - g or G standard or exponential notation (using e or E)
• Program example: Formatting.c (part 1/2)

```c
/* Formatting.c: formatted output demo */
/* author: Rainer Doemer */
/* modifications: */
/* 10/19/04 RD initial version */

#include <stdio.h>

/* main function */

int main(void)
{
    /* output section */
    printf("42 formatted as |%%d|: |%d|\n", 42);
    printf("42 formatted as |%%8d|: |%8d|\n", 42);
    printf("42 formatted as |%%08d|: |%08d|\n", 42);
    printf("42 formatted as |%%x|: |%x|\n", 42);
    printf("42 formatted as |%%o|: |%o|\n", 42);
    ...
```

Formatted Output
• Program example: Formatting.c (part 2/2)

```c
...
printf("\n");
printf("123.456 formatted as |%%f|: \%f\n", 123.456);
printf("123.456 formatted as |%%e|: \%e\n", 123.456);
printf("123.456 formatted as |%%g|: \%g\n", 123.456);
printf("123.456 formatted as |%%12.4f|: \%12.4f\n", 123.456);
printf("123.456 formatted as |%%12.4e|: \%12.4e\n", 123.456);
printf("123.456 formatted as |%%12.4g|: \%12.4g\n", 123.456);

/* exit */
return 0;
}

/* EOF */
```
**Example session:** Formatting.c

```plaintext
% vi Formatting.c
% gcc Formatting.c -o Formatting -Wall -ansi
% Formatting
42 formatted as |%d|: |42|
42 formatted as |%8d|: |   42|
42 formatted as |%-8d|: |  42|
42 formatted as |%+8d|: | +42|
42 formatted as |%08d|: |00000042|
42 formatted as |%x|: | 2a|
42 formatted as |%o|: |  52|

123.456 formatted as |%f|: |123.456000|
123.456 formatted as |%e|: |1.234560e+02|
123.456 formatted as |%g|: | 123.456|
123.456 formatted as |%12.4f|: | 123.4560|
123.456 formatted as |%12.4e|: |  1.2346e+02|
123.456 formatted as |%12.4g|: |   123.5|
```